Analytics Meta Learning

Zi Yang

May 2017

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Dr. Eric Nyberg, Carnegie Mellon University
Dr. Jaime Carbonell, Carnegie Mellon University
Dr. Ravi Starzl, Carnegie Mellon University
Dr. John M. Prager, IBM T. J. Watson Research Center
Dr. ChengXiang Zhai, University of Illinois at Urbana-Champaign

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.

Copyright © 2017 Zi Yang
Keywords: Analytics Meta Learning, Configuration Space Exploration, Intelligent Information System, Question Answering, Decision Support
This thesis is dedicated to my parents
Abstract

Data analytics is the process to analyze massive data to discover useful knowledge and make conclusions about the information, to improve predictions and support decision making. Solving analytics tasks requires more than just merely applying analysis algorithms, instead it combines high-level decision making and low-level process execution, which makes it more difficult than performing individual analyses or analysis steps. As more and more analysis components have become available nowadays, it has been more challenging and time-consuming than ever to quickly design an extensible architecture, and effectively and efficiently compose an information system from these components in order to achieve a desired or optimal level of performance on a given analytics task. In this thesis, we study the problem of analytics meta learning and the solution to assist, if not replace, humans the design, planning, and evaluation in the development of intelligent information systems for analytics tasks.

We formally define the problem of analytics meta learning and propose a solution framework that consists of three steps: analytics procedure definition, analysis component construction, and analytics space exploration. From the theoretical perspective, we focus on design and rigorous analysis of algorithms to extract procedural knowledge for analytics procedure definition, and Bayesian and MDP modeling and budgeted policy optimization for analytics space exploration. We implement a software architecture framework that enables analytics meta learning, and we leverage the framework to solve real-world analytics tasks on three domains of problems, including general biomedical question answering task, pharmaceutical decision support task, and product recommendation task, and empirically study the performance of the proposed algorithms.

The ECD framework has been successfully used in developing intelligent information systems across various domains since its first release in mid-2012, which includes TREC Genomics passage retrieval task, BioASQ QA task, target validation for drug discovery, etc. We have seen positive outcomes from several case studies, e.g., the analytics meta learning methodology has helped create the biomedical factoid and list question answering system that won the BioASQ QA 2015 and 2016 challenges. The framework has also been employed in the related courses domestically and abroad for educational purposes since 2012. The goal of the thesis is to formally define and systematically study the analytics meta learning problem through theoretical analyses and empirical case studies on a wide range of applications.
Acknowledgments

I would like to first thank my advisor Eric Nyberg for his guidance and support of my PhD research. I would also like to thank my thesis committee members Jaime Carbonell, Ravi Starzl, John Prager, and ChengXiang Zhai for insightful feedback and comments. We give special thanks to Ying Li, Xing Yang, Jeremy Gore, Venus So, Jorg Hakenberg, Raul Rodríguez-Esteban, James Cai and other current or former colleagues at Roche Innovation Center New York and global locations for their support of my research. I also need to thank my collaborators, colleagues, and friends, with whom I have spent the most memorable six years: Elmer Garduno, Yan Fang, Avner Maiberg, Eric Riebling, Ting-Hao Huang, Leo Boytsov, Di Wang, Niloy Gupta, Xiangyu Sun, Alkesh Patel, Rui Liu, Jun Araki, Collin McCormack, Chi Zhang, Di Xu, Xuezi Zhang, Ran Zhao, Trisha Black, Tao Guo, Martin Van Velsen, Naoki Orii, Suyoun Kim, Khyathi Chandu, Parag Agrawal, Salvador Medina, Qianru Zhu, Soumya Batra, Duo Ding, Chenying Hou, Rueqiong Wang, Vijay Viswanathan, Lavanya Viswanathan, Jack Montgomery, Tom Yu, Yi Song, Chalalai Chaihirunkarn, George Paliouras, Kartik Mandaville, Ming Sun, Yibin Lin, Carlos Ramirez, Haohan Wang, Srinivasa Phani Gadde, Kenton Murray, Yuchen Zhang, Andrew Fandrianto, Yubin Kim, Prasanna Kumar Mathukumar, Prasad Gajare, Hugo Rodrigues, Jun In Lee, Jary Yang Du, Matthias Grabmair, David Adamson, Justin Chiu, Zhilin Yang, Shriphani Palakodety, Petr Baudis, Victor Chahuneau, Hakim Sid Ahmed, Navendu Jain, Jay Yoon Lee, Waleed Ammar, Weston Feely, Ana Cristina Mendes, Nisarga Markandaih, Zhengzhong Liu, Phani Gadde, Zhou Yu, Rodolfo Vega, Anastasia Krithara, Manaal Faruqui, Wang Ling, Jieru Hu, Luis Marujo, Boyan Li, Junchen Jiang, Lu Jiang, Bo Lei, Lingpeng Kong, Guangyu Xia, Miaomiao Wen, Guoqing Zheng, Shuguang Han, Jiwei Li, Yulia Tsvetkov, Ni Lao, Shou-I Yu, Le Zhao, Wenyi Wang, Adams Wei Yu, Wanli Ma, Yajie Mao, Zhuyin Dai, Nan Li, Yun-Nung Chen, Grace Hui Yang, Aaron Qiaochu Li, Wei Wei, Yun Wang, Chenyan Xiong, Junjie Hu, Kay Chen, Yifei Ma, Xuezhi Wang, Mengwen Liu, Bhuwan Dhingra, and Cheng Hao, Tianle Huang, Hongwen Kang, Guokun Lai, Bin Liu, Kai Liu, Jingzhou Liu, Xueze Ma, Nanyun Peng, Haoruo Peng, Xin Qian, Yanan Qian, Jing Sun, Xu Wang, Yipei Wang, Ruosi Wang, Zhuo Wang, Qizhe Xie, Pengtao Xie, Diyi Yang, Pengcheng Yin, Hao Zhang, Tieying Zhang, Shikun Zhao, Huimin Zhao, Yu Zhao, Zeyu Zheng, Yue Zhou, Qiang Zhu, and Sivaraman Balakrishnan, Ramnath Balasubramanyan, Pallavi Baljekar, Juan Caicedo, Qicong Chen, Ran Chen, Xinlei Chen, Zihang Dai, Yuntian Deng, Rui Du, Nicholas FauciGlia, Yingkai Gao, Yu Gong, Anika Gupta, Ye Hu, Zhenhao Hua, Yi-Fei Huang, Dongyeop Kang, Anagha Kulkarni, Zhenzhong Lan, Chu-Cheng Lin, Yiu-Chang Lin, Hanxiao Liu, Bo Ma, Vipul Mascarenhas, Meghana Mehanha, Seungwhan Moon, Amos Ng, Yuanchi Ning, Kai Ren, Kotaro Sakamoto, Hideki Shima, Yanchuan Sim, Derry Wijaya, Haibo Wang, Min Wang, Yi-Chia Wang, Fei Xia, Guang Xiang, Ruochen Xu, Xiaohua Yan, Xingyu Yan, Mengda Yang, William Yang Wang, Reyyan Yeniterzi, Yuan Zhou, Haiyi Zhu.
# Contents

1 Introduction ......................................................... 1
   1.1 Thesis Statement ............................................. 3
   1.2 Thesis Overview .............................................. 4
   1.3 Achievements .................................................. 6
   1.4 Thesis Structure .............................................. 9

2 Analytics Meta Learning ........................................ 13
   2.1 Related Work .................................................. 14
      2.1.1 Execution Process Representation ....................... 14
      2.1.2 Decision Process Representation ......................... 16
      2.1.3 Information Processing Architecture Frameworks ......... 17
      2.1.4 Limitations of Specification Languages .................. 19
   2.2 Terminology ................................................... 20
   2.3 Problem Definition ........................................... 25
   2.4 Solution Framework .......................................... 28
   2.5 Conclusion .................................................... 31

3 Procedural Knowledge Discovery ............................... 33
   3.1 Related Work .................................................. 33
   3.2 Procedural Knowledge Base ................................ 34
      3.2.1 Semi-Structured Procedural Knowledge Base ........... 35
   3.3 Application: Search Task Suggestion ......................... 36
      3.3.1 Modeling & Solution ..................................... 38
      3.3.2 Data Preparation ......................................... 39
      3.3.3 Experiment Settings ...................................... 42
      3.3.4 Experimental Results .................................... 43
   3.4 Procedural Knowledge Extraction from Search Activities ... 45
      3.4.1 Modeling & Solution ..................................... 46
      3.4.2 Data Preparation ......................................... 49
      3.4.3 Experiment Settings ...................................... 49
      3.4.4 Experimental Results .................................... 50
   3.5 Conclusion ..................................................... 52
## List of Figures

1.1 Use case diagram of data analytics ............................................... 2
1.2 Analytics meta learning problem lies in the intersection .................. 3
1.3 Architecture overview of the QUADS framework ............................ 8
1.4 Architecture overview of the CSE framework ................................. 9
1.5 Thesis structure ........................................................................... 12

2.1 Development process using analytics meta learning solution framework .... 28

3.1 Subproblems in search task suggestion using procedural knowledge and automatic procedural knowledge base construction from search activities ............ 47
3.2 An example of the parallel corpus construction process ...................... 48

4.1 Modeling cost and benefit for analytics space exploration .................. 58
4.2 Variation of real-valued configurations for all components in the TREC Genomics experiment .......................................................... 66
4.3 Estimation of likelihood and posterior of cost at the end of the TREC Genomics experiment .......................................................... 68
4.4 Estimation of likelihood and posterior of cost when the TREC Genomics experiment completed 1% of total execution tasks .................. 69
4.5 Estimation of likelihood and posterior of component-level benefit when 20% of total execution tasks were completed in the TREC Genomics experiment .... 70
4.6 Generalizability of systems with various resampling methods in the TREC Genomics experiment .......................................................... 71

5.1 Classification of policy optimization methods used in this chapter and their solutions .......................................................... 84
5.2 A snapshot of evaluating a configuration space ................................... 89
5.3 Learning curves of value-based methods ......................................... 103
5.4 Varying learning rate function for value-based methods ....................... 103
5.5 Varying option selection strategy for value-based methods ................. 104
5.6 Learning curves of off-policy tabular methods ................................ 107
5.7 Learning curves of off-policy linear methods ................................... 107
5.8 Varying damping factor of off-policy tabular methods ....................... 108
5.9 Varying damping factor of off-policy linear methods ....................... 108
5.10 Varying initial padding length of off-policy tabular methods ............. 109
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.11</td>
<td>Varying initial padding length of off-policy linear methods</td>
<td>109</td>
</tr>
<tr>
<td>5.12</td>
<td>Varying option selection strategy for off-policy methods</td>
<td>111</td>
</tr>
<tr>
<td>5.13</td>
<td>Learning curves of policy-based methods</td>
<td>111</td>
</tr>
<tr>
<td>5.14</td>
<td>Varying learning rate function for policy-based methods</td>
<td>113</td>
</tr>
<tr>
<td>5.15</td>
<td>Varying option selection strategy for policy-based methods</td>
<td>113</td>
</tr>
<tr>
<td>5.16</td>
<td>Learning curves of model-based methods</td>
<td>116</td>
</tr>
<tr>
<td>5.17</td>
<td>Varying learning rate function for model-based methods</td>
<td>116</td>
</tr>
<tr>
<td>5.18</td>
<td>Varying option selection strategy for model-based methods</td>
<td>117</td>
</tr>
<tr>
<td>5.19</td>
<td>Varying damping factor in value estimation for model-based methods</td>
<td>118</td>
</tr>
<tr>
<td>5.20</td>
<td>Varying number of samples in value estimation for model-based methods</td>
<td>118</td>
</tr>
<tr>
<td>5.21</td>
<td>Varying $k$ in value estimation for model-based methods</td>
<td>119</td>
</tr>
<tr>
<td>5.22</td>
<td>Learning curves of various temporal difference methods</td>
<td>119</td>
</tr>
<tr>
<td>5.23</td>
<td>Varying learning rate of various temporal difference methods</td>
<td>120</td>
</tr>
<tr>
<td>5.24</td>
<td>Varying the difficulty of the problem via input mean</td>
<td>121</td>
</tr>
<tr>
<td>5.25</td>
<td>Varying the diversity of the problem via input variance</td>
<td>122</td>
</tr>
<tr>
<td>5.26</td>
<td>Varying the length of the pipeline</td>
<td>123</td>
</tr>
<tr>
<td>5.27</td>
<td>Varying the number of components in a single phase</td>
<td>124</td>
</tr>
<tr>
<td>5.28</td>
<td>Varying the number of components in multiple phases</td>
<td>125</td>
</tr>
<tr>
<td>5.29</td>
<td>Varying the average task performance of the components</td>
<td>126</td>
</tr>
<tr>
<td>5.30</td>
<td>Uncertainty of the task performance of the components in a single phase</td>
<td>126</td>
</tr>
<tr>
<td>5.31</td>
<td>Varying the uncertainty of the task performance of the components</td>
<td>127</td>
</tr>
<tr>
<td>5.32</td>
<td>Varying budget for TQ strategies w/ independent option selection</td>
<td>132</td>
</tr>
<tr>
<td>5.33</td>
<td>Varying budget for FA strategies w/ independent option selection</td>
<td>132</td>
</tr>
<tr>
<td>5.34</td>
<td>Varying budget for M+ strategies w/ independent option selection</td>
<td>133</td>
</tr>
<tr>
<td>5.35</td>
<td>Varying budget for TQ strategies w/ two-step option selection</td>
<td>133</td>
</tr>
<tr>
<td>5.36</td>
<td>Varying budget for FA strategies w/ two-step option selection</td>
<td>135</td>
</tr>
<tr>
<td>5.37</td>
<td>Varying budget for M+ strategies w/ two-step option selection</td>
<td>135</td>
</tr>
<tr>
<td>5.38</td>
<td>Varying scaling factor $c_{UCB}$ for strategies w/ UCB-E</td>
<td>136</td>
</tr>
<tr>
<td>5.39</td>
<td>Varying scaling factor $c_{UCB}$ for strategies w/ random/UCB1</td>
<td>136</td>
</tr>
<tr>
<td>5.40</td>
<td>Varying scaling factor $c_{UCB}$ for strategies w/ random/UCB-E</td>
<td>137</td>
</tr>
<tr>
<td>5.41</td>
<td>The runtime performance mean of the components in a single phase</td>
<td>138</td>
</tr>
<tr>
<td>5.42</td>
<td>Performance of TQ strategies w/ independent option selection when $c_{RPM}$ varies</td>
<td>138</td>
</tr>
<tr>
<td>5.43</td>
<td>Performance of FA strategies w/ independent option selection when $c_{RPM}$ varies</td>
<td>139</td>
</tr>
<tr>
<td>5.44</td>
<td>Performance of M+ strategies w/ independent option selection when $c_{RPM}$ varies</td>
<td>139</td>
</tr>
<tr>
<td>5.45</td>
<td>Performance of TQ strategies w/ two-step option selection when $c_{RPM}$ varies</td>
<td>140</td>
</tr>
<tr>
<td>5.46</td>
<td>Performance of FA strategies w/ two-step option selection when $c_{RPM}$ varies</td>
<td>140</td>
</tr>
<tr>
<td>5.47</td>
<td>Performance of M+ strategies w/ two-step option selection when $c_{RPM}$ varies</td>
<td>141</td>
</tr>
<tr>
<td>5.48</td>
<td>Performance of TQ strategies w/ independent option selection for different $\rho_{TRM,RPM}$</td>
<td>142</td>
</tr>
<tr>
<td>5.49</td>
<td>Performance of FA strategies w/ independent option selection for different $\rho_{TRM,RPM}$</td>
<td>142</td>
</tr>
<tr>
<td>5.50</td>
<td>Performance of M+ strategies w/ independent option selection for different $\rho_{TRM,RPM}$</td>
<td>143</td>
</tr>
<tr>
<td>5.51</td>
<td>Performance of TQ strategies w/ 2-step option selection for different $\rho_{TRM,RPM}$</td>
<td>144</td>
</tr>
<tr>
<td>5.52</td>
<td>Performance of FA strategies w/ 2-step option selection for different $\rho_{TRM,RPM}$</td>
<td>144</td>
</tr>
</tbody>
</table>
## List of Tables

1.1 Comparison of analytics meta learning experiments and case studies in Chapters 5–9 5

2.1 Comparison between supervised learning and analytics meta learning 29

3.1 Semi-structured procedural knowledge base feature definition 36

3.2 Averaged number and percentage of related queries suggested by Google or Bing or issued by the users subsequently in the same session that are mentioned by a task description in wikiHow 41

3.3 Search task suggestion results 44

3.4 The non-Word features that contribute the most to search task suggestion problem 45

3.5 Search task suggestion results 45

3.6 Top 4 suggested search tasks from the proposed STS system and commercial search engines for example search tasks 46

3.7 Task summary generation results 50

3.8 Task explanation generation results 51

3.9 The non-Word features that contribute the most to the automatic procedural knowledge base construction problem 52

3.10 Summary and explanation generation result for automatic procedural knowledge base construction 52

3.11 Suggested subtasks from the proposed APKBC system for example tasks 53

4.1 Posterior distributions for each of $\gamma^c_t$, $\gamma_t$ and $\beta^c_t$, $\beta_t$ in modeling analytics space exploration 60

4.2 Summary of analysis components in TREC Genomics QA system 63

4.3 Experimental settings and comparison with TREC Genomics 2006 participants 64

4.4 Variation of nominal configurations for all components in the TREC Genomics experiment 65

4.5 Result on TREC Genomics 2007 data set 71

5.1 Comparison of configuration space exploration problem with other related problems 74

5.2 Comparison of configuration space exploration strategies with related problems 75
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.3 Markov decision process modeling for configuration evaluation and</td>
<td>87</td>
</tr>
<tr>
<td>configuration space exploration. We compare the components, inputs,</td>
<td></td>
</tr>
<tr>
<td>intermediate objects, state and action definitions, policy space $M/\pi$, transition $T$, reward and terminal re-</td>
<td></td>
</tr>
<tr>
<td>ward functions $R$</td>
<td></td>
</tr>
<tr>
<td>5.4 Markov decision process modeling for persistedence configuration</td>
<td>90</td>
</tr>
<tr>
<td>space exploration.</td>
<td></td>
</tr>
<tr>
<td>5.5 Description of simulated problem set and configuration space</td>
<td>100</td>
</tr>
<tr>
<td>5.6 Description of value-based methods</td>
<td>102</td>
</tr>
<tr>
<td>5.7 Results of value-based methods</td>
<td>102</td>
</tr>
<tr>
<td>5.8 Description of off-policy methods</td>
<td>105</td>
</tr>
<tr>
<td>5.9 Results of off-policy methods</td>
<td>106</td>
</tr>
<tr>
<td>5.10 Descriptions of policy-based methods</td>
<td>110</td>
</tr>
<tr>
<td>5.11 Results of policy-based methods</td>
<td>112</td>
</tr>
<tr>
<td>5.12 Descriptions of model-based methods</td>
<td>114</td>
</tr>
<tr>
<td>5.13 Results of model-based methods</td>
<td>115</td>
</tr>
<tr>
<td>5.14 Description of simulated problem set and configuration space</td>
<td>129</td>
</tr>
<tr>
<td>5.15 Results of budgeted learning strategies with independent option</td>
<td>131</td>
</tr>
<tr>
<td>selection</td>
<td></td>
</tr>
<tr>
<td>5.16 Results of budgeted learning strategies with 2-step option selection</td>
<td>134</td>
</tr>
<tr>
<td>7.1 Comparison of biomedical question answering shared tasks: TREC</td>
<td>169</td>
</tr>
<tr>
<td>Genomics and BioASQ</td>
<td></td>
</tr>
<tr>
<td>7.2 Snippet retrieval features</td>
<td>177</td>
</tr>
<tr>
<td>7.3 Retrieval result reranking via relevant classification features</td>
<td>179</td>
</tr>
<tr>
<td>7.4 Answer type prediction features</td>
<td>183</td>
</tr>
<tr>
<td>7.5 Answer scorers</td>
<td>185</td>
</tr>
<tr>
<td>7.6 Collective answer reranking features</td>
<td>186</td>
</tr>
<tr>
<td>7.7 Yes/No question answering features</td>
<td>187</td>
</tr>
<tr>
<td>7.8 Partial official BioASQ 3B and 4B evaluation results and ranks</td>
<td>191</td>
</tr>
<tr>
<td>among systems</td>
<td></td>
</tr>
<tr>
<td>7.9 Error categories and occurrences for factoid questions in BioASQ</td>
<td>192</td>
</tr>
<tr>
<td>3B</td>
<td></td>
</tr>
<tr>
<td>7.10 Manual comparison of relevant content reranking methods</td>
<td>197</td>
</tr>
<tr>
<td>7.11 Manual comparison of factoid, list and yes/no QA methods</td>
<td>198</td>
</tr>
<tr>
<td>7.12 Description of configuration space for relevant snippet retrieval</td>
<td>200</td>
</tr>
<tr>
<td>7.13 Results of budgeted learning strategies with independent component</td>
<td>201</td>
</tr>
<tr>
<td>selection for BioASQ Phase A (relevant snippet retrieval task)</td>
<td></td>
</tr>
<tr>
<td>7.14 Results of budgeted learning strategies with random phase selection for BioASQ Phase A (relevant snippet retrieval task)</td>
<td>204</td>
</tr>
<tr>
<td>7.15 Description of configuration space for factoid question answering</td>
<td>209</td>
</tr>
<tr>
<td>7.16 Results of budgeted learning strategies with independent component selection for BioASQ Phase B (factoid QA task)</td>
<td>210</td>
</tr>
<tr>
<td>7.17 Results of budgeted learning strategies with random phase selection for BioASQ Phase B (factoid QA task)</td>
<td>213</td>
</tr>
<tr>
<td>8.1 Yes/no QA analysis components</td>
<td>225</td>
</tr>
<tr>
<td>8.2 Description of configuration space for target validation task</td>
<td>226</td>
</tr>
</tbody>
</table>
8.3 Results of budgeted learning strategies with independent component selection for target validation ........................................... 227
8.4 Results of budgeted learning strategies with random phase selection for target validation ........................................... 229
9.1 Description of analytics space for phone recommendation task ........................................... 237
9.2 Results of budgeted learning strategies with independent component selection for product recommendation ........................................................................................................... 238
9.3 Results of budgeted learning strategies with random phase selection for phone recommendation ........................................................................................................... 241
10.1 Performance persistence database ........................................... 249
A.1 SQLite schema for experiment & evaluation persistence providers ........................................... 255
Chapter 1

Introduction

Data analytics is the process to analyze massive data to discover useful knowledge and make conclusion about the information, to improve predictions and support decision making [103]. A specific data analytics use case can be defined by, at simplest, a task description representing the user’s information seeking or decision making need. The following questions are examples of real-life information needs that have been reported from biomedical experts during their research or diagnosis: [125] [221] (Examples 1 to 5) and from real online shoppers (Example 6):

(Ex. 1) How many TAp73 isoforms have been identified in humans?
(Ex. 2) What is the function of the mamalian gene Irg1?
(Ex. 3) Find the most frequent carbapenemases found in Enterobacteriaceae.
(Ex. 4) Can Alzheimer’s disease related miRNAs be detected in patients’ blood?
(Ex. 5) Is AKT1 directly involved in the breast cancer and can be a suitable target?
(Ex. 6) Choose an Android phone that best suits your needs.

A data analytics procedure to answer these questions and make their own conclusion includes discovering and evaluating all the relevant, but possibly inconsistent or even contradictory, conclusions from over 20 million biomedical research publications[1] as well as various manually curated databases. Solving analytics tasks requires more than just merely applying analysis algorithms, instead it combines high-level decision making and low-level process execution, which makes it more difficult than performing individual analyses or analysis steps. As a result, in order to support users’ decision making, an intelligent information system that solves an analytics task usually requires a systematic design and implementation based on a model of human cognition, which usually consists of multiple internal processing steps and integrates various internal and external analysis steps, including algorithms, knowledge bases or other resources from heterogeneous sources, as well as other analytics engines for sub analytics tasks as subroutine. A use case diagram is depicted in Figure [1]. In addition to the primary use case that the user submits an analytics task to the system and awaits response, we further consider that the system can leverage human decision knowledge by working actively with the user or a domain expert to tackle the analytics task.

As more and more analysis components have become available nowadays, it has been more challenging and time-consuming than ever to quickly design an extensible architecture, and effectively and efficiently compose an information system from these components in order to achieve a desired or optimal level of performance on a given analytics task. First, since the modules might be developed by people of wide range of skills and abilities, experienced core team members, novices, or unmet open source contributors, their quality might vary. Furthermore, the interoperability between data analysis algorithms and resources is also hardly guaranteed by their original developers. In the meantime, the requirement and expected performance of the information systems keep changing as new information demands arise from their users.

Proposing practices and tools to guide and automate composition, optimization, and evaluation has been a focus of information system design and development. To support integration of data analysis components, information processing architecture frameworks, such as UIMA [56, 57], define an information system in terms of a set of basic elements (such as typed object, module, configuration, resource, workflow, etc.) described via standardized representation. Explicit specification of preconditions and effects has also enabled automatic composition of applications from components [163]. However, when we apply these approaches to constructing an information system for an arbitrary analytics task, most approaches would be possible to produce a functioning system, but not sufficiently intelligent to perform the analytics task, due to the limitations of expressiveness in the specification languages and the ignorance of analytics task objectives during optimization. Similarly, most test automation frameworks [132, 184] can only serve as a sanity checker for intelligent information systems, since their performance on a given analytics task is usually binary “pass or fail”, rather than judged subjectively on a graded or continuous scale.

We believe that there is great demand for a comprehensive solution to assist, if not replace, humans the design, planning, and evaluation in the development of intelligent information systems for analytics tasks, to guarantee continuous improvement towards to the task objective. In this thesis, we envision a software framework, independent of specific analytics tasks, that extends a
Figure 1.2: Analytics meta learning problem lies in the intersection of data analytics, software architecture and supervised learning.

The biggest challenge in the development of the intelligent architecture layer is to design an algorithm that optimizes the analytics procedure that combines lower-level analysis components and analytics, simultaneously as these analysis components and analytics are optimizing and updating their own models individually and locally. We refer this problem as analytics meta learning, in analogy to traditional learning problem in data analytics, where a predefined and configured training process is statically performed, when new training data arrive, to update a model. The proposed work lies in the intersection of data analytics, software architecture and supervised learning, as shown in Figure 1.2.

1.1 Thesis Statement

The thesis introduces the analytics meta learning methodology to standardize, accelerate, and automate construction and optimization of analytics procedures for arbitrary data analytics tasks. Through both theoretical studies of each respective problem – analytics procedure definition, analysis component construction, and analytics space exploration – and empirical studies using real-
world analytics tasks, we claim the proposed methodology and the algorithms can achieve the desired level of performance in order to assist, if not replace, human in the design, planning, and evaluation during the development of intelligent information systems for analytics tasks, to guarantee continuous improvement towards to the task objective.

1.2 Thesis Overview

We study the problem from both theoretical and empirical perspectives. We start the theoretical study with giving formal definitions to the subjects we study in the thesis, which include analytics task, analytics procedure, information system (analytics engine), etc. We then formally define the analytics meta learning problem and present the solution framework, which consists of three major steps: analytics procedure definition, analysis component construction, and analytics space exploration.

Designing and describing analytics procedures are not easy tasks, especially for the users who know little about the problem domain but solving the tasks involves decision processes. As a subproblem of analytics meta learning, we introduce that problem of procedural knowledge discovery, which aims to automatically discover procedural knowledge – the knowledge exercised in the performance of some task, i.e. what actions should be performed (execution processes) and what factors should be considered to achieve some goal (decision processes) [6, 66] – and then construct analytics procedures for arbitrary analytics tasks accordingly. In this thesis, we explore a few human-generated data sources, such as on-line community how-to guides, search logs, etc., and study the canonicalization of representation to facilitate analysis component construction and execution. As oppose to domain/task-specific analysis component construction step, we study the general analytics space exploration problem, which operates directly on any arbitrary predefined and constructed analytics space. We first design a specialized hierarchical Bayesian model and propose to find the optimal configuration via a greedy algorithm for the stochastic knapsack problem. Then, we study the optimization problem in analytics meta learning – analytics space exploration, which operates directly on any arbitrary predefined and constructed analytics space. We model the evaluation and exploration processes as Markov Decision Processes (MDPs), and model a (budgeted) analytics space exploration problem as a (budgeted) policy optimization problem. We incorporate and extend the state-of-the-art policy optimization algorithms proposed for reinforcement learning problems, including tabular value estimation, function approximation, policy gradient, off-policy and model-based methods., A total of 62 non-budgeted exploration strategies and 40 budgeted exploration strategies are compared using a simulated configuration space.

In the empirical or experimental part of the thesis, we take analytics meta learning into action. We focus on exemplifying the proposed methodology to tackle analytics tasks and reporting the performance of the proposed analytics meta learning methods in terms of the learning curve throughout the entire learning process, as well as the performance of their resulting systems, in terms of task-specific metrics and measurements. We first present the principles of design and implementation of analytics meta learning software framework, and accordingly develop an open source toolkit UIMA-ECD that supports defining and specifying analysis components and analytics procedures, and further implements various policy optimization algorithms to conduct analytics
Table 1.1: Comparison of analytics meta learning experiments and case studies in Chapters 5–9

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Task</th>
<th>Space configuration space</th>
<th>Size of the analytics space</th>
<th>Budget</th>
<th>Decision processes</th>
<th>Simultaneous analysis component learning</th>
<th>Solution synthesis exploration</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>simulated biomedical question answering</td>
<td>$10^{10}$ and varied</td>
<td>$9.3 \times 10^7$ (Phase A), $5.9 \times 10^9$ (Phase B)</td>
<td>≤ 4 days</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>7</td>
<td>pharmaceutical decision</td>
<td></td>
<td></td>
<td>≤ 1 hour (Phase A), ≤ 10 minutes (Phase B)</td>
<td></td>
<td>yes (solution synthesis)</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>product recommendation</td>
<td></td>
<td>$5.2 \times 10^{29}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td>$1.7 \times 10^{35}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

We first apply analytics meta learning to a biomedical QA task, defined by the BioASQ challenge [199]. Since the task has predefined checkpoints at certain stages along the pipeline, we, in this case study, focus on configuration space optimization, rather than procedure construction and optimization. We develop analysis components for all functionalities, especially we investigate whether it is possible to design and train supervised models to answer questions, without the use of manually-constructed rules or predefined templates. We then proceed to leverage the analytics space exploration algorithms to boost the system performance using the BioASQ 2013–2015 benchmark set containing over 1K questions and their gold standard solutions nominated and created by biomedical professionals. The results show that the analytics meta learning framework can help identify an optimal system which has an even better performance than the winning system.

Next, we consider two complex analytics task scenarios that require human decision making – pharmaceutical decision task and product recommendation task, where we demonstrate construction and optimization of both analytics procedure and analysis components. Specifically, we use an expert created target validation process as an example for the pharmaceutical decision task, and perform the proposed automatic procedure construction algorithm to generate phone recommendation decision processes. Then, we employ the biomedical or general-purpose QA components and pipelines to perform sub analytics tasks. Finally, we explore the analytics spaces that contain
over decillions of analytics procedures, where we also consider the scenario where the low-level analysis components (e.g. solution synthesis) learn and update their own models at the same time as the analytics meta learning optimizes the overall performance by exploring and evaluating these analysis components.

1.3 Achievements

Despite that this thesis is the first step towards analytics meta learning, up until now, we have made some achievements in many aspects of the problem, e.g. implementation and deployment of analytics space specification framework, budgeted policy optimization strategies for optimizing a configurable intelligent information system, supervised decision synthesis algorithms, and procedural knowledge extraction methods for decision support. In this section, we summarize the achievements that have been published or have made a real world impact.

First, we proposed a YAML-based execution process description language, Extended Configuration Descriptor (ECD) [220], which extends the description languages in the UIMA framework. ECD language inherits many advantages of the description language in the UIMA framework, e.g. a multi-layered design that separates the component-level Analysis Engine (AE) descriptors from the pipeline-level Collection Processing Engine (CPE) descriptors, an explicit distinction between the capabilities and configuration parameters, etc. Furthermore, it provides a compact and editable representation of an entire solution space for an analytics task by all possible configurations of all analysis components from which an intelligent information system can be composed. We also developed an open source software toolkit to instantiate the information processing systems specified by an ECD and execute them in an efficient way. The ECD framework has been successfully used in developing intelligent information systems for various analytics tasks since its first release in mid-2012, e.g.

- (2011 – 2013) a biomedical document and passage retrieval system [220] that addresses the information needs in genomics research as defined in the TREC Genomics task [79], where we selected a number of biomedical knowledge bases, algorithms and toolkits for information retrieval, natural language processing (NLP) and machine learning (ML) that were most cited by the task participants in their notebook papers and subsequent publications,
- (2013) a machine reading comprehension system [150] that is challenged by multiple-choice questions, as defined in the CLEF QA4MRE tasks [151], which covers various scenarios, e.g. reading comprehension tests in college entrance exams, Alzheimer’s disease treatments, etc.,
- (2013 – 2014) a general decision support framework, QUADS [221], which has been applied to two real-world applications: target validation, a fundamental decision-making task for the pharmaceutical industry, and product recommendation from review texts, an everyday decision-making situation faced by on-line consumers,
- (2014 – present) a biomedical QA system that complies with the requirement of the CLEF BioASQ QA task [199], which, compared to prior similar tasks, covers a wider range of

\[http://yaml.org\]
biomedical subdomains and question types, and requires a more comprehensive decision report, including relevant documents, snippets, concepts, RDF triples, in addition to the standard exact answers and summaries. Our exact answer generation pipeline was ranked #1 in 5 out of 6 focused categories in the official evaluations.

We have also taught the ECD framework in a number of graduate-level courses related to design and engineering of intelligent information systems, and employed it as the development and collaboration platform in the course projects, which include

- (Spring, Fall 2012) CMU 11-634/11-632: MCDS Capstone Project / MCDS Capstone Planning Seminar,
- (Fall 2012, 2013) CMU 11-791/11-792: Software Engineering for Information Systems / Design and Engineering of Intelligent Information Systems,
- (Spring 2014) ITAM COM-35702: Sistemas Inteligentes,
- (Spring 2013, 2015, 2017) CMU 11-797/11-796: Question Answering / Question Answering Lab,
- (Spring, Fall 2015) CMU 11-697/11-696: MIIS Capstone Project / MIIS Capstone Planning Seminar.

In another effort to provide a more effective medium to describe arbitrary analytics task solution space, we proposed a YAML-based decision process description language, Decision Process Template (DPT) [221]. To overcome the inflexibility in adapting to new decision scenarios of traditional decision support systems [33, 182], a novelty of DPT is that it allows users to describe the task goal and steps using natural language. In addition to specify an execution process (i.e. ECD) that directly tackles the task, DPT also allows to specify analytical thinking as sub-processes by giving references to other decision factors (also DPTs) that should be considered for the decision scenario, which captures non-linear human decision logics, such as trees or graphs. Following the DPT guidelines, we created a target validation DPT, based on literature review as well as personal knowledge and experience from a group of professional biochemists and bioinformaticians working at a pharmaceutical company, and a cell phone recommendation DPT. The QUADS framework [221], as depicted in Figure 1.3 could automatically understand and process the decision scenario via DPT in real time, providing an overall weighted decision by combining evidence from answers to individual questions. We applied the QUADS framework to the two applications to help bioinformaticians validate potential gene targets for diseases and help on-line customers to choose products based on users’ reviews.

To enable automatic construction and optimization of execution processes, we developed the Configuration Space Exploration (CSE) framework [220] based on the ECD framework, as shown in Figure 1.4 with two notable extensions: performance evaluation and data persistence. Beyond the brute force approach that exhaustively evaluates all the pipeline configurations, in our initial study of the analytics meta learning problem, we proposed an exploration strategy based on hierarchical Bayesian modeling and stochastic scheduling to improve the overall system performance through iterative update of the pipeline configuration. The CSE framework with the built-in exploration strategy was first used to optimize the above-mentioned biomedical document and passage retrieval system over a trillion different configurations of components and parameter values. The
results have suggested that the proposed solution was able to find an optimal configuration of components for TREC Genomics [79] with a passage MAP score better than prior published results for the task [220]. At the same time as we tested the CSE framework with a real-world analytics task – TREC Genomics task, a group of students taking the Question Answering course were developing the machine reading comprehension system for the upcoming CLEF 2013 QA4MRE challenge [151]. Throughout the semester, each student was individually developing one or some of the analysis components and further optimizing these modules by fixing the rest of the components unchanged from the baseline system. One day prior to the submission deadline, we managed to convert the vanilla UIMA descriptors to ECDs, and then leveraged the CSE framework to identify a better combination, which achieved 59.6% performance gain over the system that integrated all the individually optimized components [150]. The CSE framework inspired us to create the AML extension, which allows to configure the exploration strategy.

Until recently, either the user or an expert had had to manually and explicitly specify the general human decision processes via DPT, such as the target validation DPT and the cell phone recommendation DPT, before the user leveraged the QUADS framework for decision support. Most recently, we made a first attempt to study procedural knowledge discovery from heterogeneous sources, and eventually construct DPTs automatically without human interference. We proposed to take advantage of an existing procedural knowledge base, e.g. wikiHow\(^3\) to assist task-oriented information seeking tasks, using query suggestion as an example [219]. We compare our proposed solution with baseline algorithms, commercial search engines, and the (manually-curated) wikiHow procedural knowledge; experimental results show an improvement of +0.28 to +0.41 in terms

\(^3\)http://www.wikihow.com
of Precision@8 and Mean Average Precision (MAP).

1.4 Thesis Structure

The thesis is organized as follows:

Chapter 2 Analytics Meta Learning
This chapter first formally defines the terminology related to analytics meta learning, including analytics tasks, analysis components, information systems, etc., then introduces the problems of analytics meta learning and budgeted analytics meta learning, and presents a general solution framework that consists of analytics procedure definition, analysis component construction, and analytics space exploration.

Chapter 3 Procedural Knowledge Discovery
Despite manual specification of analytics procedures as described in Chapter 2, this chapter focuses on a subfield of analytics meta learning – automatic procedure discovery algorithms. We explore a wide range of human-generated data sources and design algorithms to mine such knowledge and canonicalize the representation from each respective source. We focus on three types of data sources that we have successfully leveraged to this problem including users’ search query log, semi-structured human curated procedural knowledge bases, and relevant Web documents from search engine result page (SEPR), etc.

Chapter 4 Exploration Algorithm: A Hierarchical Bayesian Model Method
From this chapter, we study the optimization problem in analytics meta learning – analytics space exploration, which aims to rapidly identify the optimal information system. We first present an exploration strategy based on hierarchical Bayesian model and stochastic scheduling, and report the experimental results on our preliminary studies using the TREC
Genomics QA task, which involves a configuration space (a subspace of an analytics space) consisting of trillions of possible execution processes.

Chapter 5: Exploration via Policy Optimization for CSE-MDP
As the second step, we model the exploration problem as a Markov Decision Process (MDP) and thus we seek the optimal control policy, which allows to easily incorporate the state-of-the-art policy optimization algorithms proposed for reinforcement learning problems, including tabular value estimation, function approximation, policy gradient, model-based methods, etc. The method proposed in Chapter 4 can be seen as a special instance of the model-based method. We empirically compare different state representations, option selection strategies, and policy learning methods using simulated configuration spaces under both the unconstrained and budget-constrained conditions.

Chapter 6: Analytics Meta Learning Implementation
This chapter focuses on the development of the analytics meta learning framework. We discuss the seven high-level design principles of the architecture layer for those who intend to implement a similar layer for their in-house architecture frameworks other than UIMA, and describe our current implementation of ECD framework and AML extension based on Apache UIMA and uimaFIT\(^4\) to facilitate design and specification of analytics space and exploration over the defined analytics space respectively. We also describe the implementation of built-in framework components.

Chapter 7: Case Study: Biomedical Question Answering Task
This chapter presents the use of analytics meta learning in construction of a BioASQ-like information system that supports a series of real-life information seeking needs from biomedical experts. We first define the task and present the analysis components that we have developed for the BioASQ 2015 and 2016 challenges, then report the configuration space exploration results, compare the exploration strategies proposed in Chapter 5 and summarize the strategy selection principles based on our observation. We also compare the information systems discovered by the exploration strategies with the baseline system which participated in the official evaluation, and discover that almost half of the strategies use a two-step option selection method have identified an even better system.

Chapter 8: Case Study: Pharmaceutical Decision Task
This chapter presents the use of analytics meta learning for complex pharmaceutical decision scenarios and introduces a decision support task in drug discovery – target validation. We formally define the use case of the problem by showing inputs and expected outputs, and present the manually created analytics procedure for target validation, which includes a decision process and a simple generic execution process for each decision factor. Then, we describe the analysis components in the analytics space, which includes a machine learning solution synthesis algorithm as well as the components that we adapt from the biomedical question answering system developed in Chapter 7. We apply a full analytics meta learning approach to optimize target validation decision support system. In this experiment, the system is able to update the meta learning model simultaneously while it trains the lower-level

\(^4\)https://uima.apache.org/uimafit.html
individual machine learning components such as the solution synthesis module. We find the strategies that use the two-step option selection method and tabular MC control for policy optimization achieve the best performance – 0.75 in terms of accuracy.

Chapter 9: Case Study: Product Recommendation Task

This chapter focuses on another use case that involves complex problem solving procedure – product recommendation task and introduces the product recommendation task. We first give a formal definition to the product recommendation task, and then perform the proposed automatic procedure construction algorithms to define analytics procedures for phone recommendation. Then, we collect product recommendation related data sources and reuse or develop analysis components to each specific analysis step. Finally, we explore the configuration space that contains decillions of information systems. Despite the simplicity of the analysis components being integrated into the analytics space, the analytics space exploration strategies can still reliably find the optimal system that achieves the best performance of 0.93 in terms of accuracy.

Chapter 10: Conclusion and Future Directions

This chapter makes the conclusions and presents the future directions in the research of analytics meta learning.

We summarize the structure in Figure 1.5
We introduce a general solution framework that extends UIMA (detailed in §2.3).

**Analytics meta learning framework**

**Procedural knowledge discovery**
Automatic discovery of analytics procedures (detailed in §3).

**Bayesian modeling of CSE**
Bayesian modeling of configuration spaces and stochastic scheduling (detailed in §4).

**Policy optimization for CSE-MDP**
Exploration of analytics spaces via policy optimization for CSE-MDP (detailed in §5).

**Evidence-based yes/no QA**
A QA system adapted to drug discovery that focuses on availability of the components, detailed in §8.3.

**TREC Genomics**
A document / passage retrieval system related to genomics (detailed in §4.2).

**CLEF QA4MRE**
A reading comprehension system evaluated using multiple-choice questions (detailed in §4.2).

**BioASQ**
A general-purpose biomedical QA system for information seeking and decision support (detailed in §7).

**Product analytics engine**
A retrieval and QA system focusing on product related information (detailed in §9.3).

**Pharmaceutical decision support**
A decision support system using Bio QA technology in drug discovery, detailed in §8.

**Product recommendation support**
A decision support system that helps customers in product recommendation, detailed in §9.

**Frameworks**

**Biomedical analysis components**
(detailed in §7.4).

**General analysis components**
(detailed in §9.3).

**Libraries**

**TREC Genomics**
A document / passage retrieval system related to genomics (detailed in §4.2).

**CLEF QA4MRE**
A reading comprehension system evaluated using multiple-choice questions (detailed in §4.2).

**BioASQ**
A general-purpose biomedical QA system for information seeking and decision support (detailed in §7).

**Product analytics engine**
A retrieval and QA system focusing on product related information (detailed in §9.3).

**Applications**

**Evidence-based yes/no QA**
A QA system adapted to drug discovery that focuses on availability of the components, detailed in §8.3.

**Pharmaceutical decision support**
A decision support system using Bio QA technology in drug discovery, detailed in §8.

**Product recommendation support**
A decision support system that helps customers in product recommendation, detailed in §9.

**Figure 1.5: Thesis structure**
Chapter 2

Analytics Meta Learning

Researchers have studied related problems from various angles. For example, operations researchers and decision scientists have focused on describe and formalize decision making problems and solutions [84][169], while NLP and IR researchers have developed pieces of software to help understand human decision needs and collect relevant information. Software researchers have often viewed the problem from a systematic perspective, and proposed software development methodologies [17][143], automatic service composition algorithms [163] and automatic evaluation of systems [132][184]. The goal of analytics meta learning is to bridge the gaps between these related research areas and provide a comprehensive solution that assists developers in developing information systems given arbitrary analytics tasks.

In the next four chapters, we focus on presenting the theoretical foundation of the analytics meta learning problem, where we try to answer a series of questions, e.g. what analytics meta learning is, how it relates to traditional machine learning problems in data analytics, how we design algorithms to solve the analytics meta learning problem, and what the steps are when we want to solve an analytics task using analytics meta learning.

This chapter presents the general setup for the analytics meta learning. We first review prior work related to representation and standardization of information processing pipelines in Section 2.1 while we leave the literature review related to automatic discovery and optimization of analytics task workflows in Sections 3.1 and 5.1. Since we found that researchers could express the same concept in various ways, and on the other hand, the same expression could have different meanings in different contexts, e.g. goal vs. target vs. purpose, instruction vs. action sequence, step vs. action, etc., in Section 2.2, we give formal definitions to the terminology used in the thesis, such as analytics task, analysis component, information system, analytics procedure, etc., and introduce mathematical notations to each of them for ease of reference later in the thesis. Then, in Section 2.3, we formally define the problem of analytics meta learning, and in Section 2.4, presents a general solution framework that consists of three major steps: analytics procedure definition, analysis component construction, and analytics space exploration, where we also describe the criteria for a would-be user to evaluate if analytics meta learning is something he/she wants to undertake. We conclude this chapter in Section 2.5.
2.1 Related Work

The problem we study in the thesis is motivated by the great demand for assisting and automating the design and development of intelligent information systems. In order to effectively and efficiently compose an information system for a given analytics task, developers tend to leverage existing analysis components developed by people of wide range of skills and abilities, experienced core team members, novices, or unmet open source contributors. However, the data analysis algorithms and resources can hardly guarantee the interoperability between them; still less can they be reasoned and manipulated. In this section, we see that it is the description languages that drive the evolutionary progress towards automatic development of intelligent information systems.

To support integration of data analysis components, de facto or de jure, proprietary or open standards have been introduced to achieve interoperability at different levels. Data exchangeability is a primitive requirement for component interoperability. For example, the JAVALIN system [144] introduces a shared data standard via XML schema, and each component in the system deserializes an XML data stream for processing and serializes the output data into an XML stream, but how each component processes data is unclear at the data standard level. To further enable component and configuration interoperability, an Application Programming Interface (API) is often defined to characterize the inputs, outputs, parameters and functionalities of each component, where specification languages, such as XML-based languages e.g. WSDL-S [4], OWL-S [74, 100] and Analysis Engine XML Descriptor [194], are also leveraged to describe components rather than data.

Description languages have later been utilized to represent workflows that combine individual components. In the next subsections, we review the representations of (1) workflows (execution processes) of an information processing pipeline, (2) decision processes (when solving analytics tasks), and (3) information processing architecture frameworks, which provide a comprehensive solution to manage the components.

2.1.1 Execution Process Representation

In this thesis, an execution process representation may refer to either an abstract symbolic representation for problem modeling (also referred to as process modeling languages), a machine-readable representation that combines machine executable subprocesses, or a human-friendly unstructured or semi-structured step-by-step manual written in natural language. Compared with the first two types, the last representation has recently gained much attention due to the expansion of user-generated contents on the Web. One objective of the thesis is to try to establish its equivalence with the former two.

Abstract symbolic representation. Researchers have leveraged graph representations and graph theories to model execution processes, or business processes in general, including published standards, e.g. Business Process Definition Metamodel (BPDM) [69], Business Process Model and Notation (BPMN) [70], as well as academic proposals, e.g. Event Driven Process Chain (EPC) [174], Integrated DEFinition Method 3 (IDEF3) [126], Petri Net [141, 156], Role Activity Diagram (RAD) [82], etc. Among them, BPMN [70] was first released in 2004 and since then has been maintained by the Object Management Group (OMG). Flow objects are the main describing elements within BPMN, and consist of three core elements: events, activities, and gateways, where
activities can represent tasks, subprocesses, etc, and gateways determine forking and merging of paths. Compared to BPMN and other industrial standards, Petri nets [141, 156], which mainly consist of place and transition nodes to represent possible states of the system and events or actions causing state changes, have an exact mathematical definition of their execution semantics, with a well-developed mathematical theory for process analysis, which therefore have been utilized for mathematical proofs of process behaviors (e.g. in [158]). List and Korherr [116] conduct a comparison between these conceptual modeling languages. Researchers have also introduced representations for processes with special properties, e.g. Workflow nets [202], a subclass of Petri nets [141, 156] specifically designed to represent workflow procedures, and Refined Process Structure Tree (RPST) [205], which is equivalent to the process models where the process components are canonical. In this thesis, we also focus on (well-)structured process models [205], whose definition is given in Section 2.2, while research has also been done to proof the equivalence and convertibility between the processes using the abovementioned representations such as BPDM, Petri Nets, RPST, etc [145, 158].

**Machine-readable representation.** Various machine-readable execution process representation languages have been introduced, mainly from the software engineering and system management communities, to integrate different types of components. An example of such a language that interacts with external entities through Web service operations defined using WSDL is Business Process Execution Language (BPEL) [5], an OASIS standard for specifying actions within business processes with Web services. Beyond the basic processes, e.g. receive and reply, it also introduces structured processes to define control flows, such as sequence, flow, while, etc. Due to the distributed nature of Web services, BPEL is often used in a distributed or grid computing environment. Another example is XML Process Definition Language (XPDL) [37]. It focuses on the design, rather than the execution, of workflows, i.e., it cannot execute the processes or even has no clue how to execute them. Instead, it represents the interchanges of business process definitions between different workflow products, such as design tools, simulators, execution engines and Business Process Management (BPM) related tools. Aggregate Analysis Engine Descriptors, as part of the UIMA standard, can also be viewed as an execution process representation, which defines subprocesses via delegateAnalysisEngineSpecifiers, and mange the execution flow via flowController[1]. Converting abstract conceptual representation to machine-readable representation has been extensively studied, such as BPMN to BPEL [136]. Although people have been argued about the applicability and limitation of various business process languages [203], we believe these languages still provide valuable insight and guideline how to describe and standardize execution processes.

**Human-friendly representation.** Cognitive psychology defines procedural knowledge (or imperative knowledge) as the knowledge exercised in the performance of some task [6, 65]. The Semantic Web community has attempted to formally define ontologies to represent procedural knowledge [27], which usually include an instruction (or action sequence) and a purpose (or goal). Also defined in such ontologies are the relations between procedural knowledge elements, at different levels of granularity, such as has-step, has-goal [148] or is-achieved-by [63]. Different from

1In fact, a configuration parameter in the Flow Controller may refer to a description of the desired flow in some flow language such as BPEL.
machine-oriented execution process representations, procedural knowledge representations usu-
ally reflect business rules or best practices in everyday life and intend to share across experts and
novices. For example, cooking recipe is a common type of execution process representation that
demonstrates how to prepare or make a culinary dish. Installation instruction in the user’s manuals
is another type of execution process representation. Similar to the machine-oriented execution pro-
cess representations, the steps (equivalent to the primitive or lower level processes) and the order
(equivalent to the flow) are the two important ingredients in the representation.

The analytics meta learning problem and our solution in the thesis are general and should work
with arbitrary execution process representation and management tools. We give our definition
to some of these concepts and discuss the connection to the prior models in Section 2.2 and then we further define concepts that are related to analytics meta learning that do not exist in
the prior literature, e.g. procedure space. Also, the thesis attempts to bridge the gap between
the abstract symbolic and machine-readable representations and human-friendly natural language
representations, in order to construct an executable pipeline from a natural language objective.

2.1.2 Decision Process Representation

Although decision logic is an important part of procedural knowledge for many decision scenarios,
esp. those that involve selection among several alternative possibilities, e.g. buying guide, career
opportunity evaluation, etc., decision making and decision process representation are more often studied beyond the context of software engineering and system management. Therefore, there is
another body of research to study the description and modeling of repeatable decisions within or-
ganizations to ensure that decision models are interchangeable across organizations. Again, we
review abstract symbolic representations and machine-readable decision models that allow auto-
matic validation and/or execution.

Abstract symbolic representation. In operations research or decision science, decisions are
usually represented by a hierarchy or network of sub-criteria or sub-problems that are more eas-
ily comprehended or evaluated in isolation. Example representations include traditional decision
table [192], decision tree, as well as Analytic Hierarchy Process (AHP), or Analytic Network Pro-
cess (ANP) representation [169], or more compact Influence Diagram (ID, or decision diagram)
[84], etc. AHP or ANP decomposes a decision problem into a hierarchy or network of more eas-
ily comprehended sub-problems, each of which can be analyzed independently. The elements of
the hierarchy can relate to any aspect of the decision problem. Since estimating criteria weights
and option priorities in AHP mostly relies on the judgments of human decision makers, pairwise
comparisons of criteria and options are used in AHP instead of direct allocation of weights. AHP
focuses on one single uncertainty with multiple alternatives, and decision makers are not allowed
to explicitly assign priority judgments to non-primitive criteria. In contrast, an ID, an extension
of a Bayesian network of chance nodes augmented with decision nodes, utility functions speci-
ifying the preferences of the decision maker, and a precedence ordering, can more flexibly model
complex decision situations with multiple alternatives from multiple uncertainties, and allows the
decision makers to specify their preference at any node in the diagram. However, finding the
optimal solution to a general ID is much more computationally expensive than AHP, based on
standard Bayesian inference technique [89]. Efficient exact and approximate algorithms for solv-

16
ing traditional decision diagrams, Discrete Influence Diagrams (DID), Limited Memory Influence Diagrams (LMID) [108], etc., have therefore been extensively studied [99, 118, 191].

**Machine-readable decision models.** Object Management Group (OMG) has recently initiated an effort to establish a standardized specification of decision making with the introduction of the Decision Model and Notation (DMN) [71], which is considered complementary to the current BPMN standard for more efficient description of processes. DMN introduces four types of element at the decision requirements level: *input data, decision, business knowledge, knowledge source*, and the Friendly Enough Expression Language (FEEL) for specifying detailed decision logic.

In analytics meta learning problem, we give “first-class citizenship” to both execution processes and decision processes required for accomplishing an analytics task, which makes it possible to move towards cognitive computing – combining human expertise with machine components and achieving overall analytics meta learning. For the purpose of building information systems for analytics tasks, the decision requirements are considered part of the analytics procedure (together with execution processes), whereas the decision logics are specific algorithms and thus analysis components. Therefore, at the processes level, our analytics meta learning model combines the execution processes (as in BPMN) and decision requirements (as in DMN).

### 2.1.3 Information Processing Architecture Frameworks

As opposed to the execution process management tools that facilitate the developers in the *ex post* integration stage, information processing architecture frameworks, such as UIMA [56, 57], GATE [39, 40], Heart of Gold [172], Ellogon [155], TIPSTER [68], etc. on the other hand, manage the entire software development process in a principled way, from specification of requirement, design of system architecture, development of components, re-assessment and selection of features, functionalities or alternative solutions in each increment and iteration, testing, to final deployment of the complete system. In the development of Watson, the UIMA architecture enabled an average of roughly 25 core researchers and engineers to routinely prioritize, develop, and test new algorithms once every two weeks throughout the four-year development [59].

Information processing architecture frameworks are a suite of toolkits that standardize information systems in terms of a set of basic elements (such as typed object, module, configuration, resource, workflow, etc.). The Unstructured Information Management Architecture (UIMA) [56, 57], which is considered the most evolved and comprehensive architecture available [12], was originally developed by IBM to facilitate the analysis of unstructured information, including but not limited to text, speech, images or videos. It became an Apache Incubator project in 2006 before it graduated in 2010, and became an OASIS standard in 2009, and IBM’s analytics engine Watson [58, 59] is a prominent example that uses the UIMA architecture. The UIMA framework defines the following elements to characterize information processing workflows:

- The artifact being analyzed (e.g., a document, audio file, video stream etc.) and the analysis results are encapsulated in a *Common Analysis Structure (CAS)*, which provides a mechanism for shared access across analysis components.

---

1 An interesting comparison between these frameworks is described in a survey paper [12].
• The schema or class model for the CAS is represented by a *type system*, which defines the types of objects and their properties (or features) that may be instantiated in a CAS.

• *Analysis components* interact (read, write, modify, etc.) with CAS by implementing the *process* method. Each analysis component can be associated with one or more *Component Descriptors* that contain metadata describing the component, its identity, structure and behavior. Configuring UIMA components is generally achieved by creating XML descriptor files, or using Java annotations directly in the component code via uimaFIT. An *Analysis Engine* (AE) can be instantiated from a component descriptor by the UIMA framework.

• A simple or primitive AE contains a single annotator. *Aggregate Analysis Engines* may be defined to contain other AEs organized in a workflow, where a *flow specification* defines the order in which the internal component AEs should be run.

The General Architecture for Text Engineering (GATE) [39, 40], which, despite its name, is also capable of processing audio-visual content, is another information processing architecture framework. Since the GATE framework only focuses on building Java-based information pipeline, it reuses many Java native features. All GATE resources are Java Beans. We briefly describe the GATE component model – *Collection of REusable Objects for Language Engineering* (CREOLE), in analogy to the UIMA framework.

• The objects being processed and analyzed, regardless of their granularity, lexicons, documents, or corpora, are all called *Language Resources* (LR). *AnnotationSet* is a data structure to store annotations specific to the Document type.

• Algorithmic components are named *Processing Resources* (PR), which can be configured similarly using XML descriptors or Java annotations.

• PRs can be grouped into *Applications* using *controllers*, where the controllers are also PRs, and thus they can be nested in another.

Measurability and reproducibility are also the focuses of information system architecture frameworks. Information retrieval communities have also held a series of competitions and conferences to create standard benchmarks and metrics to evaluate retrieval systems, and organizers have attempted to tabulate and analyze the most important features of high-performance systems, based on the information provided by the system reports, e.g. [165]. Due to lack of a standard task framework, organizers rarely reproduce experiments nor try different combinations of modules across participants.

Recently, several shared tasks employ standardized data flows for successive subtasks (e.g. IR4QA and CCLQA tasks in the NTCIR Advanced Cross-Lingual Information Access task [137, 138]), and the results of the upstream tasks were collected from each system and shared for processing by all participants in the downstream tasks. This approach was able to discover configurations that combined different task systems to achieve a better result than the originally reported systems, which lends further motivation to configuration space exploration. However, the evaluation achieved interoperability through task-specific protocols, and did not provide a general framework for describing and evaluating configuration spaces that include variation on components and parameter settings.

To facilitate system evaluation based on commonly-shared benchmarks, researchers have re-
cently paid more attention to open and public evaluation infrastructures for information retrieval evaluation and information extraction [215]. More recently, the Workshop on Data infrastructurEs for Supporting Information Retrieval Evaluation [3] was held to focus on organizing benchmark collections and tools into coherent and integrated infrastructures to ensure reproducible and comparable experiments; various retrieval evaluation frameworks were presented [53]. However, systems are usually specific to a single retrieval task, and lack a clear separation of framework, component configuration, and component logic, which makes it hard to fully leverage existing tools through automatic configuration space exploration.

In this thesis, we study the theoretical problems in analytics meta learning regardless of any particular information processing architecture frameworks. In Chapter 6 we describe our extension to the UIMA framework for the architecture layer that achieves analytics meta learning. Our experience can be used to extend other existing open or proprietary information processing architecture frameworks.

### 2.1.4 Limitations of Specification Languages

These above-mentioned specification languages have too limited expressive power to meet the requirement of analysis components and intelligent information systems, e.g.

**Lack of expressivity of module configuration.** Many analysis components expose configuration parameters to allow easy adaptation. For example, a pretrained model should be specified for a statistical NLP parser, and scaling coefficients $k_1$ and $b$ should be specified for Okapi BM25 [90] based retrieval component. Most current process description languages make no distinction between the configuration parameters and the actual input data to be processed and analyzed.

**Lack of expressivity of human decision logics.** Existing description languages focus on execution processes within analytics, rather than higher-order nonlinear decision processes that combine analytics and less complex analytical thinking as sub-processes. For example, solving analytics task (5) in the earlier example (“is AKT1 directly involved in the breast cancer and can be a suitable target?”) may include an execution process of searching for direct evidence from the literature and/or a decision process that considers factors (5.1) to (5.3). The description should be able to satisfy these new requirements, e.g. indicating what role each decision factor plays in solving the bigger problem, how the output from each decision factor should be synthesized, etc.

**Implicit unstated expectations.** There may exist implicit, unconscious or unstated expectation – how the components should be put together to achieve its maximum potential. Developers may not become aware that such expectation exists, or they are not able to detail the actual semantics using existing specification ontology. This happens more often in building intelligent information system. For example, an answer generation (AG) component extracts named entities of certain answer type from a list of passages generated by an upstream passage retrieval (PR) component. If the development team has effectively controlled the quality of the PR component, the AG developer may never need to consider the situation when a PR component also generates irrelevant passages, nor specify that the input passages...
should reach a certain level of relevance as a precondition.

2.2 Terminology

In this section, we give formal definitions and notations to the prerequisite concepts needed for the definition of analytics meta learning, which include those that are commonly used in information system development and data analytics, e.g. analytics task, analytics procedure, analysis component, information system, etc. The terminology is introduced by following the same procedure that developers build an intelligent information system for an analytics task – we start with the description of an analytics task, and then break down into fine-grained analytics processes via a design process, create analysis components, and finally construct the information system via an instantiation process.

Definition 1 (Analytics, analytics task). Data analytics (or analytics) is the process (referred to as analytics procedure, denoted by $p$) to analyze massive data to discover useful knowledge about the information and make conclusion about a topic, to improve predictions and support decision making. Each analytics procedure $p$ has a domain $D(p)$ and a target $T(p)$. An analytics task is a concrete information seeking or decision making need that can be achieved by performing analytics procedure $p$, which is usually described by a task description that contains the input $x \in D(p)$ representing the task requirement and the output $y \in T(p)$ representing the expected outcome. The target set $T(p)$ is often referred to as the alternative set in decision making if it is predefined.

Extracting input type and expected output type from natural language task descriptions is similar to the focus and answer type extraction in question analysis [105]. We take Task Example 1 in Chapter 1, the input, i.e. the information need, is “TAp73 isoforms have been identified in humans”, and the implicitly stated output type is a number that corresponds to the count of TAp73 isoforms. Benchmark data set is an important source of task description with known ground truth outputs.

There are numerous kinds of analytics procedures of various complexity. A trivial form of an analytics procedure is a primitive step that can be executed with a single analysis component, which is referred to as analysis step. Usually the task descriptions are identical to the analysis step descriptions. For example, if the analytics task is defined as “to find all the words in a sentence”, then a whitespace-based tokenization, as an analysis step, can be considered as the analytics procedure to solve the task. Another form is a linear-chain execution process, which is only comprised of consecutive execution steps in series. For example, Task Examples 1 to 3 in Chapter 1 may be addressed a typical factoid question answering pipeline, which consists of four main steps: question analyzer, document retriever, passage extractor, and answer generator [193]. Also, a typical ontology-based information extraction pipeline integrates several preprocessors and aggregators [215]. We formally define an execution process as follows.

Definition 2 (Execution process). An analytics procedure $p$ for analytics tasks $D(p) \rightarrow T(p)$ is an execution process if it contains an ordered sequence of sub procedures: $p_1, \ldots, p_n$, each $i = 1, \ldots, n$ focusing on an individual subtask $D(p_i) \rightarrow T(p_i)$, such that
(1) $\mathcal{D}(p) \subseteq \mathcal{D}(p_1)$,
(2) $\mathcal{T}(p_i) \subseteq \mathcal{D}(p_{i+1})$ for $i = 1, \ldots, n - 1$, and
(3) $\mathcal{T}(p_n) \subseteq \mathcal{T}(p)$.

Many times, solving an analytics task also involves collecting and synthesizing indirect evidence for various factors that impact the overall decision, which requires human decision knowledge, which may differ across the stakeholders and evolve over time. For such complex multi-criteria analytics tasks, analytics may also employ a hierarchical decision process to capture tree-structured decision logics, such as decision tree, decision table network \[192]\, or AHP \[169]\. It follows the divide-and-conquer approach via problem reduction and solution synthesis phases. In the problem reduction phase, analytics procedures are often formalized by a recursive procedure that decomposes the task into a hierarchy or network of lower-level sub-criteria or sub-problems that are more easily comprehended and/or evaluated independently by sub analytics procedures \[136]\. In the solution synthesis phase, the original higher-level analytics task objective is achieved by synthesizing the lower-level conclusions obtained from sub analytics procedures.

For example, to make decision for Example 5, a number of sub analytics tasks, as listed in the following Examples 5.1 to 5.3, are considered necessary from the best practice of biomedical researchers \[87\, 102\, 221]\:

(Ex. 5.1) Is there any experiment showing that modulating the activity of the AKT1 with a chemical compound or genetic modification causes the breast cancer?
(Ex. 5.2) Is any mutation associated with the breast cancer?
(Ex. 5.3) Does any evidence suggest that targeting the AKT1 will have side effects?

Similarly, online how-to manuals, such as wikiHow\[3\], eHow\[4\] and many other task-specific guides, provide comprehensive and up-to-date instructions how to accomplish everyday tasks. For example, the following steps are suggested for Example 6 by a wikiHow article\[5\]:

(Ex. 6.1) Select a cellular carrier that is currently carrying Android phones.
(Ex. 6.2) Decide on the size and quality of the screen.
(Ex. 6.3) Read customer and professional reviews to get a better idea of the potential problems with any given Android phone you are considering.

We give formal definition to problem reduction and solution synthesis for the most general cases below.

**Definition 3** (Problem reduction, solution synthesis). Given all the analytics procedures $p_1, \ldots, p_n$ and an analytics procedure $p$, the problem reduction $\rho_p$ is a function defined on $\mathbb{R}^{\mathcal{D}(p)} \rightarrow \mathbb{R}^{\mathcal{D}(p_1) \times \cdots \times \mathcal{D}(p_n)}$, and the solution synthesis $\sigma_p$ is a function defined on $\mathbb{R}^{\mathcal{T}(p_1) \times \cdots \times \mathcal{T}(p_n)} \rightarrow \mathbb{R}^{\mathcal{T}(p)}$.

Intuitively, the problem reduction process for an analytics procedure $p$ is to (1) find $n$ analytics procedures from $p_1, \ldots, p_n$, (2) assign a task input $x_i \in \mathcal{D}(p_i)$ for all $i = 1, \ldots, n$ based on the task

\[\text{http://www.wikihow.com}\]
\[\text{http://www.ehow.com}\]
\[\text{http://www.wikihow.com/Choose-an-Android-Phone}\]
input $x \in \mathcal{D}(p)$, and (3) assign a real-valued weight to each assigned task input $x_i$ based on the real-valued certainty of the original task input $x$. If any analytics procedure $p'$ is not selected in the first step, or an input $x_i'$ is not assigned to analytics task $p_i$ being selected in the first step, the problem reduction function can simply assign a weight of 0. Similarly, the solution synthesis process for an analytics procedure $p$ is to (1) find $n$ analytics procedures from $p_1, \ldots, p_n$, (2) assign a task output $y \in \mathcal{T}(p)$ based on the task outputs from the impacting analytics procedures $y_i \in \mathcal{T}(p_i)$ for all $i = 1, \ldots, n$, and (3) assign a real-valued weight to each task output $y_i$ based on the real-valued certainty of the task outputs $y_i$.

We note that if no weight or certainty is involved in the decision process, i.e. the problem reduction and solution synthesis in the decision process is deterministic, then we can simplify the definition for both processes as $\rho_p : \mathcal{D}(p) \to \mathcal{D}(p_1) \times \ldots \times \mathcal{D}(p_n)$ and $\sigma_p : \mathcal{T}(p_1) \times \ldots \times \mathcal{T}(p_n) \to \mathcal{T}(p)$. However, there are a few reasons we might want to consider the certainty and weight. First, some implementation of the analytics procedure can generate more than one conclusions, each associating with some confidence score, which can be utilized by the downstream components to improve the overall performance. Second, users can specify their preference to the analytics procedures that are used in the problem reduction and/or solution synthesis processes by assigning different weight values. Third, these two processes are isomorphic to gateway nodes in the Business Process Model and Notation (BPMN) [70]. While the deterministic versions provide the hard AND-join like aggregation strategy, the generalized versions provide a probabilistic AND connection, which can be used to optimize the execution cost by prioritizing and pruning the branches.

**Definition 4** (Decision process, decision logic, decision factor). An analytics procedure $p$ for analytics tasks $\mathcal{D}(p) \rightarrow \mathcal{T}(p)$ is a decision process if it contains a set of sub analytics procedures: $p_1, \ldots, p_n$, each $i = 1, \ldots, n$ focusing on an individual subtask $\mathcal{D}(p_i) \rightarrow \mathcal{T}(p_i)$, such that there exist a problem reduction $\rho_p$ and a solution synthesis $\sigma_p$ that are defined on $\mathbb{R}^{\mathcal{D}(p)} \rightarrow \mathbb{R}^{\mathcal{D}(p_1) \times \ldots \times \mathcal{D}(p_n)}$ and $\mathbb{R}^{\mathcal{T}(p_1) \times \ldots \times \mathcal{T}(p_n)} \rightarrow \mathbb{R}^{\mathcal{T}(p)}$ respectively. The pair $(\rho_p, \sigma_p)$ is referred to as the decision logic of the decision process $p$. Each sub analytics procedure $p_i$ in the decision process $p$ (with a non-zero weight for some input $x_i$) is called a decision factor of $p$. The set of decision factors $p_1, \ldots, p_n$ is denoted by $\mathcal{F}(p)$.

A complete analytics procedure for a complex analytics task may involve multiple hierarchical procedures combined with linear-chain procedures. For example, we often see that both execution steps and decision steps are interlaced in the how-to manuals. In fact, researchers have also studied a formal taxonomy of business processes based on their complexity, e.g. if it contains cyclic or acyclic processes, if the gateway contains OR operation, etc. [158]. In this thesis, we consider the analytics procedures constructed only from analysis steps, execution processes, and decision processes, and we thus define analytics procedures inductively as follows.

**Definition 5** (Analytics procedure, design process). A design process $\delta : (x, y) \rightarrow p$ aims to find an analytics procedure $p$ for an analytics task $(x, y)$. An analytics procedure $p$ designed for task $(x, y)$ is defined as

1. an analysis step, or
2. an execution process that contains analytics procedures $p_1, \ldots, p_n$, each designed by $\delta$, or
3. a decision process that contains analytics procedures $p_1, \ldots, p_n$, each designed by $\delta$.  

22
Our definition of analytics procedures is a type of process model, which simplifies the general models such as BPMN [70], Influence Diagram (ID) or decision network [84], where the latter proposals characterize any arbitrary process as a task-flow graph and thus can represent any structured decision problem under uncertainty in a more flexible way. In addition, our definition gives concrete semantics to each process component (analysis step, execution process and decision process). In fact, based on our definition, analytics procedures are canonical process components [205], i.e. if we represent the analytics procedures using the general graph-based process model, then the subgraph induced by the flows in the analytics procedure has a single entry node and a single exit node (a.k.a. process components), and any two analytics procedures are either disjoint or one is contained in the other (a.k.a canonical). According to the classification of canonical process components [158, 205], analysis steps, execution processes, and decision processes belong to trivial, polygon, and bond components respectively. The analytics procedures are also structured [97], i.e. for every node with multiple outgoing arcs (split), there is a corresponding node with multiple incoming arcs (join) and vice versa. In our case, the only split type in analytics procedures is the AND-join like problem reduction \( \rho \), and the only join type is the AND-join like solution synthesis \( \sigma \). Naturally, any analytics procedure can be represented as a refined process structure tree [205] with the root being the end-to-end analytics procedure and the leaves being the analysis steps.

Once the analytics procedure is determined either manually or automatically through the design process \( \delta \), the developer should create an information system following the analytics procedure. Since the building blocks of all analytics procedures are analysis steps, developers need to develop or integrate an executable module complying the definition of each analysis step. Software development best practices and architecture frameworks, e.g. UIMA [56, 57], GATE [39, 40], have encouraged developers to decouple configuration from logic. As an example, we consider Question Named Entity Recognition (NER), a key analysis step in understanding the semantics from natural language representation, where the input \( x \) is a text, and the output \( y \) is a list of named entities. One could use a rule-based NER, a CRF-based NER, or a knowledge base lookup based NER. The configuration parameter value could be the set of rules to use for the rule-based NER, a weight vector trained for the CRF model for the CRF-based NER, and the knowledge base to be used by NER. Here, we formally define analysis component, parameter and configurations.

**Definition 6** (Component, parameter, configuration). An analysis component or component \( f \) is a primitive processing unit. Each component is also associated with a set of parameters, denoted by \( \{\omega_i\}_i \), which constitute a component configuration \( \omega \). If all the parameter values are specified for a component, we call it a configured component \( f^\omega \) or \( f(\omega) \).

According to the definition of component and configuration, we can write the output produced by \( f^\omega \) (or \( f(\omega) \)) as \( y = f^\omega(x) \) (or \( f(x;\omega) \)), where \( x \) is an input (typed object) and \( y \) is the output. Note that parameters are not restricted to numeric values, but can hold a reference to any typed object. Once the analysis components and configuration values are determined for all the analysis steps, an information system that can accomplish the original analytics task can be constructed from the analytics procedure. We define an analytics engine or information system and an instantiation process \( \gamma \) inductively in parallel to the construction of analytics procedures.

---

6 We use \( f(\omega) \) in the subsequent sections only if a numeric value is involved in the \( \omega \).
7 Although both analytics engine and information system have the same definition, we use analytics engine more
Definition 7 (Instantiation, analytics engine). An **instantiation** process \( \gamma : p \to s \) maps an analytics procedure \( p \) for an analytics task \((x, y)\) to an analytics engine (or information system) \( s \) that processes \( x \) and produces \( y \). An **analytics engine** \( s \) instantiated from \( p \) by \( \gamma \) is defined as

- a \( p \)-compatible configured component \( f_\omega \), i.e. it is defined on the same domain \( D(p) \to T(p) \) as \( p \), if \( p \) is an analysis step,
- a processing unit that chains in series sub analytics engines \( s_1, \ldots, s_n \), where \( s_i = \gamma(p_i) \), if \( p \) is an execution process that contains \( p_1, \ldots, p_n \),
- a processing unit that aggregates in parallel sub analytics engines \( s_1, \ldots, s_n \) and the decision logic \((\rho, \sigma)\), where \( s_i = \gamma(p_i) \), if \( p \) is a decision process that contains \( p_1, \ldots, p_n \).

From the definition of analytics engine, we can easily see that an information system is also a process model, more specifically structured process model, like an analytics procedure. The topological structures of both process models are identical given a predefined installation process \( \gamma \). However, in the next section, when we study the analytics meta learning problem, we may vary \( \gamma \) or even the analytics procedure \( p \) to obtain a family of information system for an analytics task.

Also, we can see that any analytics engine \( s \) is comprised of a set of configured components, which however need to be executed in a certain order to ensure the inputs at a later step must be generated at earlier steps. We use the term **trace** to describe the processing path of an information system when it is applied to process an analytics task, i.e. a topologically sorted list of all configured components. Specifically, we use \( c \) to represent the index sequence \([c_1, \ldots, c_n]\) of the configured components selected for each process \( p_1, \ldots, p_n \) at step \( 1, \ldots, n \), and then the trace of the system is denoted by \( s_c \) and defined as follows

\[
f_c^\omega = f_{[c_1, \ldots, c_n]} = (f_{1,c_1}^\omega, \ldots, f_{n,c_n}^\omega)
\]

where \( f_{i,c_i}^\omega \) is the \( c_i \)-th analysis engine executed at step \( i \), and for any input requirement \( x \) of \( f_{i,c_i}^\omega \), there exists a \( f_{j,c_j}^\omega \) \((j < i)\) that produce \( x \) as an output. We use \( f_{c\{i,j\}}^\omega \) \((i < j)\) to represent a subtrace of \( s_c \) from processing step \( i \) to \( j \): \((f_{i,c_i}^\omega, \ldots, f_{j,c_j}^\omega)\).

We call an information system \( s = \gamma(p) \) a **pipeline** if \( p \) is a linear-chain execution process and does not include any decision process. The trace of a pipeline is unique. We take the typical factoid question answering pipeline again as an example, which is comprised of four analysis steps: question analysis, document retrieval, passage extraction, and answer generation, in an execution process. A trace would be a unique combination of components, e.g.

```
query tokenization by whitespace string splitter ->
document retrieval from Indri repository index with default parameters ->
sentence extraction based on LingPipe sentence segmenter and Vector Space Model similarity calculator ->
answer generation based on LingPipe NER with a pretrained model and frequency counting of named entities.
```

often to refer to a subroutine and **information system** to refer to a complete solution for the original task.

24
2.3 Problem Definition

In Section 2.2, we introduce the terminology related to developing information systems for analytics tasks. We review the process of creating an information system for an analytics task by first sketching an analytics procedure via design process $\delta$ and filling in analysis components for all the analysis steps via instantiation process $\gamma$, and accordingly we define the terminology along this process. However, as the analytics task becomes more sophisticated and thus more parties are involved, the development process might involve more options and uncertainty. The choice of all the basic elements of an information system – analytics procedures, analysis components, and configurations – may change over time and differ across the stakeholders, other researchers, and the massive ordinary Web users. For example, different directions or depths of analytical thinking may lead to different decision processes with distinct decision factors and/or dependencies even if the objective is the same. As a result, a primitive factor in one decision process can be further decomposed into sub-factors or expanded into an execution process in a more in-depth analytics procedure.

This observation motivates us to extend the process models defined for a single analytics procedure and a single instantiated information system to describe all the possibilities and variations during the development of information systems, which provides a foundation for analytics meta learning. In this section, we introduce the concepts of procedure space, system space, and analytics space, and then formally define the problem of analytics meta learning.

Definition 8 (Procedure space). For a given analytics task $(x, y)$, the procedure space $P$ is defined as the set of analytics procedures designed for the task via some design process $\delta$, i.e.

$$P = \{p : x \in D(p) \land y \in T(p)\}.$$

The procedure space can be considered as a family of process models that have the same start point $(x)$ and end point $(y)$, and any process in the procedure space can individually accomplish the task. Therefore, according to the definition of OR-join gateway, the entire procedure space can also be viewed as a single process model with all the analytics procedure connected by a OR operator.

Definition 9 (System space). For a given analytics procedure $p$, the system space $S$ is defined as the set of information systems instantiated from the analytics procedure via some instantiation process $\gamma$, i.e.

$$S = \begin{cases} 
\{f^\omega : x \in D(f) \land y \in T(f)\} & \text{if } p \text{ is an analysis step} \\
S_1 \times \ldots \times S_n & \text{if } p \text{ is an execution process or decision process containing } p_1, \ldots, p_n, \text{ where } S_i \text{ is the system space for } p_i 
\end{cases}$$

The inductive definition of system space allows us to prove that the system space can also be represented by the Cartesian product of the analysis components that are used in all the analysis steps of $p$ given a certain order (e.g. trace), i.e.

$$S = \{f^\omega_{\epsilon} = (f^\omega_{1,\epsilon_1}, \ldots, f^\omega_{n,\epsilon_n}) : f^\omega_{i,\epsilon_i} \text{ is compatible with } p_i\}$$
Therefore, a system space is also referred to as a **configuration space**. Similar to the procedure space, the system space can also be viewed as a single augmented process model, where each analysis step in the original procedure $p$ is replaced by an $\lor$-operator connected configured components compatible with $p$.

If we further vary both the design process $\delta$ and the instantiation process $\gamma$, we would obtain the analytics space.

**Definition 10** (Analytics space). For a given analytics task $(x, y)$, the **analytics space** $A$ is defined as the set of analytics engines designed for the task via some decision process $\delta$ and instantiated via some instantiation process $\gamma$, i.e.

$$A = \{ s : x \in \mathcal{D}(s) \land y \in \mathcal{T}(s) \}.$$ 

Again, the analysis space can be viewed as a process model that replaces each analytics procedure in the procedure space model with a system space model. In an analytics space model, the $\lor$ gateways are used to connect options between analytics procedures as well as configured components.

In the traditional business process modeling and execution, $\lor$ implies equal effect from interchangeable subroutines, and thus no difference can be observed from the outcomes, which however does not hold for an information system. First, runtime performance such as memory usage, CPU time, network bandwidth requirement, etc., might vary across different procedures and components. Therefore, if multiple information systems can produce the same output, the information system that optimizes the runtime performance is always preferred [178]. Moreover, for an analytics task, the stakeholders tend to evaluate the “intelligence” aspect of the information system in term of the task performance, i.e. how the system produced result compares with a known phenomenon or an expert insight, in addition to the runtime performance. If the task also requires efficient processing and its performance is thus evaluated accordingly, the task performance may also combine the runtime performance factors. In other words, although any combination of an analytics procedure and a set of configured components in the analytics space can be composed as an information system $s$ to find the outcome $y = s(x)$ “in theory”, there is no guarantee that $y$ is always the same or equally optimal in practice.

Analytics meta learning, unlike many other automatic system configuration and optimization solutions, aims to assist system developers in evaluating an entire analytics space, rather than a single system or a system space, and automatically optimizing the analytics for any given task. The key to the problem is to proactively arise awareness of the performance properties (including runtime performance and task performance) at different granularities – from a configuration parameter of an analysis component to the overall information system, so that the right decision can be made at a timely manner at each $\lor$ connector. While the runtime performance can be estimated when a component is executed at runtime or even at compile time [178], the task performance can only be judged using an evaluation metric on a gold standard output.

**Definition 11** (Evaluation metric). An **evaluation metric** or **metric** $d$ is a function defined to map $(y, \hat{y}) \in \mathcal{O}^2 \rightarrow \mathbb{R}$, where $y$ is the gold standard output, $\hat{y}$ is the system output, and $\mathcal{O}$ is the output type set. An **aggregated evaluation metric** is defined on a set of outputs.
We give some examples of evaluation metrics and aggregated evaluation metrics. If the output can be represented by a short concise word or key phrase, then the evaluation metric can be as simple as a binary scoring function \( d(y, \hat{y}) = [y = \hat{y}] \), where 1 represents “agreed”, “satisfied”, “relevant”, etc., and 0 represents “disagreed”, “unsatisfied”, “irrelevant”, or a graded scoring function, where the higher the grade, the more satisfied or relevant the output is. If the output can be represented by a set of elements, e.g. nuggets in an answer passage [44][137], then the classification-based evaluation metrics, such as precision, recall and F-measure, can be calculated. If the output can further be represented by a list of elements ranked by importance, e.g. search results, then the ranking-based evaluation metrics, such as average precision, reciprocal rank \([207]\), can be used. To aggregate the instance-level evaluation measurements, algebraic mean is often used, while sometimes geometric mean is also used \([199]\). In this thesis, we use \textit{supervision} for analytics meta learning.

\textbf{Problem 1} (Analytics meta learning). Given a set of analytics tasks with task requirements \( \mathcal{X} \subseteq \mathcal{I} \) and the corresponding known (expected) outcomes \( \mathcal{Y} \subseteq \mathcal{O} \), \textit{analytics meta learning} aims to find the information system \( s^* \) from the analytics space \( A \) that are designed and instantiated for the task \( \mathcal{I} \to \mathcal{O} \), such that the expected task performance is optimized, i.e.

\[
  s^* = \arg\max_{s \in A} \mathbb{E}_{x,y}[R(s(x, y)); \mathcal{X}, \mathcal{Y}] \tag{2.1}
\]

We also consider a more realistic version of the analytics meta learning problem, where we have limited resource to allow any human or machine to learn and discover the optimal information system, i.e. a budget. Resources used by a component include execution time, storage space, network bandwidth, etc., which can be measured by CPU time, allocated memory size, and data transfers respectively; a resource utilization measure can also be a more specific function of component characteristics (e.g., the cost to execute a configured component on Amazon Web Services is a function of execution time and hardware capacity utilized).

\textbf{Problem 2} (Budgeted analytics meta learning, or analytics meta learning under limited resource). Given a set of analytics task with task requirement \( \mathcal{X} \subseteq \mathcal{I} \) and the corresponding known (expected) outcome \( \mathcal{Y} \subseteq \mathcal{O} \), \textit{budgeted analytics meta learning} or \textit{analytics meta learning under limited resource} aims to find the information system \( s^* \) from the analytics space \( A \) that are designed and instantiated for the task \( \mathcal{I} \to \mathcal{O} \), such that the expected task performance is optimized and the processing cost in the learning process is bounded by a budget (resource capacity) \( C \), i.e.

\[
  s^* = \arg\max_{s \in A, (X', Y') \subseteq (\mathcal{X}, \mathcal{Y})} \mathbb{E}_{x,y}[R(s(x, y); X', Y')] \quad \text{s.t. } c(A', X') < C \tag{2.2}
\]

where we explore a sub analytics space \( A' \) from \( A \), and a sub development set \((X', Y')\) from \((\mathcal{X}, \mathcal{Y})\) due to the resource constraint. We use \( c(A, \mathcal{X}) \) to measure the overall processing cost of processing paths in \( A \) over the input set \( \mathcal{X} \). In contrast, we intend to ensure the constructed information system \( s^* \) is generalizable to other unprocessed data in the development set \((\mathcal{X}, \mathcal{Y})\), we calculate the expectation over the instance random variable \((x, y)\).

\footnote{https://aws.amazon.com/}
In this section, we present a general recipe for developing an optimal information system for a complex analytics task using analytics meta learning methodology, which consists of

- **Analytics procedure definition**: defining the analytics procedure, in order to design an architecture/framework that satisfies the task requirement,

- **Analysis component construction**: implementing compatible and configurable lower-level analysis components, and

- **Analytics space exploration**: applying an exploration strategy for automatic pipeline evaluation and iterative planning, which eventually determines the optimal information system over the analytics space, possibly under a limited resource assumption.

These three key steps define a development iteration or increment in the entire development life cycle, which can be repeated or paired with traditional development cycles that involve human decision making to complete a development process. We illustrate this process in Figure 2.1.

In Chapter [3](#), we study the algorithms that can help users define a procedure space automatically from user created contents. In Chapters [4](#) and [5](#), we study how to optimize the objective function Eqs. 2.1 and 2.2. Analysis component construction is relatively a task-specific problem, which is often studied in each respective field such NLP, information retrieval (IR), etc. From Chapter [7](#),
Table 2.1: Comparison between supervised learning and analytics meta learning

<table>
<thead>
<tr>
<th></th>
<th>Supervised learning</th>
<th>Analytics meta learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target problem</td>
<td>Single-step prediction task</td>
<td>Multi-step complex analytics task</td>
</tr>
<tr>
<td>Supervision</td>
<td>User feedback (e.g. Training set of input/output examples)</td>
<td>Task performance and runtime performance</td>
</tr>
<tr>
<td>Performance measure</td>
<td>Feature vector</td>
<td>Information system (or flattened processing trace)</td>
</tr>
<tr>
<td>Instance</td>
<td></td>
<td>Instance preparation</td>
</tr>
<tr>
<td>Operation space</td>
<td>Feature space</td>
<td>Procedure definition and component construction</td>
</tr>
<tr>
<td>Instance preparation</td>
<td>Feature engineering</td>
<td>Analytics space exploration</td>
</tr>
<tr>
<td>Optimization process</td>
<td>Parameter estimation</td>
<td>Exploration strategies</td>
</tr>
<tr>
<td>Optimization schemes</td>
<td>Search algorithms</td>
<td></td>
</tr>
</tbody>
</table>

we empirically validate the proposed methodology using case studies, where we demonstrate how analysis components are constructed.

We compare analytics meta learning with the traditional predictive analytics using supervised learning. Although both problems assume to be learned from either a batch of user feedback (e.g. a training set of input and expected output examples) or interaction with real users, and be evaluated using task and/or runtime performance, especially in the topic of energy efficiency in the machine learning community, existing supervised learning approaches can hardly be directly applied to a complex analytics task. The solution to a complex analytics task is a multi-step information processing system, which can be more conveniently represented, from a system architecture perspective, by a processing trace in the analytics space, whereas the solution space to a single-step prediction task is usually represented by a feature space in the supervised learning setting. We make further comparison between the concepts that have been widely adopted in both problems in Table 2.1.

Using the proposed analytics meta learning solution framework to build an intelligent information system follows a similar process as traditional software development, which includes a prerequisite step – requirement and use case analysis, and three major steps – analytics procedure definition (design), analysis component construction (implementation), and analytics space exploration (testing and evaluation). The case studies reported in Chapters 7 and 9 are also organized in the same order to demonstrate how to take analytics meta learning into action. Next, we focus on comparing the differences between analytics meta learning process and traditional software development process, in order to help a user determine whether he/she should consider analytics meta learning.

**Prerequisite step** Both development processes start with requirement and use case analysis to make sure that the developer has understood the customer’s need correctly. In order to apply analytics space exploration, the task definition should also include a benchmark set, which includes a set of inputs, the corresponding expected outputs (ground truth), and an evaluation metric that can compare a system output with the ground truth. If all the stakeholders can work together offline to define a fixed benchmark set, then they can easily plug them into
the framework. Alternatively, the evaluation can also be an interactive process between the analytics meta learning framework and a domain expert, who needs to judge the system output instantly and then responds to the framework. In this case, the benchmark set is collected online throughout the entire development process. If neither is possible, they are not considered ready with analytics meta learning.

**Analytics procedure definition** Again, both development models require an architecture (work-flow) design. If the developers have applied the traditional software development model and apply a flexible architecture framework, they must have a clear idea what the pipeline should look like. For example, if they use a UIMA architecture (which we further describe in Section 6.2), they should have a clearly defined Collection Processing Engine (CPE) that specifies the Analysis Engines (AEs) to accomplish individual analysis tasks. On the framework developer side, analytics meta learning requires an extension to the original specification language that allows alternative options at both the component level and the parameter level to be specified in a single descriptor. In Section 6.2 we demonstrate how to extend the UIMA framework to the UIMA-ECD framework to support this additional requirement. On the component developer side, they need to learn how to specify the analytics procedure in the predefined specification language, such as UIMA-ECD. Two anecdotes may be helpful here to estimate the “human cost”. When we built a system for the QA4MRE 2013 task, there were a group of students who created vanilla UIMA components throughout the semester, and one day prior to the system submission due, an experienced developer was able to convert all the 20+ UIMA XML descriptors into UIMA-ECD YAML descriptors and conducted exploration in a few hours. In 2013, we also gave two lectures (each about 80-minute long) to the students, teaching how to write their own UIMA-ECD descriptors and conduct exploration, and the homework showed that most students obtained this skill after the lectures. In short, if the stakeholders have UIMA experience or intend to use UIMA, then there is no framework development cost, and very tiny developer training cost. If they intend to use another in-house architecture, then they need to implement a similar extension to allow option specification.

**Analysis component construction** There is no significant difference between the development models when it comes to analysis component construction, and there should also be no cost to convert the components written for the traditional framework to the analytics meta learning framework and vice versa. For example, the components developed for the UIMA framework (JCasAnnotators) can be plugged into the UIMA-ECD framework without changing any Java code. We note that the best practices in traditional development model are also very welcomed in the analytics meta learning framework. For example, one can specify the input/output capabilities of each analysis step in UIMA for sanity check, and one is also encouraged to expose as many parameters as possible and assign the values in a separate description file, rather than hardwire the values in the code. Both practices can maximize the exploration effectiveness.

**Analytics space exploration** Similar to the analytics procedure definition stage, there should be no extra cost at the component developer side, since this is designed to be an automatic stage. However, an experienced user can also customize the exploration strategy, including
the state representation method, the value estimation method, the option selection method, and the greedy heuristic. In this thesis, we implemented the AML extension to the UIMA-ECD framework, which defines interfaces for all necessary submodules of an exploration strategy and provides implementations to a number of state-of-the-art methods, as detailed in Section 6.3. If the analytics space is defined using an architecture framework other than the UIMA-ECD framework, then there is also extra cost at the framework developer side in building a similar extension for analytics space exploration. We should also note that since there are many open source off-the-shelf reinforcement learning frameworks, one can also integrate them into the framework rather than reinvent the wheel. Also at the end of Section 6.3, we describe how to use a BlockingQueue data structure to synchronize between a pipeline framework and a external RL agent.

2.5 Conclusion

In this chapter, we formally define the terminology used in this thesis and the two key problems we study in the subsequent chapters: analytics meta learning and budgeted analytics meta learning. We also present a general solution framework to the analytics meta learning problem, which consists of analytics procedure definition, analysis component construction, and analytics space exploration.
Chapter 3

Procedural Knowledge Discovery

Despite manual specification of analytics procedures as described in Chapter 2, this chapter focuses on a subfield of analytics meta learning – automatic procedure discovery algorithms. We explore a wide range of human-generated data sources, such as community created and maintained semi-structured procedural knowledge base, search query log, as well as procedural knowledge embedded text, and design algorithms to mine such knowledge and canonicalize the representation from each respective source.

In Section 3.1, we first review prior work related to procedural knowledge extraction, which includes extraction from semi-structured procedural text and causal relation and argumentation extraction. We then in Section 3.2 give our definition of procedural knowledge base and introduce the features that can be extracted from semi-structured procedural knowledge bases, which are utilized in the following sections for the training purpose. Before we study the algorithms to expand a procedural knowledge base or construct one from scratch, we demonstrate how to leverage an existing procedural knowledge base for information retrieval applications. We take search task suggestion or query suggestion as an example in Section 3.3. We then propose to mine procedural knowledge from various types of data sources. Section 3.4 describes a supervised learning approach to take advantage of users’ search activities in a query log to identify task procedures. Finally, we conclude and propose future work in Section 3.5.

3.1 Related Work

In this section, we review prior work related to procedural knowledge extraction, which includes extraction from semi-structured procedural text and causal relation and argumentation extraction. Several automatic procedural knowledge base construction approaches have been proposed to extract instructions from semi-structured texts, e.g. eHow or wikiHow articles [2, 92, 148, 154], recipes [154], community-based QA [11], etc. Most approaches take advantage of structural information (e.g. HTML tags [11], enumeration indicators [47, 92]), and define rules or templates to extract textual content. In a separate step, NLP tools are applied to extract relations and normalize each goal and action to its ontological form, to support linking to other ontological resources. Researchers have also studied how to identify script knowledge [173] which focuses on the tem-
poral ordering of events [31]. People [2, 92, 148, 154] have also attempted to construct structured procedural knowledge bases by defining ontologies or incorporating other existing representations such as PDDL [60].

In contrast, our approach takes advantage of the writing style of semi-structured procedural texts and proposes a set of structural and textual features for identifying procedural knowledge; the implemented approach can be optimized with a supervised learning method. Moreover, beyond the conventional use of small-scale procedural knowledge in AI planning [142] or NLP [11], our thesis also studies the problem of how to apply a large-scale procedural knowledge base to complex task search and decision support problems.

Causal relation and argumentation are also often used to describe decision logic. A number of features and models have been proposed to extract two rhetorical structures relevant to our problem: causal and argumentative relations. In fact, identifying both structures would play a major role in decision making support [180, 183]. However, little research has attempted to formally study to directly identify a set of decision factors for a decision goal from texts in order to construct processes to support decision making. Causal relations are usually identified from formal documents such as reports [180], scientific articles [134], and social problem related Web pages [76], whereas argumentation is identified from persuasive essays [183] or debate texts [75]. Such relations are identified by classifiers trained on lexical features [22, 32], using PDTB corpora [164] or domain corpora [135, 159]. A number of discourse parsers [114, 157] have also been developed. We may further explore these tools, algorithms and data sets to allow extracting procedural knowledge from unstructured texts.

3.2 Procedural Knowledge Base

Many knowledge bases such as Wikipedia or Wikidata that have been widely utilized contain a huge amount of descriptive knowledge. Procedural knowledge [6, 65], also known as know-how, is the knowledge exercised in the accomplishment of a task, i.e. how to do things, which is usually acquired by experience and considered tacit and not easily shared, compared to descriptive knowledge. However, shared explicit procedural knowledge lays a foundation for efficiently coordinated action, which is often referred to as best practices or business rules within communities or organizations. As wikiHow and many similar how-to Web sites allow users to easily share procedural knowledge, semi-structured large-scale procedural knowledge bases have become available.

In this section, we give our definition to procedural knowledge base. In Sections 2.2 and 2.3 we have given definitions to a single procedure and a family of procedures that achieve the same goal. In this section, we consider the collection of all procedures that involve tasks, which we refer to as procedural knowledge base. Procedural knowledge captures how people decompose a complex task into a mixture of several mental actions (e.g. “choose”, “determine”, etc.) and/or physical actions (e.g. “visit”, “store”, etc.), which correspond to the decision process and execution process.

Definition 12 (Is-achieved-by relation, procedural knowledge base). We refer to the relation between the task and the subtasks in the execution and decision process as is-achieved-by relation, which connects all the task descriptions. Procedural knowledge base (or procedural knowledge
\( G = (T, R) \) is the set of all tasks \( T \) and all is-achieved-by relations \( R \). A weight \( w(t, s) \) may also associate with each relation \( r(t, s) \in R \).

We can see from the definition that an analytics procedure \( p \) can be considered as a tree induced from \( G \), with the root being the original task.

### 3.2.1 Semi-Structured Procedural Knowledge Base

A widely used type of procedural knowledge base is **semi-structured procedural knowledge base**. We extend the definition of *task* in Section 2.2 to consider the semi-structured representation, which includes both the description of the task objective and the description about the procedure, as we will see that even in a semi-structured procedural knowledge base, the task requirement, objective and action(s) are stated usually in a single paragraph.

**Definition 13** (Task description in semi-structured procedural knowledge base). A task \( t \) is represented by a short and concise **summary** and a detailed **explanation** for clarification in semi-structured procedural knowledge base.

A semi-structured procedural knowledge base such as wikiHow has an identical structure, but it allows users to more easily view and edit; each article corresponds to a task, whose summary is given in the title and whose explanation is detailed in the introduction section. Each step (or substep) also corresponds to a task, whose summary is given in a bold text at the beginning of the step, which is followed by the explanation. The is-achieved-by relations have three representation forms: numbered *steps* in each article page (first-level subtasks), bulleted substeps under each step description (second-level subtasks), and free links\(^1\) directed from a step to another task page. Steps that have no substeps or outgoing free links are considered *primitive tasks*. Current semi-structured procedural knowledge bases do not allow editors to explicitly specify the relation strength between tasks or the importance of each subtask\(^2\). Therefore, we assume each relation shares an equal weight with all other outgoing relations, i.e. \( w(t, s) = 1/\text{od}(t) \), where \( \text{od} \) is the out-degree of the task \( t \).

We attempt to leverage the style guidelines for writing semi-structured procedural knowledge, such as the wikiHow guide\(^3\), which indicates that an action-oriented instruction beginning with a verb is required at the beginning of each procedural step, but we also attempt to process articles that do not fully comply with the guide. We accordingly propose a set of textual features and structural features to identify query spans from each task description, and then adapt similar features to extract wikiHow-style procedural knowledge descriptions from search queries and relevant text snippets.

**Location (LOC).** As suggested in the wikiHow writing guide, a task should provide both “skimmable information that readers can quickly understand” in the title of the article and in the beginning sentence of each step, and “rich informative articles for more patient users” in the


\(^2\) As users may verbally express their confidence and/or their attitude toward the importance in the task explanation section, we can hence incorporate richer linguistic features such as modality particles, verbal auxiliaries, etc. to identify verbal expression of weight.

<table>
<thead>
<tr>
<th>Description</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>If ( w_i ) appears in the task summary and explanation</td>
<td>2 or 0</td>
</tr>
<tr>
<td>The sentence that contains ( w_i ) has a positive, negative, and neutral stance</td>
<td>3</td>
</tr>
<tr>
<td>The sentence that contains ( w_i ) is subjective and objective</td>
<td>2</td>
</tr>
<tr>
<td>Whether ( w_i )’s POS is one of the modality types used in Slinket</td>
<td>2</td>
</tr>
<tr>
<td>Fine-grained POS tags of ( w_i ) used in the Penn Treebank</td>
<td>36</td>
</tr>
<tr>
<td>Basic Stanford dependency types of ( w_i )</td>
<td>50</td>
</tr>
<tr>
<td>If ( w_i ) is inside a named entity, noun phrase and verb phrase</td>
<td>3</td>
</tr>
<tr>
<td>Surface form, stem, and TF-IDF score of ( w_i )</td>
<td>3</td>
</tr>
<tr>
<td>Surface form, stem, TF-IDF score and POS tags of ( w_{i-1} )</td>
<td>39</td>
</tr>
<tr>
<td>Surface form, stem, TF-IDF score and POS tags of ( w_{i+1} )</td>
<td>39</td>
</tr>
</tbody>
</table>

article’s introduction and in the sentences which follow in the detailed explanation of each step. Therefore, we define features to capture the location of each word.

**Sentiment of event mention (SEN).** Similar to other wiki-based knowledge bases such as Wikipedia, wikiHow discourages personal or subjective reference in the description (such as use of “I”, “we”, etc.), which motivates us to also detect sentence-level sentiment features such as polarity, subjectivity, modality (certainty degree) and factuality [171], and assign the same feature value to all the tokens in the sentence.

**Part of speech (POS).** Since both the article title and the first sentence in each step are required to begin with a verb in bare infinitive form, we also extract the fine-grained part-of-speech value for each token.

**Parsing (PAR).** To further understand the task facets such as occurrences of subsidiary resources (e.g. a target object) or constraints (e.g. a duration), we include features extracted from dependency parsing, named entity extraction, and chunking.

**Word and context.** As in other natural language sequence labeling tasks, we also define features including the token’s surface form, its stem form, its TF-IDF score, as well as the word features and the part-of-speech tags of the previous and the next words in the task summary or explanation.

A complete list of features is presented in Table 3.1

### 3.3 Application: Search Task Suggestion

Before we further discuss how to extract procedural knowledge from various sources, we demonstrate how to leverage existing procedural knowledge for information retrieval related applications beyond analytics meta learning. In this section, we consider a widely studied topic in information retrieval – search task suggestion or specifically query suggestion in the context of search engine.
We focus on task-oriented search tasks, where the objective is to accomplish real-life tasks. For example, to organize a conference (task), the organizer needs to look for information related to choosing a hotel, identifying and deciding among banquet options, recruiting volunteers, selecting and contacting publishers, etc (subtasks). For each specific task, users must gather more detailed information. For example, when choosing a hotel, the organizer must consider the number and size of conference rooms, potential arrangements for meal catering and menu planning, whether or not a discounted rate will be available, etc. There is a huge demand for search engines to better assist their users in achieving their intended goals for such *ad hoc* tasks. We call the problem task-oriented search task suggestion.

Researchers have studied problems related to search intent analysis for general queries; for example, how to identify search intents [30, 35, 41, 72, 96, 109, 117, 162], how to suggest search queries to the user [9, 38, 91, 129, 213], how to rank results that cover diverse aspects of the task, etc. Most studies have so far mostly relied on queries and search logs, search result texts, behavioral information, etc. to improve search quality for entity-centric queries [43]. Recently, research has shown that structured knowledge bases (such as Wikidata or formerly FreeBase) or semi-structured knowledge bases (such as Wikipedia) can be used to improve search quality and experience [25, 42, 112, 225].

When users turn to search engines for information seeking and problem solving, we investigate how existing procedural knowledge can be leveraged to help understand a user’s search intent and suggest sub search tasks to users.

**Problem 3** (Task-oriented search task suggestion, STS). Given a procedural knowledge graph $G$ and a task-oriented search task $q_0$, we aim to (a) identify the task $t_0 \in T$ the user intends to accomplish, and then (b) retrieve a list of $k$ subtasks $t_1, \ldots, t_k \in T$ and also (c) suggest the corresponding sub search tasks $q_1, \ldots, q_k$.

A search task is usually represented by a search query or a key phrase [77, 78, 119]. Similarly, we assume both the input search task $q_0$ and the suggested search tasks $q_1, \ldots, q_k$ are queryable phrases. As a result, the problem can be straightforwardly solved by suggesting relevant (sub)queries for task-oriented queries. In contrast to the entities and attributes in a descriptive knowledge base, which are usually words or phrases and can used directly in query suggestion, summaries and explanations in a procedural knowledge usually contain richer action and condition information. Therefore, the problem poses three challenges. First, we need to identify the task $t_0 \in T$ the user intends to accomplish. If the query corresponding to the search task exactly matches the summary of a task $t_0$ (exact string match or after lexical transformations), then we can directly return $t_0$. However, in most cases when an exact match does not exist, we need to incorporate additional information such as the search result page of the query and the full description of each task, and rank candidate tasks using a retrieval method. Second, we need to choose the tasks $t_1, \ldots, t_k \in T$ to suggest and then prioritize them based on the knowledge $G$ containing relations $V$.

*(Sub)task in this thesis refers to an action that is performed to achieve a goal in a general context [119], as we defined in Section 2.2, which differs from the notion of *task* in the search scenario [77, 78]. We use *sub* search task to denote the latter case.
Intuitively, we can choose the task $t_0$’s top-$k$ subtasks ordered by the weight of the is-achieved-by relation, but we may consider second-level subtasks (e.g. the bulleted items in wikiHow) as well. Finally, we need to extract queryable phrases from each task description, and the algorithm should thus learn how searchers have been trained to formulate queries.

### 3.3.1 Modeling & Solution

In this section, we propose a supervised approach to learn how searchers have been trained to formulate task-oriented queries from search query log, and apply the model on the semi-structured procedural knowledge base to extract queryable phrases. The challenge here is to identify only the task-oriented queries (instead of entity-centric queries) from a list of candidate queries to annotate the task descriptions. We create a collection of task descriptions with task-oriented query phrases annotated in an iterative manner.

We first identify the exact matching pairs of searchers’ issued search tasks and task descriptions in the procedural knowledge base, although we expect that many lexically distinct but semantically identical pairs may exist that will not be extracted. To achieve this, we may scan through the entire search query log to find each query $q$ that matches the description of the task $t$. We add the annotated task description to the corpus. In addition, we collect related queries by combining the user-issued queries from the same session and the list of queries suggested by the search engine for the next iteration. If we have no large query log, we may also manually create task-oriented search tasks for the tasks in the procedural knowledge base. Specifically, we use the summary of the task $t$ to form a search query $q$. Since these search queries are “artificially” created, and do not necessarily represent how searchers normally formulate queries for these tasks in similar situations, we exclude it from the parallel corpus, which is only used for search engines to suggest real queries for the next iteration.

Once we collect a list of related queries for $q$, we compare them with the known subtasks of $t$ in the procedural knowledge base. For each related query $q'$, we first find the subtasks $t_1, \ldots, t_n$ that contain $q'$ in the task description. We then add all the annotated subtasks $t_1, \ldots, t_n$ to the corpus. As we assume that all necessary subtasks have been detailed in the procedural knowledge base, we consider a related query irrelevant to accomplishing the task if it is not mentioned in any of the subtasks, and it is excluded in the parallel corpus and discarded for future iterations. We also ignore subtasks that are not matched to any related query, a situation that may arise from a lack of evidence from query logs, or truncation due to limited space for displaying suggested queries on a result page. We continue the iterative process.

We use the annotated corpus to train supervised models that can automatically create task-oriented queries for tasks. We view Problem 3 as a word sequence extraction problem, which is usually reduced to a sequence labeling task. For training a sequence labeling model, we employ the BIO encoding, i.e., each token in the task description that begins each occurrence of the query phrase is labeled as $B$, the tokens that continue in the query phrase are labeled as $I$, and all other tokens in the task are labeled as $O$. We use the features defined in Section 3.2.1 and then apply a common statistical model for sequence labeling (such as Conditional Random Fields (CRF)), which also enables us to process articles that do not fully comply with a style guide. We train a query construction model $M$ using the training sets $(X', Y')$. To construct search queries
for an unseen task $t$, we first extract the features $x^t$ from the task description, and then apply the query construction model $M^Q$ to extract each search query candidate $q_i$, i.e., to identify each word sequence $w_i = w_{i1} \ldots w_{il_i}$ from $t$ with the corresponding $y^t_i$ labeled as $B^Q I^Q \ldots I^Q$, where $l_i$ represents the length of $w_i$, and $y^t_i$ is the optimal label sequence, i.e.,

$$y^t_i = \arg\max_{y^t \in \{B^Q, I^Q, O\}^{l_i}} p(y^t|x^t; M^Q)$$ (3.1)

where $|t|$ represents the length of the task $t$’s description.

Finally, we present our proposed approach for search task suggestion. Given a task-oriented search task represented by query $q$, we first retrieve a list of candidate tasks from the procedural knowledge base that mention the query $q$ in either the summary or the explanation. When more than one task is returned from the procedural knowledge base, we need to determine which task is the best fit for the user’s search intent. Therefore, we leverage the query construction model $M^Q$ to estimate the likelihood of the $j$-th occurrence of the query $q$ in a retrieved task $t_i$’s description (i.e. a word sequence $w_{ij} = w_{ij1} \ldots w_{ijl}$, where $l$ is the length of query $q$). We select the $i^*\text{-th}$ task that contains the $j^*\text{-th}$ word sequence which maximizes the conditional probability, i.e.

$$i^*, j^* = \arg\max_{i,j} p(y^t_{ij} = B^Q I^Q \ldots I^Q|x^t; M^Q)$$ (3.2)

Once we have identified the task $t$, we identify the tasks in the procedural knowledge base for future query suggestion using a few heuristic strategies. First, we may select only the first-level subtasks and order the weight of the is-achieved-by relation in descending order. We may also select both the first-level and second-level subtasks and use the weight of each first-level subtask or the weight of each second-level subtask multiplied by the weight of its corresponding first-level subtask.

When a list of subtask candidates $t_1, \ldots, t_n$ are retrieved from the existing procedural knowledge base, we apply the query construction model $M^Q$ again, for each subtask $t_i$’s summary and explanation to identify each word sequence $w_{ij}$ labeled as $B^Q I^Q \ldots I^Q$ using Eq. 3.1. Among the extracted query candidates $\{w_{ij}\}_j$ for each subtask $t_i$, we choose the query $q_j$ that maximizes Eq. 3.2. The “winning” query candidates $q_1, \ldots, q_n$ corresponding to the subtask candidates are globally ranked by multiplying the weight obtained from subtask retrieval with the likelihood estimation; finally, we select the top-$k$ queries. This process guarantees that the queries are extracted from distinct subtask candidates, which can lead to the most coverage of all necessary subtasks. Alternatively, a diversity-aware ranking approach can also be applied to ranking all query candidates $\{w_{ij}\}_j$ globally.

### 3.3.2 Data Preparation

We used two publicly available data sets: an English wikiHow data dump and the AOL search log and public search engines (Google and Bing) to collect the suggested queries. The data and code used in this section are available to download:

[https://github.com/ziy/pkb](https://github.com/ziy/pkb)
**English wikiHow data dump.** We crawled an English wikiHow data dump using a modified version of the WikiTeam tool [8] which contains 198,163 non-redirect articles within namespace “0” (Main[9]). We also filtered out the articles that are marked as stub (incomplete and need more information) or have no “Introduction” or “Steps”, which results in a collection of 149,975 valid articles. We performed minimal pre-processing to reformat the articles to the MediaWiki format [10] without changing any textual content.

In particular, we first extracted the top-level task summary and explanation after identifying the title and the introduction of each article. We then located the “Steps” section and extracted the enumerated or bulleted items to build a local subtask hierarchy. Next, we built a procedural knowledge graph by creating nodes representing all the top-level tasks and their subtasks and establishing relations based on both the task-subtask relation as well as internal links. Applying this approach, the constructed procedural knowledge graph contains a total of 1,488,587 tasks and 1,439,217 relations, where 265,509 of the tasks are non-primitive, and 100,605 relations come from internal links. We built a Lucene [11] index for all task descriptions, which supports retrieval of candidate tasks for the search task suggestion problem described in Section 3.3.1.

**AOL search query log.** We also used the publicly available AOL search query log, which consists of over 21M Web queries (over 10M unique queries) collected over three months in 2006 from AOL [149]. We downcased the query strings and task summaries, and removed all non-alphanumeric characters. We identified that 867 unique queries (corresponding to 9,847 new queries from 23,099 lines) match some task summary in our constructed procedural knowledge base. In the experiment, we identified 639 unique queries that have at least two tokens (corresponding to 3,086 new queries from 7,019 lines), which tend to be less ambiguous and more likely task-oriented. We could estimate that each task-oriented search task is repeated 4.8 times on average, compared to 2.1 times for all queries, supporting the intuition that common task-oriented searches tasks are more often encountered than general search tasks, providing further motivation for our study. The 639 unique queries correspond to 619 tasks if punctuation marks and whitespaces are ignored.

To retrieve the related queries issued by users in the same session from the query log, we first identified related query candidates by collecting the queries that were issued by the same user within 30 minutes after they issued each matching query. In this way, we collected 33,548 query candidates (31,955 unique queries, 50 per query). We did not use other commonly adopted session detection heuristics such as edit distance or cosine similarity between query pairs, since a task-oriented search query may not share words in common with related search queries. For example, we identified a case where “visit niagara falls” [12] is followed by another query “map of northeast” in the AOL search log, “design a living room” [13] is followed by another query “choosing colors”.

---

[8]: https://github.com/WikiTeam/wikiteam
[10]: https://www.mediawiki.org/wiki/Help:Formatting
[12]: Also on wikiHow: http://www.wikihow.com/Visit-Niagara-Falls
[13]: Also on wikiHow: http://www.wikihow.com/Design-a-Living-Room
Table 3.2: Averaged number and percentage of related queries suggested by Google or Bing or issued by the users subsequently in the same session that are mentioned by a task description in wikiHow

<table>
<thead>
<tr>
<th></th>
<th>Averaged number</th>
<th>Percentage (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Google</td>
<td>Bing</td>
</tr>
<tr>
<td>Full phrase</td>
<td>0.070</td>
<td>0.046</td>
</tr>
<tr>
<td>New words</td>
<td>0.473</td>
<td>0.592</td>
</tr>
</tbody>
</table>

etc. We instead counted on the annotated corpus construction process (described in Section 3.3.1) to correctly identify which of the query candidates are relevant to accomplishing the task.

**Queries suggested by search engines.** To create “artificial” search tasks, we randomly sampled 1,000 non-primitive tasks from the constructed procedural knowledge that do not appear in the query log. We limited the total number of tasks in building the parallel corpus partly due to our limited access to commercial search engines, and also because in a preliminary experiment we found that performance of the sequence labeling model on the test set had become stable. We merged them with the 639 identified queries from AOL search query log, using commercial search engines (Google and Bing) to build the parallel corpus, which correspond to 1,619 tasks. We constructed each query without using additional operators such as quotes, plus signs, minus sign, etc., unless quotes appear in the summary. Google may suggest up to eight queries at the bottom of the search page for each search task and Bing may suggest up to sixteen queries within the search result column and on the right side of the results page. We collected a total of 9,906 queries suggested by Google (6.11 per query on average, 8 maximum) and 9,715 suggested queries by Bing (5.99 per query on average, 13 maximum).

We notice that both Google and Bing are able to display procedures directly on the result page from wikiHow (only Bing) or other procedural data sources (Google), which allows users to grasp the knowledge without an extra click, but hardly helps users or automatic decision support systems identify how to explore more related information from the Web. We also found that both commercial search engines tend to suggest and display related queries for short queries, as they are more likely topically broad and semantically ambiguous. However, for task-oriented search, the complexity of describing the task does not necessarily reflect the difficulty of accomplishing the task, in terms of which aspects to consider and the steps to perform.

**Annotated corpus.** Including the 639 queries that have already been associated with some tasks in the procedural knowledge base, we identified that a total of 1,182 query-task description pairs (from 1,146 distinct descriptions) among the suggested queries can be found in the subtask descriptions of the task corresponding to the original query.

To further understand how many of the related queries are aligned to a task description, we show in Table 3.2 the averaged number and percentage of related queries suggested by Google or Bing or issued by users subsequently in the same session that are mentioned within a task description. “New words” refer to the subset of words that are in the suggested queries but not in the original queries. First, we can see that the queries identified in the same session do not
have a higher quality (1.131% – 4.639%) in related task suggestion, which may be due to an oversimplified session detection algorithm. In comparison, related queries collected from commercial search engines tend to be more relevant (0.727% – 9.522%). Moreover, if we focus on the new words, we identify many more (from 0.727% – 1.162% to 8.487% – 9.522%) tasks that are aligned with the queries suggested by the search engines.

We note that if a query cannot be aligned with a task description using our proposed construction method, this does not guarantee that the query is not semantically identical (e.g. a paraphrase) of any task description. In fact, as motivated in Section 3.3.1, the surface form matching approach can simplify the parallel corpus construction process while still guarantee the parallelity.

### 3.3.3 Experiment Settings

We describe the experiment settings, including the tools we used to extract features and learn the sequence labeling models, the evaluation methods, and baseline methods.

**Evaluation methods.** We conducted two types of evaluation for both problems. We first evaluate the performance of proposed sequence labeling approach on the query construction task using the annotated corpus. Then, based on manual judgment, we evaluate our proposed end-to-end solution and compare with commercial search engine suggestions.

We performed 10-fold cross validation on the constructed parallel corpus, and report average performance results. We extracted text spans from the predicted labels and used precision, recall and F-1, averaged over the performance on all test instances (macro-averaged) and averaged on each task then across all tasks (micro-averaged), to compare the proposed approach with baseline methods. We also employed two ROUGE scores (F-1 based ROUGE-2 and ROUGE-S4\textsuperscript{14}) after we removed common English stop words\textsuperscript{15} and constructed an array of words in lemma forms.

In the end-to-end comparison, we randomly sampled 50 triples from the corpus and we applied both our approach for search task suggestion and commercial search engine services to produce up to eight related queries. We manually judged whether each search query can obtain a search result that can help achieve the user’s goal. Given that this judgment is binary, we report the macro-averaged and micro-averaged Precision@8, and MAP averaged over all 50 test instances.

In the NTCIR-11 IMine task\textsuperscript{119}, importance judgment was conducted manually by the annotators for each subtopic/subtask to obtain a ranked list of gold-standard tasks for a given topic/task, which allowed the use of nDCG and D#-measures\textsuperscript{170} for performance evaluation. However, since semi-structured procedural knowledge bases such as wikiHow or eHow and proposed situation ontologies\textsuperscript{11, 47, 92, 148} do not explicitly specify an importance rating for each subtask, we consider the subtasks to be unordered.

**Baseline methods.** For the query construction task, we compare the proposed approach CRF with other supervised and unsupervised approaches. We varied the classifier to evaluate HMM, a Hidden Markov Model based sequence labeling approach based on surface forms, LR, a logistic regression classifier, and an SVM classifier. The latter two are trained on the same set of features.

\textsuperscript{14}We did not use any unigram based metric (such as ROUGE-1 or ROUGE-SU) or recall based metric because we did not set a limit for either the number of chunks or the total length of the text returned for each task.

\textsuperscript{15}We used \url{/resources/stopwords-rouge-default.txt} from ROUGE 2.0 tool. \url{http://kavita-ganesan.com/content/rouge-2.0}
to classify $B$, $I$, and $O$ without the sequence labeling assumption. We also compared performance with models that use only one category of the proposed features ($W$), all but one category ($W/O$), and only local or context features. Finally, we compare with a key word extraction method based on TF-IDF ($T F I D F$), where we try to maximize the macro-averaged F-1 score by tuning the TF-IDF score threshold when determining whether a word is selected as a key word.

**Feature extractors and learners.** We used Stanford CoreNLP\(^{16}\) to extract sentences, tokens, stems, POS tags, dependency labels, chunks (noun phrases and verb phrases), and named entities using the default configuration. We used MALLET\(^{17}\)\(^{128}\) to learn and test the sequence labeling models (CRF and HMM) with sparse weighting method, LibLinear\(^{18}\) for training and testing LR and SVM models. As the Web documents tend to be noisy, we preprocessed the texts by inserting periods at certain places to improve the parsing performance. Details can be found in the source code.

### 3.3.4 Experimental Results

We first compare the query construction result between the proposed approach and the baseline methods in Table 3.3. We also conducted a t-test to calculate the significance level of each baseline method compared against the proposed approach (CRF). We can see the proposed CRF-based sequence labeling approach can significantly outperform other baseline classifiers (at a significance level of $\leq 0.05$ in terms of F-1 and $\leq 0.1$ in terms of ROUGE).

The performance gap between the sequence labeling approaches and the independent classification-based approaches such as LR and SVM suggests that the query construction problem is similar to other sequence labeling problems such as named entity detection or supervised key word extraction. With only surface form features, HMM performs worse than CRF. All the supervised approaches, which take advantage of the constructed parallel corpus, can outperform the unsupervised approach (TF-IDF).

When we test each feature category (POS, Parsing, Location, Word, Local, and Context) independently, we can see none of them can beat the proposed approach with a confidence level of $\geq 0.99$. We also see that word features contribute the most to performance, which can achieve around 90% of the performance in terms of F-1 and 95% in terms of ROUGE when all the features are used. Both F-1 and ROUGE scores drop when we remove any feature category from the feature list, and they drop the most when Word features are removed, which implies that all the features have positively contributed to the query construction task.

To better understand what non-Word features most likely promote a word to become a $B^O$ or $I^O$, we shows the top-5 features for $O \rightarrow I^O$, $B^O \rightarrow I^O$, $I^O \rightarrow I^O$ transitions in Table 3.4. We see that the whole query phrase is very likely extracted from the summary part of a description (sum) due to its clarity and conciseness. We also see that both singular nouns (NN or NNP) and verbs in either VB or VBP forms are selected as indicators for beginning a query, and verb phase (VP) is a useful feature when deciding whether to continue a query.

---

\(^{16}\) [http://nlp.stanford.edu/software/corenlp.shtml](http://nlp.stanford.edu/software/corenlp.shtml) Ver. 3.5.2

\(^{17}\) [http://mallet.cs.umass.edu/](http://mallet.cs.umass.edu/) Ver. 2.0.7

\(^{18}\) [http://www.csie.ntu.edu.tw/~cjlin/liblinear/](http://www.csie.ntu.edu.tw/~cjlin/liblinear/) Ver. 1.8
Table 3.3: Search task suggestion results. A dagger (†) and one to three stars (⋆) represent significance levels of $p \leq 0.1, 0.05, 0.01,$ and $0.001$ respectively, and ns represents not significant.

<table>
<thead>
<tr>
<th>Classifiers</th>
<th>Ma P</th>
<th>Ma R</th>
<th>Ma F1</th>
<th>Mi P</th>
<th>Mi R</th>
<th>Mi F1</th>
<th>R-2</th>
<th>R-S4</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRF</td>
<td>.8691</td>
<td><strong>.6563</strong></td>
<td><strong>.7471</strong></td>
<td>.9359</td>
<td><strong>.6934</strong></td>
<td><strong>.6930</strong></td>
<td><strong>.8112</strong></td>
<td><strong>.8087</strong></td>
</tr>
<tr>
<td>HMM</td>
<td>.5281†</td>
<td><strong>.1984</strong></td>
<td><strong>.2870</strong></td>
<td>.9160†</td>
<td>.3113†</td>
<td>.3161†</td>
<td>.5200†</td>
<td>.5147†</td>
</tr>
<tr>
<td>SVM</td>
<td>.7589**</td>
<td>.6428†</td>
<td><strong>.6955</strong></td>
<td>.8641**</td>
<td>.6674*</td>
<td>.6612*</td>
<td>.7922*</td>
<td>.7892*</td>
</tr>
<tr>
<td>LR</td>
<td>.8461ns</td>
<td>.5728**</td>
<td>.6826**</td>
<td>.9142*</td>
<td>.6078**</td>
<td>.6144**</td>
<td>.7476**</td>
<td>.7446**</td>
</tr>
<tr>
<td>TFIDF</td>
<td>.0087**</td>
<td>.0072**</td>
<td>.0079**</td>
<td>.4431**</td>
<td>.0262**</td>
<td>.0244**</td>
<td>.2279**</td>
<td>.2041**</td>
</tr>
<tr>
<td>W/ subsets of features (+context)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>POS</td>
<td>.8346†</td>
<td>.4720**</td>
<td>.6015**</td>
<td>.9248ns</td>
<td>.5487**</td>
<td>.5541**</td>
<td>.7108**</td>
<td>.7079**</td>
</tr>
<tr>
<td>PAR</td>
<td>.6846**</td>
<td>.3245**</td>
<td>.4390**</td>
<td>.8811**</td>
<td>.3973**</td>
<td>.4018**</td>
<td>.6020**</td>
<td>.5954**</td>
</tr>
<tr>
<td>LOC</td>
<td>.2476**</td>
<td>.0900**</td>
<td>.1241**</td>
<td>.5980*</td>
<td>.1671**</td>
<td>.1678**</td>
<td>.3488**</td>
<td>.3420**</td>
</tr>
<tr>
<td>WORD</td>
<td>.8164**</td>
<td>.5849**</td>
<td>.6803**</td>
<td>.9105**</td>
<td>.6185**</td>
<td>.6175**</td>
<td>.7713**</td>
<td>.7657**</td>
</tr>
<tr>
<td>W/O subsets of features (+context)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>POS</td>
<td>.8811†</td>
<td>.6493ns</td>
<td>.7466ns</td>
<td>.9400ns</td>
<td>.6856†</td>
<td>.6870†</td>
<td>.8113ns</td>
<td>.8082ns</td>
</tr>
<tr>
<td>PAR</td>
<td>.8689ns</td>
<td>.6519ns</td>
<td>.7442ns</td>
<td>.9357ns</td>
<td>.6864ns</td>
<td>.6861†</td>
<td>.8069†</td>
<td>.8044†</td>
</tr>
<tr>
<td>LOC</td>
<td>.8431**</td>
<td>.6470*</td>
<td>.7311**</td>
<td>.9153**</td>
<td>.6800*</td>
<td>.6785*</td>
<td>.8104ns</td>
<td>.8065ns</td>
</tr>
<tr>
<td>WORD</td>
<td>.8699ns</td>
<td>.5133**</td>
<td>.6444**</td>
<td><strong>.9411</strong></td>
<td>.5867***</td>
<td>.5911***</td>
<td>.7322**</td>
<td>.7299**</td>
</tr>
<tr>
<td>Local</td>
<td>.7808**</td>
<td>.5935**</td>
<td>.6737**</td>
<td>.8935**</td>
<td>.6449**</td>
<td>.6447**</td>
<td>.7807**</td>
<td>.7746**</td>
</tr>
<tr>
<td>Context</td>
<td>.8155**</td>
<td>.6248**</td>
<td>.7068**</td>
<td>.8983**</td>
<td>.6538**</td>
<td>.6535**</td>
<td>.7907*</td>
<td>.7857**</td>
</tr>
</tbody>
</table>

We then evaluated the end-to-end performance of the proposed STS solution using the query construction model. We present the results in Table 3.5 and show examples in Table 3.6. We see the proposed solution can outperform the commercial search engines by +0.35 to +0.41 in terms of Precision@8 and +0.28 to +0.32 in terms of MAP. We see that our proposed method, which is tailored for task-oriented search, can provide unique insights when suggesting tasks, compared to current general-purpose commercial search engines, which have been designed for entity-centric search and tend to suggest queries by appending keywords such as “product”, “image”, “logo”, “online”, “free”, etc. We discovered three types of errors from our system: (1) some suggested queries are ambiguous when presented by themselves (e.g. “fix it”, “install”), (2) duplicated queries are suggested from different subtasks, and (3) the manually created knowledge sometimes still contains a few non-instructional “steps”. We could further improve the proposed approach by conducting a co-reference analysis or other approaches to incorporate original queries and contexts in the suggested queries. We could also collectively rank the candidates to avoid suggesting duplicated tasks.
Table 3.4: The non-Word features that contribute the most to each label in the search task suggestion problem. “+1”/“-1” presents the feature of the next/previous word, “sum” refers to summary, “VP” represents if the word is inside a verb.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>O → B⁰</td>
<td>POS: NNP</td>
<td>LOC: sum</td>
<td>DEP: ccomp</td>
<td>POS: VB</td>
<td>DEP: nsubjpass</td>
</tr>
<tr>
<td>B⁰ → I⁰</td>
<td>POS¹: VB</td>
<td>LOC: sum</td>
<td>POS¹: VBP</td>
<td>POS¹: NNP</td>
<td>POS¹: NN</td>
</tr>
<tr>
<td>I⁰ → I⁰</td>
<td>LOC: sum</td>
<td>POS¹: IN</td>
<td>VP</td>
<td>DEP: dobj</td>
<td>POS¹: JJ</td>
</tr>
</tbody>
</table>

Table 3.5: Search task suggestion results

<table>
<thead>
<tr>
<th></th>
<th>Proposed</th>
<th>GOOGLE</th>
<th>BING</th>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>Macro P</td>
<td>.4457</td>
<td>.0972</td>
<td>.0333</td>
<td>.0676</td>
</tr>
<tr>
<td>Micro P</td>
<td>.4457</td>
<td>.0973</td>
<td>.0313</td>
<td>.0612</td>
</tr>
<tr>
<td>MAP</td>
<td>.3361</td>
<td>.0553</td>
<td>.0120</td>
<td>.0549</td>
</tr>
</tbody>
</table>

3.4 Procedural Knowledge Extraction from Search Activities

Although community-centric procedural knowledge bases may have collected hundreds of thousands of task descriptions, users still face ad hoc situations (tasks) that are not covered by an existing procedural knowledge base. From this section, we focus on procedural knowledge extraction, which is also referred to as **automatic procedural knowledge base construction**.

In this section, we hypothesize that other searchers may have faced similar situations in the past and have already interacted with search engines to attempt a solution, which means we may discover implicitly expressed procedural knowledge from users’ raw search activities (e.g. search logs) as well as those aggregated from many searchers (e.g. suggested queries). Therefore, we investigate whether we can reverse the process we have proposed in Section 3.3 and automatically construct a procedural knowledge base like wikiHow using search queries and relevant documents returned from search engines.

**Problem 4** (Automatic procedural knowledge base construction from search activities, APKBC). Given a task \( t \), we aim to (a) identify a search task \( q \), and (b) collect \( k \) related search tasks \( p_1, \ldots, p_k \), and then (c) identify \( n (\leq k) \) search tasks \( p_{i_1}, \ldots, p_{i_n} \) to generate \( n \) tasks \( s_1, \ldots, s_n \) that can be performed to accomplish the task \( t \) with text description.

Related search tasks can be identified from users’ search activities [30] [109] [162] and/or suggested by search engines [9] [38] [91] [129] [213]. Depending on the search intent, related actions, attributes, entities, or any form of reformulation of the original search task may be considered related, which however do not necessarily embed procedural knowledge. Therefore, the first challenge in extracting procedural knowledge is to identify the is-achieved-by relations, i.e., which of them correspond to the tasks that can achieve the goal. A textual description is a less ambiguous representation of a task in a procedural knowledge base, which can be accessed both by humans and by automatic decision support systems that require predefined natural language decision pro-
Table 3.6: Comparison of top 4 suggested search tasks produced by the proposed STS system and commercial search engines for example search tasks.

<table>
<thead>
<tr>
<th>Proposed</th>
<th>GOOGLE</th>
<th>BING</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Task: make blueberry banana bread</strong></td>
<td>blend the mixture</td>
<td>vegan blueberry banana bread</td>
</tr>
<tr>
<td></td>
<td>mix flour</td>
<td>buttermilk blueberry banana bread</td>
</tr>
<tr>
<td></td>
<td>add 1 egg</td>
<td>great blueberry banana bread</td>
</tr>
<tr>
<td></td>
<td>vanilla extract</td>
<td>blueberry banana bread with yogurt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>healthy banana blueberry bread</td>
</tr>
<tr>
<td><strong>Task: slim down</strong></td>
<td>weight loss</td>
<td>slim down diet</td>
</tr>
<tr>
<td></td>
<td>heavy food</td>
<td>7 day slim down</td>
</tr>
<tr>
<td></td>
<td>junk food</td>
<td>weight loss</td>
</tr>
<tr>
<td></td>
<td>keep up the mood</td>
<td>slim down thighs</td>
</tr>
<tr>
<td><strong>Task: play red alert2</strong></td>
<td>build a barracks</td>
<td>red alert 2 complete (iso) original 2 disc</td>
</tr>
<tr>
<td></td>
<td>build a war factory</td>
<td>play red alert 2 free</td>
</tr>
<tr>
<td></td>
<td>radar should</td>
<td>play red alert 2 online free</td>
</tr>
<tr>
<td></td>
<td>build a power plant/tesla reactor</td>
<td>play red alert 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>play ra2 online</td>
</tr>
<tr>
<td></td>
<td></td>
<td>red alert 2 download</td>
</tr>
<tr>
<td></td>
<td></td>
<td>free red alert 3</td>
</tr>
</tbody>
</table>
necessarily optimized for interactive search. We first describe how to extend the annotated corpus defined in Section 3.3.1 and further build a three-way parallel corpus for training, and then define linguistic and structural features for training sequence labeling models for both the search task suggestion problem and the automatic procedural knowledge base construction problem.

We create a three-way parallel corpus that contains triples \( \{ q, t, c \} \) with a query \( q \), a matching task \( t \), and a context \( c \) of \( q \) and \( t \), which is a passage of text that describes the task. We build the three-way parallel corpus in an iterative manner.

We first identify the exact matching pairs of searchers’ issued search tasks and task descriptions in the procedural knowledge base, which guarantees precision in the first step, similar to the annotated corpus construction process, and can be achieved by scanning through the entire search query log to find each query \( q \) that matches the description of the task \( t \). We then retrieve the context \( c \) by extracting the textual content from the top documents relevant to the description of the task \( t \). We add the triple \( \langle q, t, c \rangle \) to the parallel corpus. We also collect related queries by combining the user-issued queries from the same session and the list of queries suggested by the search engine for the next iteration. If necessary, we may also use the summary of the task \( t \) to form an “artificial” search query \( q \) and issue it to the search engine to extract the context \( c \). Again, we exclude the triple \( \langle q, t, c \rangle \) that contains the original query \( q \) from the parallel corpus, which is only used for search engines to suggest real queries for the next iteration. Then, we compare the related queries for \( q \) with the known subtasks of \( t \). For each related query \( q' \), we first find the subtasks \( t_1, \ldots, t_n \) that contain \( q' \) in either its summary or explanation, and retrieve its context \( c' \). We then add all \( \{ \langle q', t_i, c' \rangle \}_{i \in [1,n]} \) to the parallel corpus. We continue the iterative process.

Furthermore, we annotate the occurrence of the query \( q \) in the task \( t \)’s description. We also similarly annotate the occurrence of the summary and explanation of the task \( t \) in the context \( c \) by finding the contiguous sequence of words (a passage) from the context that is most relevant to the task summary or explanation. An example is shown in Figure 3.2. For training a sequence labeling model for automatic procedural knowledge base construction problem, we also employ the BIO encoding, i.e., each token in the task description (or context) that begins each occurrence of the query phrase (or the task summary or explanation) is labeled as \( B^Q \) (or \( B^{TS}, B^{TE} \)), the tokens that

Figure 3.1: Subproblems in search task suggestion using procedural knowledge problem (brown solid arrows and brown labels) and automatic procedural knowledge base construction from search activities problem (purple dashed arrows and purple labels)
continue in the query phrase (or the task summary or explanation) are labeled as $I^Q$ (or $I^{TS}$, $I^{TE}$), and all other tokens in the task (or the context) are labeled as $O$.

We use the constructed parallel corpus, which contains triples of queries, task representations, and contexts, to train supervised models that can automatically create task-oriented queries for tasks, as well as task descriptions for task-oriented search tasks. We extract the same set of features (except the location feature) from both the context $c$ ($X^c$) and task description $t$ ($X^t$) for each triple in the parallel corpus, which together with the corresponding annotated BIO label ($Y^c$ and $Y^t$) comprises a training instance for sequence labeling. We train a query construction model $M^Q$ using the training sets ($X^t, Y^t$), a task summary construction model $M^{TS}$ and a task explanation construction model $M^{TE}$ using ($X^c, Y^c$).

The training and prediction process for search task suggestion is the same as in Section 3.3.1. Similarly, to construct task descriptions for an unseen query $q$, we first retrieve the context $c$ and extract the features $x^c$ from the context, and then apply the task description construction model $M^c$ to extract each word sequence labeled as $B^{TS} I^{TS} \ldots I^{TS}$ and $B^{TE} I^{TE} \ldots I^{TE}$ as the summary and the explanation of a task $t_i$ respectively from optimizing the following equation:

$$y^{ct} = \arg\max_{y^c \in \{B^c, I^c, O\}^{|c|}} p(y^c|x^c, M^T)$$  \hspace{1cm} (3.3)

Finally, given a task description $t$, we directly apply the query construction model $M^Q$ to extract a task-oriented search query $q$ using Eq. 3.1 and then (similar to the process in building the parallel
corpus) we identify the queries related to \( q \) in both search logs and suggested queries. The entire search session (reformulation of queries, click activities, etc.) can be analyzed to determine how a specific user works to accomplish a task; therefore, by mining the search session data, we should be able to model how users typically accomplish tasks. As a result, the search engine should be able to correctly suggest to the user related tasks, rather than related entities or attributes. Although this assumption may not always hold in real-world search scenarios, it allows us to consider how related tasks can be further explored to extract subtasks for \( t \), as long as the task description construction model can identify any passage that can be used for the task summary or explanation.

For each related query \( p_i \), we collect its context by extracting relevant document snippets from search engines, and apply the task description construction model \( M_{\text{T}} \) to extract the most likely summary candidate \( w_i \) and explanation candidate \( v_i \) for task \( s_i \), according to Eq. \[3.3\] among all extracted summary candidates and explanation candidates for \( s_i \). We select the top-\( n \) tasks ordered by the estimated likelihood for their summaries.

### 3.4.2 Data Preparation

In addition to the two publicly available data sets (an English wikiHow data dump and the AOL search log) described in Section 3.3.2, we also leveraged public search engines (Google and Bing) to collect contexts. The data and code used in this section are available to download \[19\].

We used Google to collect the contexts of the queries both in the parallel corpus (used for model training and testing) and those that are potentially useful for new procedural knowledge. We first extracted URLs for context candidates from their first search result pages, and excluded Web documents from the wikihow.com domain to increase the reliability and generalizability of the trained model, and also excluded google.com domains and URLs that have only domain names without subpaths, which are usually navigational search results. Then, we used the Boilerpipe toolkit \[20\] to extract textual content from the HTML documents. For queries in the parallel corpus, we finally downloaded 7,440 context documents and successfully processed 7,437 documents using Boilerpipe, and for end-to-end evaluation, we downloaded and extracted 3,512 context documents.

### 3.4.3 Experiment Settings

The baseline methods, feature extractors and learners are the same as those described in Section 3.3.3. When we evaluate our proposed end-to-end solution for the automatic procedural knowledge base construction problem, we compare with the current wikiHow knowledge base. In particular, we applied our procedural knowledge base construction approach to generate up to eight procedural descriptions and merged it with the wikiHow “steps”. We manually judged whether each subtask summary and explanation that the system produced can be considered a valid “step” description for wikiHow. We also report the macro-averaged and micro-averaged Precision@8, and MAP averaged over all 50 test instances.

\[19\] https://github.com/ziy/pkb
\[20\] https://github.com/kohlschutter/boilerpipe Ver. 1.2.0

49
Table 3.7: Task summary generation results. A dagger (†) and one to three stars (*) represent significance levels of \( p \leq 0.1, 0.05, 0.01, \) and 0.001 respectively, and ns represents not significant.

<table>
<thead>
<tr>
<th>Classifiers</th>
<th>Ma P</th>
<th>Ma R</th>
<th>Ma F1</th>
<th>Mi P</th>
<th>Mi R</th>
<th>Mi F1</th>
<th>R-2</th>
<th>R-S4</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRF</td>
<td>.7576</td>
<td>.3260</td>
<td>.4207</td>
<td>.8524</td>
<td>.3344</td>
<td>.3455</td>
<td>.4463</td>
<td>.4392</td>
</tr>
<tr>
<td>HMM</td>
<td>.0042**</td>
<td>.0008**</td>
<td>.0014**</td>
<td>.1213**</td>
<td>.0010**</td>
<td>.0015**</td>
<td>.1017**</td>
<td>.0660**</td>
</tr>
<tr>
<td>SVM</td>
<td>.1464***</td>
<td>.1116**</td>
<td>.1175***</td>
<td>.4585***</td>
<td>.1041**</td>
<td>.119***</td>
<td>.2425**</td>
<td>.2301**</td>
</tr>
<tr>
<td>LR</td>
<td>.1131***</td>
<td>.0489**</td>
<td>.0668**</td>
<td>.4210**</td>
<td>.0476**</td>
<td>.0511**</td>
<td>.1734**</td>
<td>.1692**</td>
</tr>
<tr>
<td>TFIDF</td>
<td>.0160***</td>
<td>.1254*</td>
<td>.0271**</td>
<td>.0556***</td>
<td>.1347*</td>
<td>.0683**</td>
<td>.1583**</td>
<td>.1026**</td>
</tr>
</tbody>
</table>

W/ subsets of features (+context)

| POS         | .5325* | .2881ns | .7093** | .2923ns | .3153ns | .3822ns | .3788ns |
| PAR         | .3175** | .0641** | .0988** | .7893†  | .0838** | .1996** | .1996*  | .1968*  |
| WORD        | .7738ns | .2464** | .3279** | .9061*  | .2433** | .2526** | .3435** | .3412** |

W/O subsets of features (+context)

| POS         | .7489ns | .2651** | .3460** | .8813ns | .2576** | .2695** | .3678** | .3645** |
| PAR         | .7723ns | .3167ns | .4129ns | .8749†  | .3088†  | .3198†  | .4170*  | .4118*  |
| WORD        | .5359*  | .2945ns | .3632ns | .7084** | .3185ns | .3408ns | .3787ns | .3704ns |

Local        | .7590ns | .2757*  | .3755*  | .8439ns | .2734*  | .2902*  | .4053*  | .4006*  |
| Context     | .7736†  | .3113†  | .4081ns | .8604ns | .3159ns | .3260ns | .4069*  | .4022*  |

3.4.4 Experimental Results

We compare the results of our proposed approach with the baseline methods for task description (summary and explanation) construction in Tables 3.7 and 3.8. We first see that all the methods produce much lower F-1 and ROUGE scores for APKBC than those for STS, which suggests that task description generation is a difficult task, esp. explanation construction. Nevertheless, we can see that the proposed approach significantly outperforms all other classifiers in terms of F-1 and ROUGE metrics (at a significance level of \( \leq 0.1 \)).

When comparing with the results from using each feature category, we see that not a single category can reach the same performance level as the proposed approach, which again implies that all the feature categories have contributed to the overall performance in task description construction problem. Comparing with using all but one feature categories, we can find that Word features are crucial for summary generation but not explanation generation, whereas POS and Parsing features are crucial for explanation generation but not summary generation. We also observe the non-Word features that contribute the most in Table 3.9. Although the performance does not match that of the query construction approach, we can still find interesting patterns in the results. First, we can see that nouns and verbs are still crucial in constructing task descriptions, and verbs (VB and VBP) rather than nouns are more likely selected as the beginning of a summary. To begin the task ex-
Table 3.8: Task explanation generation results. A dagger (†) and one to three stars (⋆) represent significance levels of \( p \leq 0.1, 0.05, 0.01, \) and 0.001 respectively, and ns represents not significant.

<table>
<thead>
<tr>
<th>Classifiers</th>
<th>Ma P</th>
<th>Ma R</th>
<th>Ma F1</th>
<th>Mi P</th>
<th>Mi R</th>
<th>Mi F1</th>
<th>R-2</th>
<th>R-S4</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRF</td>
<td>.4984</td>
<td>.3853</td>
<td>.310</td>
<td>.7180</td>
<td>.3342</td>
<td>.3577</td>
<td>.3698</td>
<td>.3686</td>
</tr>
<tr>
<td>HMM</td>
<td>.0000</td>
<td>.0000</td>
<td>.0000</td>
<td>.4724</td>
<td>.0050</td>
<td>.0050</td>
<td>.2450</td>
<td>.2324</td>
</tr>
<tr>
<td>SVM</td>
<td>.0053</td>
<td>.0026</td>
<td>.0034</td>
<td>.4911</td>
<td>.0012</td>
<td>.0018</td>
<td>.1737</td>
<td>.1659</td>
</tr>
<tr>
<td>LR</td>
<td>.0000</td>
<td>.0000</td>
<td>.0000</td>
<td>.6978</td>
<td>.0000</td>
<td>.0000</td>
<td>.1470</td>
<td>.1408</td>
</tr>
<tr>
<td>TFIDF</td>
<td>.0002</td>
<td>.0015</td>
<td>.0003</td>
<td>.0623</td>
<td>.0363</td>
<td>.0103</td>
<td>.2657</td>
<td>.2498</td>
</tr>
<tr>
<td>W/ subsets of features (+context)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>POS</td>
<td>.1904</td>
<td>.0463</td>
<td>.0687</td>
<td>.7588</td>
<td>.0704</td>
<td>.0889</td>
<td>.1647</td>
<td>.1637</td>
</tr>
<tr>
<td>PAR</td>
<td>.7000</td>
<td>.0000</td>
<td>.0000</td>
<td>.9909</td>
<td>.0000</td>
<td>.0000</td>
<td>.1325</td>
<td>.1325</td>
</tr>
<tr>
<td>WORD</td>
<td>.4901</td>
<td>.3056</td>
<td>.3639</td>
<td>.7292</td>
<td>.2961</td>
<td>.3176</td>
<td>.3489</td>
<td>.3472</td>
</tr>
<tr>
<td>W/O subsets of features (+context)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>POS</td>
<td>.4722</td>
<td>.3198</td>
<td>.3695</td>
<td>.7206</td>
<td>.3411</td>
<td>.3554</td>
<td>.3580</td>
<td>.3554</td>
</tr>
<tr>
<td>PAR</td>
<td>.4837</td>
<td>.3169</td>
<td>.3718</td>
<td>.7274</td>
<td>.3208</td>
<td>.3468</td>
<td>.3804</td>
<td>.3793</td>
</tr>
<tr>
<td>WORD</td>
<td>.2546</td>
<td>.0247</td>
<td>.0410</td>
<td>.7575</td>
<td>.0194</td>
<td>.0292</td>
<td>.1614</td>
<td>.1614</td>
</tr>
<tr>
<td>Local</td>
<td>.1337</td>
<td>.0355</td>
<td>.0521</td>
<td>.6105</td>
<td>.0370</td>
<td>.0474</td>
<td>.1858</td>
<td>.1828</td>
</tr>
<tr>
<td>Context</td>
<td>.4534</td>
<td>.2879</td>
<td>.3559</td>
<td>.6823</td>
<td>.2922</td>
<td>.3532</td>
<td>.3500</td>
<td>ns</td>
</tr>
</tbody>
</table>

In explanation, the word is expected to have a “Begin” indicator and/or a dependency type of nsubj, meaning the nominal subject of a sentence. Verb phrases are also important in identifying a task description.

Finally, we evaluated our proposed end-to-end APKBC solution, and compared it with wikiHow articles in Table 3.10. Due to the difficulty of the task description construction problem and lack of high-quality task-oriented search query candidates, we find that the automatic approach performs much worse than manual curation in building a brand new procedural knowledge base from scratch; nevertheless, we still find that the proposed approach is able to discover relevant subtasks that are not covered in the current wikiHow article, i.e., it can be used to improve the coverage and quality of existing procedural knowledge bases. Some examples are shown in Table 3.11. For example, for the task “sign up for airbnb”, one of the suggested queries “sign up for airbnb coupon” implies a coupon may be an important subsidiary resource of a task that the current wikiHow article does not know. For this particular task, a statement with the detailed coupon info is able to be extracted to describe the concrete task related to the resource. Even though this result may be from an advertisement Web page, it still delivers the freshest information that can hardly be added and updated instantly in manually created procedural knowledge bases.

We also summarize the errors into three major categories: (1) ambiguous task representa-
Table 3.9: The non-Word features that contribute the most to each label in the automatic procedural knowledge base construction problem. “-1” presents the feature of the previous word, “VP”/“NP” represents if the word is inside a verb/noun phrase, “Begin” refers to the beginning of the text.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>summary ($M^{TS}$)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O $\rightarrow$ B$^{TS}$</td>
<td>POS: VB</td>
<td>POS: VBP</td>
<td>POS: NN</td>
<td>DEP: appos</td>
<td>POS: NNP</td>
</tr>
<tr>
<td>B$^{TS}$ $\rightarrow$ I$^{TS}$</td>
<td>POS$^{-1}$: VB</td>
<td>POS$^{-1}$: VBP</td>
<td>POS$^{-1}$: NNP</td>
<td>POS$^{-1}$: NN</td>
<td>DEP: case</td>
</tr>
<tr>
<td>I$^{TS}$ $\rightarrow$ I$^{TS}$</td>
<td>POS$^{-1}$: VBP</td>
<td>POS: NNP</td>
<td>POS$^{-1}$: IN</td>
<td>DEP: xcomp</td>
<td>POS: JJR</td>
</tr>
</tbody>
</table>

|     | | | | | |
| explanation ($M^{TE}$) | | | | | |
| O $\rightarrow$ B$^{TE}$ | Begin | POS: VBG | POS: NN | DEP: compound | DEP: nsubj |
| B$^{TE}$ $\rightarrow$ I$^{TE}$ | VP | POS$^{-1}$: NN | POS$^{-1}$: DT | NP | POS$^{-1}$: VB |
| I$^{TE}$ $\rightarrow$ I$^{TE}$ | POS$^{-1}$: NN | VP | POS$^{-1}$: NNS | POS$^{-1}$: , | POS$^{-1}$: NNP |

Table 3.10: Summary and explanation generation result for automatic procedural knowledge base construction

<table>
<thead>
<tr>
<th></th>
<th>Summary</th>
<th>Explanation</th>
<th>wikiHow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Macro P</td>
<td>.0997</td>
<td>.2046</td>
<td>.9677</td>
</tr>
<tr>
<td>Micro P</td>
<td>.0995</td>
<td>.2041</td>
<td>.9515</td>
</tr>
<tr>
<td>MAP</td>
<td>.0527</td>
<td>.1331</td>
<td>.9404</td>
</tr>
</tbody>
</table>

olutions. For example, the “find a match” article on wikiHow\footnote{http://www.wikihow.com/Find-a-Match} describes how to “look for a partner”, whereas search engines try to disambiguate the search intent by also suggesting queries in the context of data analysis such as “find a match in excel”, “find a match spreadsheet”, etc., (2) duplicated descriptions, and (3) quality of the text extracted from noisy Web documents. Moreover, as we note that not all wikiHow articles exactly follow the writing guide, the proposed approach can be used to estimate the quality of each existing task description, similar to Eq. 3.2, and report potentially suspicious low-quality articles or spurious steps such as “You’re done.”, which violate the writing guidelines.

3.5 Conclusion

In this chapter, we study a subproblem of analytics meta learning – procedural knowledge extraction, which may include extraction from semi-structured procedural text and causal relation and argumentation extraction. We first demonstrate how to leverage an existing procedural knowledge base for information retrieval applications, such as search task suggestion or query suggestion. This module is also used later for actionable concept identification in the case study in Chapter 9.
Table 3.11: Partial suggested subtasks from the top 8 outputs produced by the proposed APKBC system for example tasks.

**Task: sign up for airbnb**

Airbnb is no longer running the $50 OFF $200 promo but you can still save $25 OFF Your First Airbnb Stay of $75 or more by copying and pasting this link into your browser: ...

**Task: make blueberry banana bread**

Please don’t use regular whole wheat in this recipe – the loaf will turn out very dense
Add the wet ingredients - the egg mixture to the flour mixture and stir with a rubber spatula until just combined
If you’re in need of a quick, easy and delicious way to use up the ripe bananas in your house…definitely

**Task: become a cell phone dealer**

However, the cell phone provider may place restrictions on the manner in which you can use its company name, phone brands and images
Visit the state’s business licensing agency’s website and your city’s occupational/business licensing department’s website to determine if you need a license for your prepaid cell phone business

We then propose to mine procedural knowledge from various types of data sources. We propose to take advantage of search activities to identify task procedures from search query log. The experimental results show that the proposed methods outperform the commercial search engines and also provide useful supplementary materials to existing procedural knowledge bases.

We plan to extend our work on leveraging procedural knowledge for task-oriented search [219] and further explore a wider range of human-generated data sources, such as on-line community question answering sites, bibliographic databases, etc., and study the canonicalization of representation to facilitate analysis component construction and execution.
Chapter 4

Exploration Algorithm: A Hierarchical Bayesian Model Based Method

This chapter introduces a solution based on hierarchical Bayesian modeling and stochastic scheduling to the optimization problem in analytics meta learning – configuration space exploration with budget constraint, which aims to rapidly identify the optimal information system. This is a sub-problem of analytics space exploration that considers only the information pipelines with fixed analytics procedures, i.e. those that contain only execution processes.

In Section 4.1, we first present an exploration strategy based on hierarchical Bayesian model and stochastic scheduling. We report the experimental results on our preliminary study using the TREC Genomics QA task in Section 4.2, which involves a configuration space consisting of trillions of possible execution processes. We conclude in Section 4.3.

4.1 Configuration Space Exploration

In this section, we consider a simpler problem – configuration space exploration, which aims to optimize the configuration space for information pipelines with a fixed analytics procedure. We propose a novel formulation and solution to the associated constrained optimization problem of analytics meta learning (Problem 2), which comprises two sub-problems: component characteristics estimation and stochastic scheduling.

The utility of an information system can be measured by runtime performance and task performance, and in the constrained version of the analytics meta learning problem (Problem 2), the total cost of executing the entire analytics space is also measured. In order to estimate the performance at the configured component level, we introduce notations $c(f^\omega, x)$, $b(f^\omega, x)$, and $u(f^\omega, x)$ to capture two important characteristics of the configured component $f^\omega$: the cost of resource required to execute the component on input $x$, the benefit of executing the configured component to performance improvement, and the utility of the configured component, which can be defined as a function of cost and benefit, or

$$u(f^\omega, x) = h(b(f^\omega, x), c(f^\omega, x)).$$
The benefit of a single component is relatively difficult to measure without being integrated into a system with other components, where we can leverage commonly-used evaluation metrics (Definition [11]) for information systems, e.g. F-1 and MAP, etc. In information pipelines, we simply assume a configured component shares the benefit with the component that follows, if no direct measurement is available; the true cost or benefit of a configured component is estimated based on the cumulative effect over trillions of pipeline combinations.

We extend the concepts of cost and benefit for a configured component $f^ω$ to a system $s$ and a configuration space $S$: the cost to execute a system is the sum of costs to execute each configured component, and the performance of a system corresponds to the final output from last execution. They can be formally defined as follows:

$$c(s, x) = c(f^ω_{e}, x) = \sum_{i=1}^{n} c\left(f^ω_{i,e_i}, x_{e[1,i-1]}\right)$$  \hspace{1cm} (4.1)$$

$$b(s, x) = b(f^ω_{e}, x) = b\left(f^ω_{n,e_n}, x_{e[1,n-1]}\right)$$  \hspace{1cm} (4.2)$$

where $f^ω_{e}$ is a processing trace of $s$, which consists of $n$ consecutive steps, and $x_{e[i,j]}$ represents the output from the subtrace $f^ω_{e[i,j]}$ with input of $f^ω_{e_i}$ being $x$, i.e.

$$x_{e[i,j]} = f^ω_{e[i,j]}(x) = f^ω_{j,e_j} \left(\ldots \left(f^ω_{i,e_i}(x)\right)\right)$$

The output is used as the input for $f^ω_{e_i}$. The utility of the configured component can also be defined as a function of cost and benefit, or

$$u(s, x) = h(b(s, x), c(s, x)).$$

The cost of the entire configuration subspace is defined as the sum of unique executions of configured components on all outputs from previous steps. The benefit or utility of the configuration space is defined as that of the best-performing system.

$$c(S, x) = \sum_{i=1}^{n} \sum_{c_i=1}^{m_i} \ldots \sum_{c_i=1}^{m_i} c\left(f^ω_{i,e_i}, x_{e[1,i-1]}\right)$$  \hspace{1cm} (4.3)$$

$$b(S, x) = \max_{s \in S} b(s, x)$$  \hspace{1cm} (4.4)$$

$$u(S, x) = \max_{s \in S} u(s, x)$$  \hspace{1cm} (4.5)$$

where $m_i$ is the number of configured components at step $i$, and $c[1,i-1]$ is indeed $(1, c_1), \ldots, (i-1, c_{i-1})$. We see that $c(S, x) \neq \sum_{s \in S} c(s, x)$.

In fact, if an information pipeline $s$ has been executed, and another information pipeline $s'$ has the same prefix, i.e.,

$$f^ω_{1,c_1}, \ldots, f^ω_{i_{1},c_{i_{1}-1}} = f^ω_{1,c'_1}, \ldots, f^ω_{i_{t},c'_{i_{t}-1}}$$

for some $t$, then we do not need to repeat the executions for the configured components along the prefix, which is one of the key ideas for the problem solution and implementation.

The analytics meta learning problem description (Problem [2]) appears isomorphic to constrained optimization. However, in the analytics meta learning case, both cost and benefit of a configured component are unknown to the model until the component is executed. In the next three sections, we describe an exploration strategy to the configuration space exploration problem.
4.1.1 Solution Overview

In our initial case study, we adopt the constraint that each parameter must have a finite number of values, in order to support an efficient practical implementation and experimentation. Moreover, we note the problem in this section focuses on determining the globally optimal system general to all inputs, whereas researchers have observed that some components or configurations might improve the system performance on only some types of inputs while hurt the performance on other types. Accordingly, one can further extend the solution to a variant also incorporating the dependency on the input type.

We note the constrained optimization problem (Problem 2) is trivial if the total cost to iteratively execute all the components and configurations for all inputs does not exceed available processing capacity. However, this is not the case for most real world problems, since the number of systems and executions grow exponentially as the number of steps increases. A typical information processing pipeline consisting of 12 components is shown in Section 4.2; with up to four parameters per component, and up to six options each parameter, there would be an estimated $6.050 \times 10^{13}$ executions if all the unique systems were evaluated.

In an ideal case where (1) the exact cost $c(f^\omega, x)$ of each configured component on each input is known, (2) the benefit $b(f^\omega, x)$ of each configured component $f^\omega$ is an i.i.d. random variable, and (3) the utility only considers the benefit, i.e. $u(f^\omega, x) = b(f^\omega, x)$, the optimal solution to this problem can be yielded by adding configured components and inputs in descending order of least cumulative cost (LCC), which is defined as the sum of the component’s original cost plus minimal additional cost to execute the components down the pipeline, and formulated as follows:

$$\text{LCC} \left( f^\omega_{i,c}, x \right) = c \left( f^\omega_{i,c}, x \right) + \sum_{j=i+1}^{n} \min_{c' = 1}^{m_j} c \left( f^\omega_{j,c'}, x \right)$$  \hspace{1cm} (4.6)

When we relax the first assumption used for the simplified case (the cost is revealed until the execution is done), the approaches for deterministic scheduling problems do not provide an adequate solution. Instead, we refer to the stochastic scheduling problem or stochastic knapsack problem, which makes a different assumption that job durations are random variables with known probability distributions. These approaches provide a more general formulation suitable for the configuration space exploration problem.

We also relax the second assumption and assume the performance of a component is not i.i.d. For example, a weak configured component $f^\omega$ (e.g. a naive approach or buggy code) tends to exhibit low performance for every input $x$; on the other hand, a complex input $x$ (e.g., a very difficult question posed to a question answering system) may have a negative effect on the performance of all configured components. We are inspired to leverage dependency of characteristics.

To incorporate continuous parameters, one can discretize the feasible region by equally binning the region or sampling values from it. Directly tackling an optimization problem with continuous parameters requires a better understanding of how the behavior of the optimization objective varies as the parameter changes (e.g., linearity, convexity, etc.), and further relies on various continuous optimization techniques. Extending the exploration strategy for continuous parameter values is left to future work.

As the intermediate data are persisted, a post-performance analysis can be easily conducted to identify the correlation between the input type and the system performance.
Figure 4.1: Modeling cost and benefit for analytics space exploration

and prior knowledge to dynamically (and more wisely) select which system to evaluate next from the pool. We can utilize the execution history to estimate \( c(\omega, x) \) and \( b(\omega, x) \) by establishing a dependency between them. Intuitively, components that previously showed high benefit (or low cost) tend to achieve again high benefit (or low cost) combined with components from other steps, and components of low benefit (or high cost) tend to be pruned from the pool. In addition, we can also estimate the cost from prior knowledge, e.g., retrieving relevant passages usually requires more resources than tokenizing question texts in a retrieval system. We may also relax the third assumption based on our estimation of cost and benefit, which is discussed in Chapter 5.

We develop the solution to this problem based on hierarchical Bayesian modeling, where both prior knowledge and previous execution history are incorporated into the same framework. The basic idea is that we associate each cost \( c(\omega, x) \) or benefit \( b(\omega, x) \) with an unknown distribution, with parameters representing our knowledge and past observations. Starting with some known priors, we select the system to execute based on the greedy algorithm for the stochastic knapsack problem, and then update the priors each time we finish an execution. The process repeats until the budget \( C \) is reached. We first discuss how to estimate the cost and benefit of each component to solve the stochastic scheduling problem in Sections 4.1.2 and 4.1.3, and show an intuitive example in Section 4.1.4.

4.1.2 Modeling Cost & Benefit Distributions

We apply a hierarchical Bayesian model to capture the component characteristics hierarchically inherited from global level and step level. Specifically, we let two hyperparameters \( \gamma \) and \( \beta \) denote the average cost and benefit of a configured component globally across all steps and inputs, and then for each configured component \( f_{i,c}^\omega \) in the \( i \)-th step, we introduce step-level parameters \( \gamma_i \) and \( \beta_i \) to capture the characteristics shared by the components in the \( i \)-th step, and component-level parameters \( \gamma_{i,c} \) and \( \beta_{i,c} \) to model the performance of \( f_{i,c}^\omega \) regardless of the specific input \( x \). Analogous to \( \gamma_i \) and \( \beta_i \), we introduce parameters \( \gamma_x \) and \( \beta_x \) to denote the average cost and benefit of

\[ \gamma \rightarrow \gamma^x \rightarrow \gamma \]

\[ \gamma_i \rightarrow \gamma_i,c \rightarrow \gamma \]

\[ \beta \rightarrow \beta^x \rightarrow \beta \]

\[ \beta_i \rightarrow \beta_{i,c} \rightarrow \beta \]

\[ m_i \]

\[ n \]

3 Due to the discrete configuration assumption, the same component with a different configuration is treated as a different component. One can easily extend the model to incorporate the dependency between the configured components and their base components.
all configured components for each input $x$, which indicates the difficulty of processing each input. Given all the hyperparameters, the random variable $c(f_{i,c}^{\omega}, x)$ or $b(f_{i,c}^{\omega}, x)$ is the outcome from a series of generative processes from $\gamma$ or $\beta$, which hence defines a joint probability distribution, shown in Figure 4.1.

Based on prior knowledge of expected cost and benefit, we can assign values for global hyperparameters $\gamma$ and $\beta$, and optionally for some step-level hyperparameters $\gamma_i$ and $\beta_i$. Subsequently, unspecified step-level hyperparameters and configured component level parameters $\gamma_{i,c}$ and $\beta_{i,c}$ can be estimated with maximum a posteriori method based on priors $\gamma$ and $\beta$, and all observations (corresponding to previous execution records). For configured components that have not been executed, $c(f_{i,c}^{\omega}, x)$ and $b(f_{i,c}^{\omega}, x)$ can be predicted from estimated parameters $\gamma_{i,c}$, $\gamma^x$ and $\beta_{i,c}$, $\beta^x$ with Bayesian inference method.

### 4.1.3 Optimization via Stochastic Scheduling

Once $c(f_{i,c}^{\omega}, x)$ and $b(f_{i,c}^{\omega}, x)$ are estimated for the configured components in the pool, we can rewrite the objective for the deterministic scheduling problem (Equation 2.2) to a stochastic variant (a.k.a stochastic knapsack problem [46, 49]), which assumes that job durations and/or rewards are random variables with known probability distributions.

Among the solutions (or scheduling policies) for the stochastic scheduling problem, adaptive policies have been studied extensively in the literature, which dynamically choose which systems to execute next based on previously executed systems. In general, they can achieve a better approximation ratio to the optimal solution [46], and can also be naturally integrated with the configuration space exploration problem, since the estimation of all parameters and distributions of $b(f_{i,c}^{\omega}, x)$ and $c(f_{i,c}^{\omega}, x)$ are dynamically updated as execution proceeds. A commonly adopted greedy policy for the (stochastic) knapsack problem is to sort the items by

- decreasing (expected) benefit,
- increasing (expected) cost,
- decreasing (expected) benefit density, or
- $b(f_{i,c}^{\omega}, x)/\text{LCC}(f_{i,c}^{\omega}, x)$

in the context of analytics meta learning problem. Finally, based on the benefit evaluated for each system $f_{i,c}^{\omega}$ and input $x$, we select the systems by further measuring their generalizability to new data not yet realized, which can be estimated by various model selection methods, e.g. cross-validation, bootstrap, etc.

We present the solution in Algorithm 1 where $\langle P \rangle$ represents the system subspace spanned by the component pool $P$, $h(f_{i,c}^{\omega}, x)$ is a heuristic function defined on $f_{i,c}^{\omega}$ and $x$, to prioritize the components. Examples of $h$ function can be

- cost relevant ($-\mathbb{E}_c[\text{LCC}(f_{i,c}^{\omega}, x)]$),
- benefit relevant ($\mathbb{E}_b[\text{LCC}(f_{i,c}^{\omega}, x)]$),
- benefit density ($\mathbb{E}_b[b(f_{i,c}^{\omega}, x)]/\mathbb{E}_c[\text{LCC}(f_{i,c}^{\omega}, x)]$), or
- profit ($\mathbb{E}_b[b(f_{i,c}^{\omega}, x)] - \lambda \mathbb{E}_c[\text{LCC}(f_{i,c}^{\omega}, x)]$), etc.
Algorithm 1: Greedy algorithm

| input: Hyperparameters $\gamma$, $\beta$ and capacity $C$ |
| output: Optimal system $s^*$ |

1. $X' \leftarrow \emptyset$, $P \leftarrow \emptyset$, $Q \leftarrow \{(f_i^\omega, x)\}_{i,c}$;
2. while $c(P, X') < C$ do
3.   foreach $(f_i^\omega, x) \in Q$ do predict $b(f_i^\omega, x)$, $c(f_i^\omega, x)$, $\text{LCC}(f_i^\omega, x)$;
4.   $(f_i^\omega, x') = \arg \max_{(f_i^\omega, x)\in Q} h(f_i^\omega, x)$;
5.   $Q \leftarrow Q \setminus \{(f_i^\omega, x')\}$, $X' \leftarrow X' \cup \{x'\}$, $P[t] \leftarrow P[t] \cup \{(f_i^\omega, x')\};$
6.   foreach $x \in X'$ and $s \in (P)$ do if $s$ has not been executed for $x$ then execute to instantiate $c(s, x)$ and $b(s, x)$;
7.   foreach $t$ and $f_i^\omega \in A$ do update $\gamma_i,c, \gamma^x_i,c, \beta_i,c, \beta_i$;
8. $s^* = \arg \min_{s \in A} E_X[\max\{b(s, x)\};$
9. END

Table 4.1: Posterior distributions for each of $\gamma^c_i, \gamma_i, \beta^c_i, \beta_i$ in modeling analytics space exploration.

<table>
<thead>
<tr>
<th>$\theta$</th>
<th>$\phi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\gamma_i,c$</td>
<td>$n_{i,c} \bar{c}<em>{i,c}/\tau</em>{i,c}^2 + \gamma/(\tau_i^2 + \tau^2)$</td>
</tr>
<tr>
<td>$\gamma_i$</td>
<td>$\sum_c n_{i,c} \bar{c}_{i,c}/(\tau_i^2 + \tau^2) + \gamma/\tau^2$</td>
</tr>
<tr>
<td>$\beta_{i,c}$</td>
<td>$n_{i,c} \bar{b}<em>{i,c}/\sigma</em>{i,c}^2 + \beta/(\sigma_i^2 + \sigma^2)$</td>
</tr>
<tr>
<td>$\beta_i$</td>
<td>$\sum_c n_{i,c} \bar{b}_{i,c}/(\sigma_i^2 + \sigma^2) + \beta/\sigma^2$</td>
</tr>
</tbody>
</table>

Algorithm 1 applies an adaptive policy to execute the trace (lines 6–7), predict the random variables (line 3), and reestimate parameters (line 8) inside the loop (lines 2–8). This framework assumes no prior beliefs, nor probability distributions of component characteristics, which can be customized according to specific tasks. One can also consider to change the strategy function $h$ over time, e.g. promoting configured components with greater benefit variance or greater cost once the exploration is stuck in a local minimum. In Chapter 5 we compare the exploration process using a fixed exploration rate with that using a varied exploration rate.

4.1.4 An Example: 3-Stage Linear Characteristics

To motivate our solution, we assume that the component characteristics follow a simple and intuitive 3-stage linear model [115], where each observation or hyperparameter follows a Gaussian distribution, with mean drawn from a distribution parameterized by the upper level in the hierarchy and variance predefined and fixed. In the simplified 3-stage linear model, we assume each input $x$ to each configured component $f_{i,c}^\omega$ follows uniform distribution over all possible inputs. The generative processes for $c(f_{i,c}^\omega, x)$ and $b(f_{i,c}^\omega, x)$ can therefore be instantiated from the general solution.
framework (Figure 4.1) and formulated as follows:

\[
c(f^\omega_{i,c}, x) \overset{\text{iid}}{\sim} \mathcal{N}(\gamma_{i,c}, \tau_{i,c}^2)
\]

\[
\gamma_{i,c} \overset{\text{iid}}{\sim} \mathcal{N}(\gamma_i, \tau_i^2)
\]

\[
\gamma_i \overset{\text{iid}}{\sim} \mathcal{N}(\gamma, \tau^2)
\]

\[
b(f^\omega_{i,c}, x) \overset{\text{iid}}{\sim} \mathcal{N}(\beta_{i,c}, \sigma_{i,c}^2)
\]

\[
\beta_{i,c} \overset{\text{iid}}{\sim} \mathcal{N}(\beta_i, \sigma_i^2)
\]

\[
\beta_i \overset{\text{iid}}{\sim} \mathcal{N}(\beta, \sigma^2)
\]

We derive the posterior distributions for \(\gamma_{i,c}, \gamma_i\) and \(\beta_{i,c}, \beta_i\), which can be used to update the hyperparameters (line 8 in Algorithm 1). All the posterior probabilities follow Gaussian distribution of the form \(\mathcal{N}(\phi^{-1}\theta, \phi^{-1})\), and the values of \(\theta\) and \(\phi\) are listed in Table 4.1 for each parameter, where \(n_{i,c}\) represents the number of times \(f^\omega_{i,c}\) has been executed thus far, and \(\bar{c}_{i,c}\) and \(\bar{b}_{i,c}\) are the average cost and benefit of the component in the history. We see that at the beginning of the experiment, when \(n_{i,c} = 0\) for all \(i\) and \(c\), each parameter takes the same value from the user specified hyperparameter (\(\gamma\) or \(\beta\)), and as the experiment proceeds, each parameter is dynamically shifted away from the value of hyperparameter to better model the execution behavior of the step or component (\(\bar{c}_{i,c}\) and \(\bar{b}_{i,c}\)).

Finally, we define function \(h\) (line 5 in Algorithm 1) as the profit, i.e.

\[
E_b[b(f^\omega_{i,c}, x)] - \lambda E_c[LCC(f^\omega_{i,c}, x)]
\]

for the preliminary experiments. Equivalently, we can dynamically prune the components of either high cost or low benefit. In Section 4.2 we illustrate how the model was implemented in the framework to tackle a real-world analytics task.

### 4.2 Case Study: TREC Genomics Passage Retrieval

In this section, we implement the proposed exploration strategy and apply the configuration space exploration framework to the problem of building an effective biomedical document and passage extraction system from available components and component options. The goal of this work is to demonstrate the effectiveness of the proposed approach and the current open source implementation. Specifically, we employed the topic set and benchmarks from the question answering task of TREC Genomics Track [79], as well as commonly-used tools, resources, and algorithms cited by participants. A set of basic components was selected and adapted by writing wrapper code where necessary. Then an experiment configuration descriptor was defined for the resulting set of configured components. This configuration space was explored with the proposed strategy automatically, yielding an optimal and generalizable configuration which outperformed published results of the given components for the same task.
4.2.1 Task Description

The TREC Genomics QA task involves retrieval of short passages that specifically address an information need expressed as a question, and the answers are provided along with a reference to the location of the answer in the original source document. A total of 64 genomics-related topics were developed for this task, asking about biological processes or relationships between biomedical entities. We employed both the document collection and the topic set from the official evaluation and focused on two task-specific metrics: DocMAP and PsgMAP. Intuitively, DocMAP measures the relevance of the documents retrieved by the system, regardless of the relevance and conciseness of the passages extracted. The PsgMAP metric also considers the relevance of extracted passage spans. Details can be found in the overview paper [79].

To evaluate the generality of the framework in selecting optimal systems for novel data, and further test the performance of the selected systems, we implemented nested K-fold cross-validation (leave-one-out, 5-fold and 10-fold) and nested bootstrap methods, where two subsets of 28 topics used by TREC Genomics 2006 were held out for system selection (validation set) and performance evaluation (test set) respectively. Furthermore, we employed 36 topics from TREC Genomics 2007 to test the adaptability of systems for a similar but slightly different task, with questions asking for lists of specific biomedical entities (in Section 4.2.6).

4.2.2 Component Construction: Resource & Algorithm Integration

We mainly considered four different aspects when collecting resources and implementing algorithms to build a biomedical passage retrieval system: NLP tools, knowledge bases (or KBs), retrieval tools, and reranking algorithms. We considered popular and successful approaches reported in the TREC Genomics literature to explore. We summarize these components in Table 4.2, and briefly describe them in the rest of this subsection.

First, many successful systems employed natural language processing of the questions and/or target texts, using algorithms, toolkits, and pre-trained models for sentence segmentation, tokenization, part-of-speech tagging, named entity recognition, etc. To establish a benchmark for NLP in our system, we focused on several rule-based approaches for generating lexical variants, and supervised learning methods provided by LingPipe.

Second, tens of biomedical KBs have been organized and maintained. Some involve knowledge from all areas of biomedical research, while others focus on only a few subareas, e.g., disease, genomics, etc. The implication is that for a particular topic, the quality of the answer generated by the system may vary by the KB(s) that are used. We focused on three resources most popular with TREC Genomics participants: UMLS, EntrezGene, and MeSH.

Next, to retrieve relevant documents from the unstructured biomedical corpus, we need to rely on a widely-used open-source search engine, e.g. Indri. Finding an optimal retrieval configuration requires selecting parameter values specific to each search engine, such as the retrieval model and the parameters in the scoring function, smoothing method, query formulation strategy, etc.

Finally, a number of participants performed postprocessing on their passage output. This was done to refine the boundaries and ranks of retrieved passages, using techniques such as evidence
targeting, answer merging and rescoring, etc. The aspect that categorizes these approaches is reranking algorithms.

We followed the basic pipeline steps for a question answering system [193] and implemented a domain-independent QA framework. We integrated benchmarks, task-specific evaluation methods, as well as 12 components, and specified one to several values for each parameter associated with each component, and plugged them into the framework. To support straightforward reproducibility of results, the frameworks, components, and support materials are available online as part of our open source release.

### 4.2.3 Experimental Results

We designed two experiments, given moderate and large-scale configurations respectively. In the first experiment, we limited the number of options and consequently derived 32 configured components (A), which could produce a maximum number of 2,700 systems and require 190,680 individual execution steps to process all the questions. An experiment of this complexity was expected to execute within a day on our available test hardware (corresponding to C’ with resource defined by execution time). The mean and standard deviation for global execution time were initially set as 10 seconds (γ and τ) and the expected benefit for a single configured component was set as 0.1 in terms of PsgMAP (β and σ), and the parameters at step and component levels were estimated by an empirical Bayesian method. In the second experiment, we planned to test the scalability of the implementation by aggressively setting up to six values for each parameter, and thus yielded an analytics meta learning problem with 2,946 configurations and \(1.426 \times 10^{12}\) systems, requiring \(6.050 \times 10^{13}\) executions in total to evaluate the entire space.

We compare the settings for the two experiments with the official TREC 2006 Genomics test
<table>
<thead>
<tr>
<th></th>
<th>Participants</th>
<th>CSE</th>
<th>Scaled CSE</th>
</tr>
</thead>
<tbody>
<tr>
<td># Component</td>
<td>~1,000</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td># Configuration</td>
<td>~1,000</td>
<td>32</td>
<td>2,946</td>
</tr>
<tr>
<td># Trace</td>
<td>92</td>
<td>2,700</td>
<td>~1.426×10^{12}</td>
</tr>
<tr>
<td># Execution</td>
<td>~1,000</td>
<td>190,680</td>
<td>~6.050×10^{13}</td>
</tr>
<tr>
<td>Capacity (hours)</td>
<td>N/A</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>DocMAP Max</td>
<td>.5439</td>
<td>.5648</td>
<td>.5072</td>
</tr>
<tr>
<td>DocMAP Median</td>
<td>.3083</td>
<td>.4770</td>
<td>.3509</td>
</tr>
<tr>
<td>DocMAP Min</td>
<td>.0198</td>
<td>.1087</td>
<td>.2679</td>
</tr>
<tr>
<td>PsgMAP Max</td>
<td>.1486^{a}</td>
<td>.1773</td>
<td>.1181</td>
</tr>
<tr>
<td>PsgMAP Median</td>
<td>.0345</td>
<td>.1603</td>
<td>.0713</td>
</tr>
<tr>
<td>PsgMAP Min</td>
<td>.0007</td>
<td>.0311</td>
<td>.0164</td>
</tr>
</tbody>
</table>

^{a}0.174 was reported after the official evaluation [226].

results for the participating systems in Table 4.3. Although many more configured components were implemented by the original participants, only 92 different systems were evaluated. We estimate the number of components, configurations and executions evaluated by the official test, and show the evaluation results [80] in Table 4.3 for reference. The results reported from the two experiment settings were evaluated with the nested leave-one-out cross-validation method. Different strategies of estimating generalizability of systems are further compared in Section 4.2.6.

From Table 4.3, we can see that the best system derived automatically by the proposed approach outperformed the best participating system in terms of both DocMAP and PsgMAP, with fewer, more basic components. Similarly constrained exploration of the much larger Scaled CSE configuration failed to yield a system that performs as well in comparable time, highlighting the importance of a priori human selection of appropriate configurations and/or hyperparameter values. Nevertheless, the configuration yielded by exploring the larger space at the same capacity cost still outperformed most of the participating systems reported in the TREC Genomics track paper. The detailed pipeline configuration of the best system derived by the exploration strategy is described in Section 4.2.4 as the baseline to analyze component contributions. The goal of Chapter 5 is to further improve the performance of exploring a giant configuration space like (up to decillions of combinations, much greater than Scaled CSE) with a limited resource (up to 2000 iterations, much fewer than CSE).

### 4.2.4 Component Analysis

As the best system has been discovered, we report the performance of systems with only a single component or configuration different from the best system, in terms of PsgMAP, to provide a useful comparison. In this section, we investigate in detail how each component or configuration...
Table 4.4: Variation of nominal configurations for all components, including synonym and acronym expansion, lexical variants, sentence extraction, proximity based ranking, filtering of overlapping or identical passages, and score transformation. Significance levels of 0.1, 0.05, 0.01, and 0.005 are indicated by a sharp symbol (†) and one to three star symbols (*) respectively. Best baseline system: resolve acronym by UMLS; filter identical passages; use lexical variants; use proximity based ranking with synonyms; apply sentence extraction with synonym; resolve synonym by MeSH+EntrezGene; combiner scores transformed by reciprocal rank.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>DocMAP</th>
<th>PsgMAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best baseline system</td>
<td></td>
<td>.5257</td>
<td>.1744</td>
</tr>
<tr>
<td>Synonym expansion</td>
<td>None</td>
<td>.4169**</td>
<td>.1256*</td>
</tr>
<tr>
<td></td>
<td>UMLS</td>
<td>.4619†</td>
<td>.1439</td>
</tr>
<tr>
<td>Synonym expansion with KBs</td>
<td>MeSH</td>
<td>.5050</td>
<td>.1710</td>
</tr>
<tr>
<td></td>
<td>EntrezGene</td>
<td>.4302*</td>
<td>.1233*</td>
</tr>
<tr>
<td></td>
<td>UMLS+MeSH</td>
<td>.4942</td>
<td>.1609</td>
</tr>
<tr>
<td></td>
<td>UMLS+EntrezGene</td>
<td>.4639†</td>
<td>.1405</td>
</tr>
<tr>
<td></td>
<td>UMLS+MeSH+EntrezGene</td>
<td>.4959</td>
<td>.1603</td>
</tr>
<tr>
<td>Acronym expansion</td>
<td>None</td>
<td>.5253</td>
<td>.1751</td>
</tr>
<tr>
<td></td>
<td>MeSH</td>
<td>.5226†</td>
<td>.1753</td>
</tr>
<tr>
<td>Acronym expansion with KBs</td>
<td>EntrezGene</td>
<td>.5250</td>
<td>.1751</td>
</tr>
<tr>
<td></td>
<td>UMLS+MeSH</td>
<td>.5236</td>
<td>.1746</td>
</tr>
<tr>
<td></td>
<td>UMLS+EntrezGene</td>
<td>.5254</td>
<td>.1744</td>
</tr>
<tr>
<td></td>
<td>MeSH+EntrezGene</td>
<td>.5223*</td>
<td>.1752</td>
</tr>
<tr>
<td></td>
<td>UMLS+MeSH+EntrezGene</td>
<td>.5233†</td>
<td>.1746</td>
</tr>
<tr>
<td>Document and passage retrieval score combiner</td>
<td>Exponential of normalization</td>
<td>.5353</td>
<td>.1773</td>
</tr>
<tr>
<td></td>
<td>Exponential</td>
<td>.5362</td>
<td>.1759</td>
</tr>
<tr>
<td></td>
<td>Logarithmic</td>
<td>.5472</td>
<td>.1744</td>
</tr>
<tr>
<td></td>
<td>Normalization of exponential</td>
<td>.5471</td>
<td>.1730</td>
</tr>
<tr>
<td></td>
<td>Normalization</td>
<td>.5473</td>
<td>.1744</td>
</tr>
<tr>
<td></td>
<td>No transformation</td>
<td>.5471</td>
<td>.1730</td>
</tr>
<tr>
<td>Ablation of lexical variants</td>
<td></td>
<td>.4696</td>
<td>.1613</td>
</tr>
<tr>
<td>Ablation of sentence extraction</td>
<td></td>
<td>.4545***</td>
<td>.1259***</td>
</tr>
<tr>
<td>Ablation of proximity based ranking</td>
<td></td>
<td>.5444</td>
<td>.1693†</td>
</tr>
<tr>
<td>Filtering overlapping passages</td>
<td></td>
<td>.5342</td>
<td>.1773</td>
</tr>
</tbody>
</table>
tion contributes to overall system performance. The results are shown in Table 4.4 for nominal configurations, e.g. different KBs, score transformation methods, and Figure 4.2 for real-valued configurations, e.g. smoothing parameters, term weights, etc. We also report the significance test (t-test) results, and label the scores with different significance levels, in Table 4.4. The configuration of the best system is also shown in Table 4.4, which uses most available resources (KBs, reranking algorithms, NLP tools).

We see that the performance of leveraging various sources varied for synonym expansion and acronym expansion. Incorporation of MeSH in synonym expansion could make the biggest contribution to the performance, while both UMLS and EntrezGene also benefited the system’s performance. However, they also contained more noisy content than MeSH, which hurt the performance when combining with MeSH. As most acronyms were expanded as synonyms, integrating any combination of KBs hardly affected the performance, though we find that EntrezGene is a useful source for acronym expansion compared with others. Unsurprisingly, we confirm that synonyms should be considered in extracting important sentences and reranking, and filtering overlapping passages could better help the task than only filtering identical passages. Different transformation strategies hardly affected the overall performance.

Figure 4.2: Variation of real-valued configurations for all components in the TREC Genomics experiment. Y-axis corresponds to PsgMAP scores. (a) Weighting for concept terms (C), regular terms (R), gene names (G), and verbs (V). (b) Document relevance score weighting when combined with passage relevance scores (DW), Dirichlet (D) and Jelinek-Mercer (J) smoothing parameter. (c) Sentence extraction parameters: number of sentences in a passage (#S), number of passages in a paragraph (#P), importance and neighboring sentence thresholds (IT and NT). (d) Proximity-based ranking parameters (C1, C2, C3) and document score weighting when combined (CW).
For the real-valued parameters, we can see from Figure 4.2 that altering weights for concept terms or verbs could greatly change the performance. In particular, concept terms favored higher weights and verbs favored moderate weights (∼0.4). Jelinek-Mercer outperformed Dirichlet smoothing for this task with the best performance achieved when the parameters were set as 0.1 and 1000. We also found the performance was improved with the parameters set to lower values than those reported in their original paper for the important sentence extraction and reranking algorithms [190].

4.2.5 Exploration Strategy Performance Analysis

In this subsection, we demonstrate how the exploration strategy discovered the best system presented in Section 4.2.3 by analyzing the parameters estimated by the framework at different analysis steps.

We show in Figure 4.3 the likelihood and posterior distributions of cost estimated by the exploration strategy when the experiment completed. First, Figures (a) and (b) represent likelihood estimation of cost for five steps: score combination methods (C), overlapping passage solutions (O), retrieval strategists (R), sentence extraction and reranking (S), and term proximity based reranking (P) respectively. We can clearly see that the framework discovered each step had a different distribution of execution time, and C and O components usually performed faster than R, S, and P. Figures (c) to (d) represent likelihood estimation of component-level cost for these steps. We see that the configured components in each of C, O, and P had similar cost distributions, while the behavior of different S and R components varied, since the computational complexity of these components tended to be easily affected by different configurations. Figures (e) to (h) are the posterior estimations of $\gamma_i$ and $\gamma_{i,c}$ to predict $c(f_{i,c}, x)$ for unseen components. Since enough samples were provided at the end, the mean of each $\gamma_i$ or $\gamma_{i,c}$ was close to that of the likelihood estimation, and since more samples were available to estimate the step-level costs than the component-level costs, the variance of $\gamma_i$ tended to be smaller than that of $\gamma_{i,c}$.

We further investigate how the likelihood and posterior distributions of cost changed through the execution process by showing both distributions estimated when the experiment completed 1% of total execution tasks in Figure 4.4. Comparing with Figure 4.3, we see that the means of both distributions were smaller than those estimated at the end, due to the randomness of initially sampled components and input sets; the variances of the likelihood distributions were smaller due to fewer samples. According to Algorithm 1, components of downstream steps (C and O) tended to be sampled and tested more frequently than those of upstream steps (R, S, and P), due to smaller LCC scores; variances of the estimated posterior distributions of R, S, P components tended to be greater than those of C and O, which is clearly shown in Figures (e) to (h).

Estimating benefit had a similar process as estimating cost. We use various configurations to illustrate the estimated likelihood and posterior distributions of benefit when the 20% of total execution tasks were completed. Figures 4.5(a) and (b) show the likelihood estimation of the benefit for different configurations of the reranking algorithms and KBs, which exhibited different effects. Since KBs were integrated at the beginning at the pipeline while reranking algorithms were placed at the end, very few combinations of various KBs had been tested thus far, which correspond to the few “sharp” distributions, and the likelihood distributions corresponding to others remain un-
Figure 4.3: Estimation of likelihood and posterior of cost at the end of the TREC Genomics experiment. (a) and (b) represent likelihood estimation of step-level cost, where C, O, R, S, P correspond to score combination methods, overlapping passage solutions, retrieval strategists, sentence extraction and reranking, term proximity based reranking respectively. (c) and (d) represent likelihood estimation of cost at component level. (e) to (h) are the posterior estimations of cost mean $\gamma_i$ and $\gamma_{i,c}$. X-axis is the execution time in seconds.
Figure 4.4: Estimation of likelihood and posterior of cost when the TREC Genomics experiment completed 1% of total execution tasks. (a) to (h) correspond to Figure 4.3 (a) to (h). X-axis is the execution time in seconds in all subfigures.
known; on the other hand, different reranking methods were tested more equally frequently, as a result, their distributions were more alike. This assumption can also be justified from Figures (c) and (b), which correspond to the posterior estimations of both component-level benefit means, where variances of posterior distributions for reranking algorithms were close, while those corresponding to the untested KBs took the global variance.

### 4.2.6 Generalizability Analysis

In this subsection, we first demonstrate the performance of system selection results by various resampling methods. Besides the leave-one-out cross-validation, we also applied 5-fold cross-validation, 10-fold cross-validation (each repeated 100 times) and bootstrap (repeated 300 times) for model selection and performance evaluation. We find that the 10 highest ranked systems produced by all the resampling methods are identical; we further compare the performance of all systems evaluated on the validation set and test set in Figure 4.6(a). We can see that the systems produced by the proposed framework with a resampling based system selection method are generalizable to unseen data for the task, and cross-validation methods are less biased than bootstrap method.

We also tested these automatically selected configurations on 36 topics from TREC Genomics...
Figure 4.6: Generalizability of systems with various resampling methods on (a) holdout topic set from 2006 and (b) unseen topic set from 2007 in the TREC Genomics experiment. X- and Y-axis correspond to PsgMAP scores.

Table 4.5: Result on TREC Genomics 2007 data set

<table>
<thead>
<tr>
<th></th>
<th>DocMAP</th>
<th>PsgMAP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Max</td>
<td>Median</td>
</tr>
<tr>
<td>Participants</td>
<td>.3286</td>
<td>.1897</td>
</tr>
<tr>
<td>CSE</td>
<td>.3144</td>
<td>.2480</td>
</tr>
</tbody>
</table>

2007 to test their adaptability for a different but similar task. TREC Genomics 2007 topics are in the form of questions asking for lists of specific entities, e.g. “What [GENES] are genetically linked to alcoholism?”. For this example question, the expected output would be a ranked list of passages that relate one or more entities of type GENE to alcoholism.

We used the 2007 topic set to test the configurations selected by the exploration strategy for the 2006 data set (Table 4.5). We can see that the best system slightly outperformed the best reported in [81]. The difference in performance on the 2006 and 2007 data sets is plotted in Figure 4.6(b), which is clearly larger than the performance difference between different folds drawn from the 2006 data set. The best configuration is also different, and favors different KBs and weights. Nevertheless, the coefficient of correlation between 2006 and 2007 tests (0.272) still confirms that the best configurations from 2006 tend to perform better for 2007. We also see that data points are clearly clustered into groups, and the performance difference between clusters tends to be more significantly different than that within each cluster. This indicates that systems which statistically significantly differ from the top systems should be explored to better diversify the system outputs and enhance the likelihood of achieving optimal performance.
4.3 Conclusion

This chapter presents an exploration strategy based on hierarchical Bayesian model and stochastic scheduling. We report the experimental results on our preliminary study using the TREC Genomics QA task, which involves a configuration space (a subspace of an analytics space) consisting of trillions of possible execution processes.

In fact, the proposed algorithm is a special instance of the value-based policy optimization method. How does the value estimation method compare with other alternatives such as policy-based methods or model-based methods? How does the option selection method compare with others that maintain more efficient exploration? As most existing policy optimization methods are defined for the problems modeled by a Markov Decision Process (MDP), we can hardly utilize these methods for the configuration space exploration problem.

In Chapter 5, we formally define the exploration process as a Markov Decision Process (MDP), which allows to incorporate various policy optimization methods developed for multi-armed bandit and reinforcement learning problems for configuration space exploration and analytics meta learning problem, and improve the exploration efficiency over the strategy proposed in this chapter.
Chapter 5

Exploration via Policy Optimization for CSE-MDP

In this chapter, we systematically study the exploration strategies for the configuration space exploration problem. The problem setting is similar to but different from the multi-armed bandit problem and the reinforcement learning problem. We compare the definitions of these problems in terms of MDP, objective, reward, and cost in Table 5.1. The multi-armed bandit problem studies a one-step MDP model, and aims to maximize the total rewards (or minimize the regret) until the exploration converges. Budgeted multi-armed bandit problems and best arm identification problem also consider the budget constraint and variable or fixed cost of pulling an arm. On the contrary, the reinforcement learning problem deals with a multi-step (finite or infinite) MDP but usually assumes no budget constraint. The (budgeted) configuration/analytics space exploration problem requires policy optimization over a multi-step MDP with a budget constraint.

The differences in the definitions affect the selection of strategy modules for each respective problem, such as value estimation method, option selection method, and bootstrapping method. Table 5.2 on the other hand, compares the strategies.

**Value estimation** The traditional MAB problems study a single-state MDP, where each action (pulling an arm) is independent of previous actions, which suggests that value estimation is a relatively easy task for MAB problems. For example, one can simply use maximum likelihood method to estimate the expected reward of each action (i.e. arm). On the other hand, RL problems usually deal with multi-step MDPs, where the nature of sequential dependency between state-actions makes the value estimation a rather difficult task. Therefore, in addition to the tabular value-based methods (a straight-forward extension of the maximum likelihood method), RL researchers have also developed function approximation value-based methods (e.g. linear), policy-based methods, and also model-based methods. Since the CSE-MDP also defines multiple steps, we can leverage the value estimation methods introduced for RL problems to CSE problem.

**Option selection** Since traditional MAB problems, although do not necessarily have an explicit budget constraint, aim to maximize the total rewards throughout the entire exploration process (or equivalently minimize the total regrets before it converges), finding a better arm as
Table 5.1: Comparison of configuration space exploration problem with other related problems

<table>
<thead>
<tr>
<th>Problem</th>
<th>MDP</th>
<th>Objective</th>
<th>Reward</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi-armed bandit (MAB)</td>
<td>One-step</td>
<td>( \max \sum R )</td>
<td>Variable</td>
<td>Not considered</td>
</tr>
<tr>
<td>Multi-armed bandit with budget constraint and fixed cost (MAB-BF)</td>
<td>( \max \sum R ) s.t. ( \sum C &lt; C )</td>
<td>Fixed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multi-armed bandit with budget constraint and variable cost (MAB-BV)</td>
<td>( \max \sum R ) s.t. ( \sum C &lt; C )</td>
<td>Variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Best arm identification / pure exploration with budget constraint</td>
<td>( \max \mathbb{E}[R] ) s.t. ( \sum C &lt; C )</td>
<td>Fixed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reinforcement learning</td>
<td>Multi-step</td>
<td>( \max \mathbb{E}[R] )</td>
<td>Not considered</td>
<td></td>
</tr>
<tr>
<td>Configuration space exploration (CSE)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Budgeted configuration space exploration (BCSE)</td>
<td></td>
<td>( \max \mathbb{E}[R] ) s.t. ( \sum C &lt; C )</td>
<td>Variable</td>
<td></td>
</tr>
</tbody>
</table>

soon as possible is their priority. Their option selection methods, such as UCB [8, 104], interval estimation, GapE [64], and Thompson sampling, have the same general formula (often referred to as index) that combines the estimated value \( (\hat{q}) \) and the exploration bonus (or confidence \( b \)) of the value, i.e.

\[
a^* = \arg\max_a \ q_a + c(\cdot)b_a
\]  

(5.1)

but different methods use different ideas to estimate the confidence (e.g. standard deviation in interval estimation, and visit frequency in UCB, etc.) and different ways to scale the two terms (a constant scaling factor or a decayed scaling factor) with \( c(\cdot) \). Usually, these methods have a good balance between exploration and exploitation, and behave more aggressively at earlier exploration stage than later. In contrast, RL research less focuses on the option selection methods and mostly use the simpler \( \epsilon \)-greedy method, which maintains the same tiny pace (with probability \( \epsilon \)) throughout the entire exploration to explore the action space, due to the following reasons. First, value estimation is difficult, even in the stationary environment, due to sequential dependency between the steps, which makes the value term (i.e. first term in Eq 5.1) often biased from the true value. As a result, it is unclear when and how to decrease the exploration bonus, i.e. \( c(\cdot) \), whereas \( \epsilon \)-greedy method has fewer parameters (a single constant \( \epsilon \)). Also, the RL problems usually do not have a budget, and thus, can apply a more conservative exploration method for more iterations (many times millions of iterations) until it finally converges. Since the (budgeted) CSE problem has a budget constraint, it prefers an option selection method more aggressive than \( \epsilon \)-greedy, yet still easy to configure and also steady enough to enable accurate value estimation.
<table>
<thead>
<tr>
<th>Problem</th>
<th>Value estimation</th>
<th>Option selection</th>
<th>Bootstrapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi-armed bandit and variants</td>
<td>maximum likelihood</td>
<td>random, UCB, -BV [50], -E [7], Successive Rejects [7], interval estimation, Thompson sampling, etc.</td>
<td>no</td>
</tr>
<tr>
<td>Reinforcement learning</td>
<td>value-based, policy-based, model-based, etc.</td>
<td>$\epsilon$-greedy</td>
<td>no (MC), yes (TD, e.g. Sarsa($\lambda$) or Q-learning)</td>
</tr>
<tr>
<td>(Budgeted) configuration space exploration</td>
<td>all of above and others.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bootstrapping**  Bootstrapping refers to the case when estimating a parameter value depends on other estimated values, rather than just actual returns, which is only necessary for value estimation for multi-step MDPs. If the RL problem deals with finite-horizon MDPs, the Monte Carlo (MC) method (no bootstrapping) can be used. Otherwise, the temporal difference methods, such as Sarsa($\lambda$) or its off-policy variant Q-learning [209], can be used. Researchers have proved that temporal difference methods can converge to global optimum while the Monte Carlo methods can only converge to local optimum. However, the temporal difference methods usually lead to slow convergence, making it inappropriate for the (budgeted) CSE problem.

In fact, the study in analytics space exploration problem can also benefit the research in optimization. For example, MAB researchers have rarely studied budgeted policy optimization algorithms for multi-step MDPs.

In Section 5.1, we review prior work related to MDP modeling and policy optimization. In Section 5.2, we go over the important definitions, including Markov decision process (MDP) and commonly used policy optimization methods. In Section 5.3, we model the configuration space exploration process as an MDP (referred to as the CSE-MDP), where we propose to present each intermediate processing snapshot as a state and a component as an action. In Section 5.4, we describe how to use the state-of-the-art value-based and policy-based methods for policy optimization, and introduce a number of extensions, including an off-policy learning method for nonstationary environments, and model-based methods, which provides a total of 62 strategies. In Section 5.5, we report the experimental results using simulated environments, where we first compare the learning performance of the 62 exploration strategies that combine various state representation methods, value estimation methods, and option selection methods, and then, we vary the simulated environment to observe the impact of different problem sets and configuration spaces on the performance of the exploration strategies. In Section 5.6, we study the budgeted configuration space exploration problem, and develop 40 budgeted exploration strategies. In Section 5.7, we conduct another set
of simulated experiments to compare the performance of the 40 budgeted exploration strategies for the budgeted configuration space exploration problem. In Section 5.8, we relax the pipeline assumption and study to generalize the configuration space exploration strategies to the analytics space exploration problem, which incorporates nonlinear decision processes. Finally, we conclude in Section 5.9.

5.1 Related Work

In this section, we review prior work related to modeling and learning the configuration space exploration problem from two angles. In Section 5.1.1, we focus on the general and efficient tool – the policy optimization methods for sequential decision making problem. In Section 5.1.2, we review the approaches that prior research has proposed to automate or assist system composition and optimization.

5.1.1 Policy Optimization

Policy optimization is a key aspect in sequential decision making problems, which provides a general recipe to select the best action given a description of the current state. Markov decision process (MDP) is often used to describe this process. A formal definition of MDP is given in Section 5.2. Two subproblems are involved in planning and reinforcement learning [188]: prediction (policy evaluation) and control (policy optimization). Traditionally, when an information system is static, rather than customized to inputs or intermediate objects, it can be defined as a deterministic policy and thus can be evaluated by a policy evaluation algorithm. The CSE problem can be seen as policy optimization.

Policy evaluation and optimization

Solutions to the policy evaluation problem include: iterative policy evaluation [19, 83] and Monte Carlo policy evaluation. When the MDP is known, iterative policy evaluation algorithm does not require actual experience. When the transition function \( T \) and/or reward function \( R \) of the MDP is unknown, or we don’t even know whether the dynamics is Markovian or not, online methods, such as Monte Carlo policy evaluation, can learn from actual experience under policy \( \pi \). The convergence of these methods is guaranteed, i.e. the estimated state-value function \( v(s) \) converges asymptotically to \( v_\pi(s) \), the true state-value function under the policy \( \pi \), as the experience continues, at least for the tabular case [45, 186]. However, the convergence rates of these methods are still open questions in the research of reinforcement learning [188]. For known MDPs, solutions to the control problem in planning include policy iteration and value iteration [161]. Policy iteration methods [19, 83] consist of a policy evaluation step and a policy improvement step at each iteration, and aim to estimate the expected state-value functions \( v(s) \) and/or action-value function \( q(s, a) \) of the real dynamics under certain policies. Value iteration methods directly find the optimal policy, i.e. \( \max_a q(s, a) \), such as Q-value iteration. For unknown MDPs, in the learning problem, when \( q_\pi \) is evaluated from experience, Monte Carlo policy iteration/control [133, 179].
is used. These control methods are guaranteed that $q(s, a)$ converges to $q_*(s, a)$ under certain conditions [166, 210].

Both value functions $v(s), q(s, a)$ and the policy function $\pi(s, a)$ can be parameterized [18]. In the case that the state $s \in S$ is represented as a feature vector $\phi(s)$, the state-value function is approximated by a linear combination of features parameterized by a weight vector $v$, i.e. $v_\pi(s) \approx v^\top \phi(s)$ [18, 35, 168]. Then, evaluating a policy $\pi$ requires update of $v$, using (semi-)gradient descent method, instead of $v_\pi$ [186, 188]. Linear Monte Carlo algorithm is guaranteed to converge to or near the global optima [45, 186]. In the case that the state $s \in S$ and action $a \in A$ pair is represented as a feature vector $\psi(s, a)$, the action-value function is approximated by a linear combination of features parameterized by a weight vector $w$, i.e. $q_\pi(s, a) \approx w^\top \psi(s, a)$. Then, finding the optimal policy requires update of $w$ instead of $q$ in each policy evaluation step. In the case that the policy $\pi(s, a)$ is a parameterized distribution [214]. A softmax policy uses a linear combination of state-action features $\psi(s, a)$ weighted by $\theta$, and for the continuous action spaces, a Gaussian policy assumes the actions are drawn from a Gaussian distribution with a state-dependent mean, which can be represented by a linear combination of state features $\phi(s)$ weighted by $\theta$. Then, finding the optimal policy requires update of $\theta$ directly instead of $\pi$ or $q$ as in the value-based policy optimization. REINFORCE is a policy gradient algorithm based on Monte Carlo method [214]. Actor-critic method parameterizes both the policy function and the value function [13].

The information systems that we deal with in this thesis, regardless of whether they contain decision processes (full analytics space) or only pipelines (configuration space), have multiple steps of fixed length, i.e. each processing path forms a complete episode, which complies with the requirement of the above-mentioned Monte Carlo methods. However, when an information system has loops, i.e. the system may come back to a previous processing step before it terminates, e.g. involving iterative training of an analysis component, we may further consider the temporal-difference methods, e.g. TD(0) [101, 186, 216], TD($\lambda$) [186], Sarsa [168, 187], Sarsa($\lambda$) [209], Q-learning [209], and expected Sarsa [188], for infinite-horizon or incomplete episodes. These methods use bootstrap, which updates the estimate of values based on other estimates, rather than the true returns from the complete episodes. Monte Carlo methods are usually more efficient than temporal difference methods. TD(0) and TD($\lambda$) algorithms are guaranteed to converge to or near the global optima [45, 186]. The asymptotic error of the TD method is no more than $1/(1-\gamma)$ times the smallest possible error given by the Monte Carlo method [200]. The linear Sarsa algorithm is guaranteed to converge to near the global optima [67].

Model-based methods are another category of policy optimization methods, which directly estimate both transition and rewards, i.e. $P(P, R|H)$, rather than value functions. In the context of configuration space exploration, a component’s value represents the expected overall system performance if we use this component in the final optimal system and follow the optimal policy to select the remaining components, whereas a component’s reward (or benefit) is the measure of its share or contribution to the final performance, which is independent of other components in the pipeline. If the transition is known and the rewards need to be estimated, then a linear regression method is often used. In this thesis, we use the term value estimation to represent not only the value-based methods that directly estimate values but also the model-based methods that predict
benefits.

**Improving efficiency**

With the budget constraint, planning efficiency is another major concern when applying these state-of-the-art policy optimization algorithms to the exploration of configuration space. Most on-policy learning methods make an assumption that the policy the agent uses to generate episodes and estimate the value of each state and action is the current optimal policy, which forbids the agent to aggressively explore the state/action space but only allows it to take a steady step at each time. As a result, although many on-policy learning methods can eventually converge to the optimal policy, they are not efficient if we have a budget to constrain the exploration cost. Off-policy learning methods, on the other hand, use two policies: target policy (usually denoted by $\pi(a|s)$) and behavior policy (denoted $\mu(a|s)$) for the same MDP. The former is learned about and that becomes the optimal policy, and the latter is more exploratory and is used to generate behavior, in order to explore all actions. Therefore, we may use off-policy learning methods to maintain a more efficient exploration.

Another strategy to improve planning efficiency is to prioritize which action to select first at certain state and/or which state to explore next. The former topic is usually studied in the context of multi-armed bandit problem. When the solution to control problem is value-based, i.e. involves updating $q(s,a)$, the policy $\pi$ can be improved with respect to the new $q(s,a)$. The state-of-the-art methods include $\epsilon$-greedy [209], Upper Confidence Bound (UCB) [8, 104], and probability matching, gap-based exploration (GapE) [64]. The key idea is to select the action that tends to have higher value and/or be less frequently explored with higher probability and select the action that has lower value and/or be more frequently explored with lower probability. The latter topic is referred to as backup ordering, studied in the context of reinforcement learning. While iteration policy evaluation and policy/value iteration algorithms use full backup for planning and learning, Monte Carlo, TD, and other methods use sample backup, where the next state and/or action to be backed up are randomly chosen (possibly from a uniform distribution). Backup ordering deals with prioritizing the state-action pairs according to their urgency, benefit, easiness, etc. By selecting the order in which small backups are done it is possible to greatly improve planning efficiency. The simplest heuristic is prioritized sweeping [139, 153]. A queue is maintained of every state-action pair whose estimated value would change, prioritized by the size of the change. When the top pair in the queue is backed up, the effect on each of its predecessor pairs is computed. RTDP [14] and LRTDP [24] are planning methods for stochastic shortest path problems. Forward focusing strategy [13, 153] is to focus on states according to how easily they can be reached from the states that are visited frequently under the current policy.

**Budget constraint**

Although a number of planning efficiency improvement solutions are proposed, reinforcement learning problem rarely explicitly considers a budget constraint, and its algorithm terminates only when it converges. A more realistic scenario is that we have to rely on some “best-to-date” policy the system produces before the algorithm converges, due to a budget on the exploration process and
the cost each execution incurs. This scenario is often modeled in a multi-armed bandit problem. Even in the standard multi-armed bandit (MAB) problem setting, pulling an arm incurs a fixed and identical cost of 1. In a budgeted multi-armed bandit problem [50, 124, 196, 197], pulling an arm may incur a pulling cost, and the agent’s exploration budget limits the number of times it can sample the arms. Most work studies this problem with an additional assumption that the costs are fixed and known, e.g. Tran-Thanh et al. [196], where they consider the MAB problem with an overall budget, i.e. both exploration and exploitation phases are budget-limited and constrained by a single budget. Then, the problem is translated into an NP-hard problem –\textit{unbounded knapsack problem}, which has a reward-cost ratio (density) ordered greedy algorithm. Ding et al. [50] study the problem of MAB with budget constraint and variable costs, where both rewards and costs need to be estimated while exploration. Theoretical bounds are given to the proposed greedy algorithm. Different from multi-armed bandit problem, which aims to maximize the expected total rewards during the entire exploration phase, best arm identification problem [7, 64, 95] aims to find the best arm given the budget constraint, which is more relevant to the CSE setting. Audibert et al. [7] extend the UCB strategy and propose the UCB-E strategy to encourage exploration and also propose an elimination algorithm, named Successive Rejects, for the best arm identification problem.

Another constraint that is often added to the multi-armed bandit or best arm identification problem is the confidence constraint, which defines an alternative stopping criterion – a fixed confidence about the quality of the returned arms. Both are practical constraints a user may consider for configuration space exploration. In fact, research has shown both constraints can in fact be unified in a single framework [64]. In this work, we focus on the budget constraint.

\textbf{Meta-learning}

Meta-learning [26, 195] is relevant to our problem of configuration space exploration, which leverages meta information of learning algorithms and transfers knowledge from one training process to benefit the efficiency of training on another problem domain. Similar to ours, Li and Malik [110] approach this problem from the reinforcement learning perspective and represent the optimization algorithm as a policy. It parameterizes the mean of the policy using a neural net and uses a guided policy search method to update the policy. In our work, we compare the simplest existing value-based and policy-based methods for policy update, and discuss the pros and cons of each algorithm when being applied to the configuration space exploration problem.

\subsection{5.1.2 Automatic System Composition & Optimization}

The general problem of system engineering is studied in the field of operations research. We can think of analytics meta learning as an application of system engineering and operations research in the field of intelligent information system. However, there is some uniqueness/difference in representation and optimization when it comes to analytics meta learning problem. For example, traditional optimization solution in operations research, such as linear programming, dynamic programming, network flow, etc., requires a comprehensive understanding of the operation mechanism, which is used to derive mathematical formulas to represent this process. This idea is similar
to the *model-based* method in reinforcement learning. In contrast, analytics meta learning *never* attempts to understand how a component or a pipeline works from the first principle, instead it treats it as a black box, due to the complexity and uncertainty of components (either manual or automatic, rule based or neural model based, etc.), i.e., we adopt a similar assumption in reinforcement learning and thus use *model-free* methods, and we discover that they also work well for analytics meta learning.

Information system development process undergoes a series of steps from composition (integration) of components to evaluation and optimization of the integrated system. Researchers and developers have made efforts to automate or assist each respective development step or proposed solutions for some specific types of information systems. As a result, they have studied a similar problem but published in different research areas, which makes us difficult to conduct a complete survey of all these prior works. In this subsection, we compare the CSE problem and our proposed solution with a few representative works that relate to ours.

Rao [163] has surveyed the workflow composition algorithms or AI planning algorithms to tackle the automatic service composition problem. Most automatic composition approaches only deal with *configured components*, i.e., users have to predetermine the values to the configuration parameters. Similarly, most composition approaches include an evaluation process to select the top composition strategies among those fulfilling the requirement, which also requires specification of additional non-functional attributes of each component such as users’ preference [163]. After all, the performance depends heavily on what values the user assigns to these parameters, which is, however, clueless, time-consuming, and subjective, and thus unreliable. When it comes to constructing an intelligent information system, we can, fortunately, capture a component’s utilities by its “intelligence” (*task performance*) and its resource consumption (*runtime performance*), both of which can be measured against a benchmark during task execution. A fully automated composition approach should configure these attributes itself. In Rao [163], an “intelligent layer” is introduced to incorporate learning strategies to measure and optimize the composition. However, neither the survey paper itself or any mentioned work has attempted to propose a systematic solution how to design and implement a general intelligent layer for arbitrary workflow composition.

Siegmund et al. [178], as another example, propose to learn a performance-influence model and apply a sampling based method to find the optimal configuration for a software system, whose configuration space contains dependencies or libraries, which may also have configuration parameters to be specified. Due to the nature of software systems, the lower-level dependencies and libraries are compiled together with or linked to the high-level software code, which results in a single indecomposable “executable”. The performance-influence model explicitly considers the correlation between the configuration options. In contrast, an intelligent information system contains a series of processing steps that are dynamically combined, which takes an output from a previous processing step and produces a new intermediate output each at a time. Therefore, we can model this process using a sequential decision making model, such as Markov decision process (MDP), which gives us two advantages over the problem and the feature vector representation, studied in Siegmund et al. [178]. First, we may utilize the intermediate data to customize the selection of next processing step. Also, we do not need to explicitly model the correlation between the configurations as features, which is handled by the transition probability in a sequential model.
A deep neural network \cite{175} is another example of complex system that usually combines multiple layers and substructures in order to achieve various processing tasks, which simplifies feature engineering but requires a careful architecture design. Similar to our work, Zoph and Le \cite{228} train a model description generation RNN with reinforcement learning to maximize the expected accuracy of the generated architectures on a validation set. Despite that deep neural networks have been widely used to accomplish various end-to-end information process tasks, the components that are allowed to integrate into the architecture are mostly limited to only neural structures such as recurrent networks, convolutional neural networks, etc. Our solution focuses on arbitrary information systems that contain heterogeneous black-box components. Also, we consider efficiency for system optimization and thus study a more practical setting – the budgeted configuration space exploration, which limits the total computational cost for the exploration process and expects higher convergence rate.

Although evaluation is involved in automatic service composition, it is more widely discussed in the context of general software development. Traditionally, version evaluation in an iterative development process \cite{107} is done by manual judgment, with each iteration, based on how well each integrated component satisfies the task goal, and planning is done by manual prioritization of changes that are most needed in the next iteration based on their priorities and schedule estimates. This process has been partially automated or assisted by test automation and continuous integration (CI) tools \cite{132, 184}, which have been widely adopted in most modern development processes such as extreme programming \cite{16}, where a set of predefined test cases are executed automatically after each code commit and the actual outcomes are compared with the output specification to check if the requirements are met after the change. However, most test automation frameworks can only serve as a sanity checker for intelligent information systems, since their performance on a given analytics task is usually binary “pass or fail”, rather than judged subjectively on a graded or continuous scale. The machine learning community has also attempted to make the runtime performance of individual analysis components easily compared between various implementations and versions and accessed by the developers, e.g. MLComp\cite{1}, ml-benchmarks\cite{2}, OpenML\cite{3}, and further integrate the automatic benchmarking system into an automated build process, such as Jenkins CI framework\cite{5}.

\section{5.2 Preliminaries}

In this section, we go over the important definitions including Markov decision process, policy, prediction, control, and summarize the most representative policy optimization methods proposed for the reinforcement learning problem and used in Sections \ref{section:5.4}, \ref{section:5.5}, and \ref{section:5.7}.

Markov decision process describes the interaction between an agent and the environment. The agent first makes an observation ($O$) of the environment, and takes an action ($A$) in response to the observation, and then receives an immediate reward ($R$) that indicates how well the agent is doing.

\begin{itemize}
  \item \url{http://mlcomp.org}
  \item \url{http://scikit-learn.org/ml-benchmarks}
  \item \url{http://www.openml.org}
  \item \url{https://jenkins-ci.org}
\end{itemize}
As this process continues, we can describe these interactions using a sequence of action, observation, and reward triples \( \langle A, O, R \rangle \) (with an initial observation at the beginning of the sequence), which is usually referred to as the history \( H_t = O_1, R_1, A_1, \ldots, A_{t-1}, O_t, R_t \). The only tasks for the system is to take the actions at each time-step \( t \). In order to make predictions of what may happen next, the agent may gather relevant information from the history to describe its current status, which is referred to as the state \( S_t = f(H_t) \). When all states are Markovian, i.e. the transition from one state to another does not depend on the states in the past, a sequential decision making problem is described by a **Markov decision process** (MDP) [20].

**Definition 14** (Markov decision process, MDP). A Markov decision process is a quadruple \( (S, A, T, R) \), where

- \( S \) is the finite set of states,
- \( A \) is the finite set of actions,
- \( T \) is the state transition probability matrix, \( T_{s,s'}^a = P(S_{t+1} = s'|S_t = s, A_t = a) \), and
- \( R \) is a reward function, \( R_{s}^a = \mathbb{E}[R_t|S_t = s, A_t = a] \).

A **finite-horizon MDP** is guaranteed to terminate after a fixed number of steps \( N \), and a **terminal reward** may be given at the end of the episode, whose amount only depends on the final state \( R_s = \mathbb{E}[R_N|S_N = s] \). On the contrary, an **infinite-horizon MDP** may terminate after any arbitrary number of steps, where a discount factor \( \gamma \in [0, 1] \) is often defined to determine the impact of future reward on the present. Many times, the states only keep minimal, sufficient information to help decide what the strategy the agent should take to maximize the total return. However, during the exploration process, before we realize which strategy is optimal, we may need to keep other information about the exploration/experiment process in each state as well, which is referred to as **information state** \( \tilde{s} \) [20]. A MDP is also defined on the information state space \( \tilde{S} \):

\[
\tilde{M} = (\tilde{S}, \tilde{A}, \tilde{T}, \tilde{R}, \gamma).
\]

The system may use a **policy** to select \( a \in A \) at each time-step and drive the dynamics of the Markov decision process.

**Definition 15** (Policy). A (stochastic) policy \( \pi \) is a distribution over actions given states, \( \pi(a|s) = P(A = a|S = s) \). A deterministic policy \( \pi \) can be simply expressed as \( a = \pi(s) \).

In order to maximize the long-term future reward, **return**, **state-value function**, and **action-value function** are defined, which respectively describe the actual total future reward of a single trajectory following the process and sample-independent “goodness” of a state and a state-action pair.

**Definition 16** (Return, value functions, optimal policy). For discounted infinite-horizon MDP, the return \( G \) is the total discounted reward from a certain time-step:

\[
G = \sum_{k=1}^{\infty} \gamma^{(k-1)} R_k.
\]

For finite-horizon MDP, the return \( G \) is the total reward

\[
G = \sum_{k=1}^{N} R_k.
\]
The state-value function \( v_\pi(s) \) is the expected return starting from state \( s \) and then following policy \( \pi \):
\[
v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s].
\]
The optimal state-value function \( v_*(s) \) is the maximum value function over all policies
\[
v_*(s) = \max_\pi v_\pi(s).
\]

The action-value function \( q_\pi(s, a) \) is the expected return starting from state \( s \), taking action \( a \), and then following policy \( \pi \):
\[
q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a].
\]
The optimal action-value function \( q_*(s, a) \) is the maximum value function over all policies
\[
q_*(s, a) = \max_\pi q_\pi(s, a).
\]
An optimal policy can be found by maximizing over \( q_*(s, a) \)
\[
\pi_*(a|s) = 1 \left( a = \arg\max_{a \in A} q_*(s, a) \right).
\]

Two subproblems are involved in planning and reinforcement learning [188]: prediction (policy evaluation) and control (policy optimization). Given a MDP \((S, A, T, R)\) and policy \( \pi \), the prediction problem aims to find the value function \( v_\pi \). Given a MDP \((S, A, T, R)\), the control problem aims to find the optimal value function \( v_* \) and optimal policy \( \pi_* \).

The methods for the control problem are categorized into model-free methods and model-based methods. The model-free methods make no assumption of the transition and/or reward functions of the MDP, whereas the model-based methods attempt to fully recover the underlying MDP. The model-free methods can further be classified as value-based methods and policy-based methods. Value-based methods directly estimate the value functions, i.e. expected total reward for each state and each state-action pair. Policy-based methods do not learn the value functions, and instead they directly learn a policy function to determine \( \pi(a|s) \), i.e. how well a configured component might perform over other components. Categorical names and feature vectors are two common representations of states and state-action pairs. For moderate-sized finite/discrete state and action space, values functions can be stored in tables with names as the keys, in tabular value-based methods. Both value-based methods and policy-based methods can take feature vectors. A linear representation of a value function is a special case of function approximation. Different representations lead to different objective functions and solutions. Value functions in tabular forms can be estimated using maximum likelihood approach using Monte Carlo sampling. If value functions are represented as linear combinations of feature values weighted by a weight vector, the weight can be learned using a linear regression method, which can be optimized via minimizing mean-squared error. Any (incremental) linear regression (e.g. stochastic gradient descent) can be used to learn the weight. We summarize the most widely used methods in Figure 5.1 and give the update formula.
<table>
<thead>
<tr>
<th>Model-free</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value-based</td>
</tr>
<tr>
<td>Tabular</td>
</tr>
<tr>
<td>Function approximation</td>
</tr>
<tr>
<td>Linear</td>
</tr>
<tr>
<td>Policy-based</td>
</tr>
<tr>
<td>Softmax</td>
</tr>
<tr>
<td>Actor-critic</td>
</tr>
<tr>
<td>Model-based</td>
</tr>
</tbody>
</table>

Figure 5.1: Classification of policy optimization methods used in this chapter and their solutions.

**Monte Carlo (MC) policy evaluation.** It first initializes the counter \( n(s) \) and the mean return \( v(s) \) to 0. For each state \( s \) in each episode (following policy \( \pi \)) with actual return \( G \) (obtained at the end of the episode), it updates the counter and the mean return:

\[
\begin{align*}
  n(s) &\leftarrow n(s) + 1, \\
  v(s) &\leftarrow \alpha G,
\end{align*}
\]

(5.2)

where \( \alpha \) is the step-size parameter (learning rate), which can be a constant (referred to as constant-\( \alpha \) MC), or \( 1/n(s) \) if full history is used.

**Linear Monte Carlo policy evaluation.** If the state \( s \in S \) is represented as a feature vector \( \phi(s) \), the state-value function is approximated by a linear combination of features parameterized by a weight vector \( v \), i.e. \( v_\pi(s) \approx v^T \phi(s) \). It first initializes the weight \( v \). For each state \( s \) in each episode (following policy \( \pi \)) with actual return \( G \), it updates the \( v \):

\[
\Delta v = \alpha (G - v^T \phi(s)) \phi(s).
\]

(5.3)

**Tabular Monte Carlo control.** It first initializes the counter \( n(s,a) \) and the mean return \( q(s,a) \) to 0, and a policy \( \pi \). At each iteration, it first samples a complete episode using \( \pi \). For each state \( s \) and action \( a \) in the episode with actual return \( G \), it updates the counter and the mean return:

\[
\begin{align*}
  n(s,a) &\leftarrow n(s,a) + 1, \\
  q(s,a) &\leftarrow \alpha G,
\end{align*}
\]

(5.4)

The new policy \( \pi \) is derived from a policy exploration method, e.g. \( \epsilon \)-greedy (Eq. 5.9), with \( \epsilon = 1/k \) (for GLIE with guarantee of convergence).

**Linear Monte Carlo control.** If the state \( s \in S \) and action \( a \in A \) pair is represented as a feature vector \( \psi(s,a) \), the action-value function is approximated by a linear combination of features

---

5 We use \( x \leftarrow x + \alpha (y - x) \) to represent \( x \leftarrow x + \alpha (y - x) \).

6 We use \( \Delta x = y \) to represent \( x \leftarrow x + y \).
parameterized by a weight vector $w$, i.e. $q_\pi(s,a) \approx w^T \psi(s,a)$. It first initializes the weight $w$ arbitrarily. At each iteration, it first samples a complete episode using $\pi$. For each state $s$ and action $a$ in the episode with actual return $G$, it updates $w$:

$$\Delta w = \alpha (G - w^T \psi(s,a)) \psi(s,a). \quad (5.5)$$

The new policy $\pi$ is derived from a policy exploration method, e.g. $\epsilon$-greedy (Eq. [5.9]).

**Monte Carlo policy gradient (REINFORCE).** The policy $\pi(s,a)$ is a parameterized distribution.

$$\pi_\theta(s,a) \propto \exp \left( \theta^T \psi(s,a) \right). \quad (5.6)$$

It first initializes the weight $\theta$ arbitrarily. At each iteration, it first samples a complete episode using the policy distribution (e.g. softmax or Gaussian) parameterized by $\theta$. For each state $s$ and action $a$ in the episode with actual return $G$, it updates $\theta$.

$$\Delta \theta = \alpha \nabla \theta \log \pi_\theta(s,a) G. \quad (5.7)$$

**Linear Monte Carlo actor-critic.** It first initializes the weight $v$ and $\theta$. At each iteration, it first samples a complete episode using the policy distribution parameterized by $\theta$. For each state $s$ and action $a$ in the episode with actual return $G$, it updates $v$ and $\theta$.

$$\Delta v = \beta (G - v^T \phi(s)) \phi(s),$$

$$\Delta \theta = \alpha (G - v^T \phi(s)) \nabla \theta \log \pi_\theta(s,a). \quad (5.8)$$

For the standard multi-armed bandit and reinforcement learning problems, $\epsilon$-greedy and UCB1 are two most commonly used exploration (or option selection) strategies. When a budget constraint is given, UCB-E and Successive Rejects [7] are two state-of-the-art algorithms.

**$\epsilon$-greedy.** It randomly chooses non-optimal actions with a probability of $\epsilon$.

$$\pi(a|s) = \epsilon/m + (1 - \epsilon) 1 \left( a = \arg\max_{a' \in A} q_\pi(s,a') \right)$$

where $m$ is the number of actions with nonzero probability.

**UCB1.** It maximizes a specific Upper Confidence Bound and has a cumulative regret of order $log(n)$.

$$\pi(a|s) = \arg\max_{a \in A} c_{UCB} q(s,a) + \sqrt{\frac{2 \log \sum_{s,a} n(s,a)}{n(s,a)}} \quad (5.10)$$

where $n(s,a) = \sum_{t=1}^T 1[A_t = a]$ counts the number of times action $a$ is taken at state $s$.

**UCB-E.** It tries to explore much more and typically uses a parameter linear in $n$.

$$\pi(a|s) = \arg\max_{a \in A} c_{UCB} q(s,a) + \sqrt{\frac{\sum_{s,a} n(s,a)}{H_1(s)n(s,a)}} \quad (5.11)$$

where $H_1(s) = \sum_a \frac{1}{\Delta(s,a)^2}$ and $\Delta(s,a) = \max_b q(s,b) - q(s,a)$. 

85
Successive Rejects (SR). Given $K$ arms (candidate actions), it defines $K - 1$ phases. At phase $k$, it selects each arm $n_k - n_{k-1}$ times, where $n_k$ is defined as

$$n_k = \left[ \frac{1}{\log(K)} \frac{n - K}{K + 1 - k} \right]$$

for $k > 1$ and $n_0 = 0$. $\log(K) = \frac{1}{2} + \sum_{i=2}^{K} \frac{1}{i}$. At the end of each phase, the arm with the lowest estimated value, i.e. $\arg\min_{a \in A} q(s, a)$ is eliminated.

In both UCB formulas Eqs. 5.10 and 5.11 we add a scaling factor $c_{\text{UCB}}$ to the estimated value term and ignore the scaling factor $c$ in the original UCB-E formula [7]. In this chapter, we extend these baseline strategies to adapt to $N$-step system optimization problem.

5.3 Configuration Space Exploration Markov Decision Process (CSE-MDP)

In this section, we describe the dynamics of the configuration space evaluation process, and formally model the problem as a Markov decision process by defining the set of states, actions, transition and reward functions.

We make the same assumption as in [220] that each parameter of component can only take nominal values. In this work, we make another assumption that all components are independent, even if they configure the same component. We use $s$ to represent a (configured) component.

5.3.1 Configuration Evaluation MDP

Configuration evaluation problem is a subproblem of configuration space exploration, which focuses on a single information processing system $s$, i.e. a known policy. We use $s_i$ to denote the component at the $i$-th phase. In the simplest case, we evaluate the performance of this system using a single input $x$. Each component $s_i$ ($1 \leq i \leq N$) takes the output from previous phase and produces a new intermediate object, denoted as $x^{(i-1)}$ and $x^{(i)}$ respectively (the input to $s_1$, $x^{(0)} = x$).

For the configuration evaluation MDP, we may define $N + 1$ states, each corresponding to the intermediate object $x^{(i)}$ ($0 \leq i \leq N$), and $N$ actions, each corresponding to the component at step $i$: $\langle s_i \rangle$ ($1 \leq i \leq N$) which can be read as execute $s_i$. Since we have a single input and a single pipeline, at each state $\langle x^{(i)} \rangle$, we have no other choice than to take the only possible action: $\langle s_{i+1} \rangle$. Once it produces the new intermediate object $x^{(i+1)}$, it arrives deterministically at the new state $\langle x^{(i+1)} \rangle$ and receives an immediate reward, whose value depends deterministically on the component $s_{i+1}$ and the input $x^{(i)}$, or formally $r_{i+1}^{\text{im}}(x^{(i)})$. This process terminates after $N$ actions, and gives the final state a terminal reward $r_{\text{te}}(x^{(N)})$. The whole process can be described as a sequence of length $N$.

7Real-valued parameters should be discretized to a finite set of values. As a next step, this work may be extended to consider continuous MDP, which can incorporate real-valued parameters directly without discretization.

8In this chapter, we use $\langle \cdot \rangle$ to represent a state, $\langle \cdot \rangle$ to represent an action, and $T$ and $R$ for transition and reward functions.
Table 5.3: Markov decision process modeling for configuration evaluation and configuration space exploration. We compare the components, inputs, intermediate objects, state and action definitions, policy space $M / \Pi$, transition $T$, reward and terminal reward functions $R$.

### Configuration evaluation

<table>
<thead>
<tr>
<th>Intermediate objects</th>
<th>$x^{(i)}$</th>
<th>$(0 \leq i \leq N)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>$\langle x^{(i)} \rangle$</td>
<td>$(0 \leq i \leq N)$</td>
</tr>
<tr>
<td>Action</td>
<td>$\langle s_i \rangle$</td>
<td>$(1 \leq i \leq N)$</td>
</tr>
</tbody>
</table>

$M(\langle x^{(i)} \rangle) = \{ \langle s_{i+1} \rangle \}$  
$T(\langle x^{(i)} \rangle, \langle s_{i+1} \rangle) = \langle x^{(i+1)} \rangle$  
$R(\langle x^{(i)} \rangle, \langle s_{i+1} \rangle) = r_{i+1}^{im}(x^{(i)})$  
$R(\langle x^{(N)} \rangle) = r_{e}^{le}(x^{(N)})$

### Configuration space exploration

<table>
<thead>
<tr>
<th>Components</th>
<th>$s_{i,j}$</th>
<th>$(1 \leq i \leq N, 1 \leq j \leq M_i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs</td>
<td>$x$</td>
<td>(unknown random variable)</td>
</tr>
<tr>
<td>Intermediate objects</td>
<td>$x_{c_1:i}$</td>
<td>$(0 \leq i \leq N, \forall k: 1 \leq c_k \leq M_k)$</td>
</tr>
<tr>
<td>State</td>
<td>$\langle x_{c_1:i} \rangle$</td>
<td>$(0 \leq i \leq N, \forall k: 1 \leq c_k \leq M_k)$</td>
</tr>
<tr>
<td>Action</td>
<td>$\langle s_{i,j} \rangle$</td>
<td>$(1 \leq i \leq N, 1 \leq j \leq M_i)$</td>
</tr>
</tbody>
</table>

$\Pi(\langle x_{c_1:i} \rangle) = \{ \langle s_{i+1,j} \rangle \mid 1 \leq j \leq M_{i+1} \}$  
$T(\langle x_{c_1:i} \rangle, \langle s_{i+1,j} \rangle) = \langle x_{c_1:i} \rangle$  
$R(\langle x_{c_1:i} \rangle, \langle s_{i+1,j} \rangle) = r_{i+1,j}^{im}(x_{c_1:i})$  
$R(\langle x_{c_1:N} \rangle) = r_{c_1:N}^{le}(x_{c_1:N})$

### 5.3.2 Configuration Space Exploration MDP

The configuration space exploration problem makes the assumption that each processing phase can be instantiated with multiple components and configurations. We use $s_{i,j}$ to denote the $j$-th component instantiated for the $i$-th phase, and use $S$ to denote the entire configuration space: $S = \{ s_{i,j} \mid 1 \leq i \leq N, 1 \leq j \leq M_i \}$, where $M_i$ is the total number of such components integrated at the $i$-th phase. We define trace as the index sequence of components at their phases in the processing order starting from the first phase, which can be used to uniquely identify the processing paths. Specifically, the trace $(c_1, \ldots, c_l)$ (abbreviated as $c_{1:l}$) represents the processing sequence of components $s_{1:c_1}, \ldots, s_{i:c_i}$. Given an input $x$, the output from a sequence of components indexed by $c_{1:l}$ is denoted by $x_{c_{1:l}}$.

A configuration space $S$ defines $\prod_{i=1}^{N} M_i$ configurations (information systems), each of which takes an arbitrary input $x$ and uses a component $s_{i,j}$ at each phase $i$ to process the output from the previous phase $x_{c_{1:i-1}}$. Similar to the configuration evaluation MDP, this process (1) is guaranteed to terminate after a fixed number ($N$) of steps (also referred to as finite horizon), (2) never executes more than one component at each phase. These constraints limit what actions the agent can take at each step. Intuitively, since the MDP-MDP focuses on the single input $x$, we only need to consider the most recent intermediate object $x_{c_{1:k}}$ when deciding what action to take for the next step, i.e.
they define a subset of the policy space
\[ \Pi((x^{c_{1:k}})) = \{ \langle s_{k+1,j} \rangle \mid 1 \leq j \leq M_{k+1} \}. \]

We compare the configuration evaluation MDP and the configuration space exploration MDP in Table 5.3.

### 5.3.3 Persistenced Configuration Space Exploration MDP

A configuration space exploration framework can optionally utilize a persistence layer to persist intermediate objects, which gives a *persistenced configuration space exploration MDP* (PCSE-MDP) more flexibility and efficiency over the standard configuration space exploration MDP. First, the persistence layer can prevent repeating executions. If an input is sampled twice and processed by two different traces, CSE-MDP needs to perform exactly \( N - 1 \) executions, where \( N \) is the length of the pipeline, whereas if two traces share a finite-length common prefix, PCSE-MDP only executes the prefix once. Second, it also allows to switch the processing focus between multiple inputs and/or intermediate objects, by advancing the execution for some intermediate object and pausing the executions for all other inputs, before none of them reaches the terminal state (i.e. *infinite-horizon*). We see PCSE-MDP defines a much larger state space, action space, and policy state than the standard CSE-MDP.

A PCSE process can be defined as follows. The execution starts from the input \( x \). Once all the \( M_1 \) components at the first phase finish executing the input, we have \( M_1 \) possible outputs from the first phase: \( x^{(1)}, \ldots, x^{(M_1)} \), and then, we can use each of the \( M_2 \) components at the second phase to process each of the \( M_1 \) outputs from the first phase. As the execution continues, a tree (often referred to as *search tree*) is gradually grown from the root \( x \), whose nodes are intermediate objects \( x^{c_{1:i}} \) and edges connecting the \( i \)-th level and the \( (i + 1) \)-th level are components \( s_{i,j} \) (\( 1 \leq j \leq M_i \)). We use \( T^S \) to denote the final complete search tree, and \( T \preceq_x T^S \) to represent a *partial tree* of \( T^S \) co-rooted at node \( x \).

For PCSE-MDP, we define each state as a partial tree \( (T / x^{c_{1:k}}) (T \preceq_x T^S) \), where we explicitly specify the most recent intermediate object \( x^{c_{1:k}} \). At each state, it selects which intermediate object \( x^{c_{1:i}} \) from the partial tree \( T \) needs our attention and is then used as the input, and which component from the \( M_{i+1} \) options in the \( (i + 1) \)-th (next) phase to execute, i.e. \( \langle x^{c_{1:i}}, s_{i+1,j} \rangle \) (\( 0 \leq i \leq N - 1, 1 \leq j \leq M_{i+1} \)), which can be read as process \( x^{c_{1:i}} \) using \( s_{i+1,j} \). Once the output \( x^{c_{1:i,j}} \) is produced, the corresponding node is added to the partial tree, which defines the new state \( (T \cup \{ x^{c_{1:i,j}} \}) / x^{c_{1:i}} \). Again, we use \( r^{im}_{c_{1,j}}(x^{c_{1:i}}) \) to quantify the immediate reward from the transition, and \( r^{te}_{c_{1:N}}(x^{c_{1:N}}) \) for the terminal reward.

This representation can also support multiple inputs. We can slightly modify the definition to support multiple inputs. We introduce additional \( L \) “virtual” components \( s_{0,1}, \ldots, s_{0,L} \), each

\[ 9 \text{In the implementation of a partial tree, one can use a list of intermediate objects ordered by the recency of being added.} \]

\[ 10 \text{Alternatively and equivalently, we could define intermediate object selection and component selection as two separate actions } \langle x^{c_{1:i}} \rangle, \langle s_{i+1,j} \rangle, \text{ which however introduces additional transition states to describe the statuses after the intermediate object is selected but before the component is selected.} \]

88
representing selecting an input, and then integrate them at the 0-th phase. This extended search tree is denoted as $T^{S,X}$. The search tree in Figure 5.2 shows a snapshot of an example evaluation process using three inputs $x_1$, $x_2$, and $x_3$. The configuration space contains two phases and two components in each phase: $s_{1,1}$, $s_{1,2}$, $s_{2,1}$, and $s_{2,2}$. The gray circles represent the intermediate objects that have been generated from the previous executions, which describe the current state:

$$\langle x, x_1^{(1)}, x_2^{(1)}, x_2^{(1)}, x_2^{(2)} \rangle,$$

and the circles with dashed border represent the objects that do not exist but can be produced from a single-step execution of a component on an existing intermediate object, which define the set of actions

$$\langle x_1^{(1)}, s_{1,1} \rangle, \langle x_1^{(2)}, s_{2,1} \rangle, \langle x_2^{(1)}, s_{2,2} \rangle, \langle x_2^{(1)}, s_{2,2} \rangle, \langle x_2^{(2)}, s_{2,1} \rangle, \langle x_2^{(2)}, s_{2,2} \rangle, \langle x, s_{0,3} \rangle.$$

The PCSE-MDP is formally defined in Table 5.4. As a first attempt to model the configuration space exploration problem, we focus on the standard CSE-MDP representation.

### 5.3.4 State Representation

So far, we have represented a state as an intermediate object with meta information associated with it, e.g. $\langle x^{c_{1,i}} \rangle$. However, since there can be infinite number of initial inputs (e.g. various natural language questions or images awaiting recognition) and enormous amount of options of components and configurations, we need to deal with infinite number of states and actions, which makes the MDP models difficult to generalize to unseen inputs. In this section, we introduce
Table 5.4: Markov decision process modeling for persisted configuration space exploration.

<table>
<thead>
<tr>
<th>Components</th>
<th>Actions</th>
<th>States</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_{i,j}$</td>
<td>$x_l$</td>
<td>$\langle T/x_i^{c_{1:k}} \rangle$</td>
</tr>
<tr>
<td>$x_l^{c_{1:i}}$</td>
<td>$\langle x_l^{c_{1:i}}, s_{i+1,j} \rangle$</td>
<td>$\langle T \cup {x_l^{c_{1:j}}}/x_l^{c_{1:j}} \rangle$</td>
</tr>
<tr>
<td>$x_l^{c_{1:N}}$</td>
<td>$\langle x_l^{c_{1:N}} \rangle$</td>
<td>$r_{c_{1:N}}(x_l^{c_{1:N}})$</td>
</tr>
</tbody>
</table>

Persisted configuration space exploration

From a state $\langle x^{c_{1:i}} \rangle$ or $\langle T/x_i^{c_{1:k}} \rangle$, we are able to extract the meta information, including the current phase $i$, the current component $s_{i,c_i}$, and the partial trace that produces the intermediate object $c_{1:i}$, as well as the data being analyzed, including the current intermediate object $x^{c_{1:i}}$ and the initial input $x$. We may use any type of meta information or raw data, or any combination of them to simplify the state representation.

If we assume an i.i.d. distribution of the input $x$, and only consider the meta information, we have a fixed number of states, regardless of the input $x$, and therefore, we can use tabular techniques to calculate the value functions.

- Among them, the simplest representation – current phase (CP) information, i.e. $i$ – provides the minimum information in order to pick the action complying with the requirement of the atomic information processing task: any component integrated at phase $i+1$. The state-value function would become $v(\langle i \rangle)$ or $v_i$, which represents the mean expected return at the $i$-th phase, and the action-value function would become $q(\langle i \rangle, \langle s_{i+1,j} \rangle)$ or $q_{i+1,j}$, which simply assigns a global score to each component $s_{i+1,j}$.

- If we further consider the current component (CC), i.e. $s_{i,c_i}$, the state-value function would become $v(\langle s_{i,c_i} \rangle)$ or $v_{i,c_i}$, which is the mean expected return of a component, semantically similar to the action-value function using the current phase representation. The action-value function can be described as $q(\langle s_{i,c_i} \rangle, \langle s_{i+1,j} \rangle)$ or $q_{i,c_i},(i+1,j)$, which captures the pair-wise dependency between two components integrated at adjacent phases. Then, the selection of a component would partly depend on what component is used in the previous phase.

- The complete processing history of the current intermediate object can be fully recovered from the partial trace (PT) representation, i.e. $c_{1:i}$. The state-value function $v(\langle c_{1:i} \rangle)$ or $v_{c_{1:i}}$ would estimate the performance of a partial trace, and the action-value function $q(\langle c_{1:i} \rangle, \langle s_{i+1,j} \rangle)$ or $q_{c_{1:i}},s_{i+1,j}$ would suggest the next component based on the partial trace. One

11Our goal of this work is to find the optimal configuration within a finite configuration space. In an extended version of this problem, we may allow the test phase to reason about components that do not exist in the configuration space in the training phase. In this case, we also need to parameterize action representations.
disadvantage is that it may suffer from the sparsity issue as the number of phases and the number of options at each phase increase.

- If the user intends to define certain long-range dependencies, rather than just adjacent dependency, while avoid the sparsity issue of partial trace representation, we may also consider Bayesian network or influence diagram (ID, in the context of operations research), which decreases the state space first and thus also decreases the search space. For example, we can give each node in the influence diagram a name, which contains the dependency information of a state, and then the state representation can also include the **current ID node**, rather than the current phase.

We want to clarify that here, by (pair-wise) dependency between two components, we mean **active dependency**, i.e. the components’ performance at a later phase $P_2$ is a distribution parameterized by the components’ performance at an earlier phase $P_1$ or $P_2 \sim D(P_1)$. In contrast, the two random variables $P_1$ and $P_2$ may also have **passive dependency**. If the overall performance or return $G \sim D(P_1, P_2)$, which means $P_1$ and $P_2$ are conditionally dependent (another way of saying "not conditionally independent") given $G$, then although there is no correlation at all between $P_1$ and $P_2$, they are passively dependent in this case. The fact that if a configuration is promising, similar configurations would be favored in the exploration space, is a consequence of passive dependency. If we can further prove that the performance of components at each phase is independent with that of another phase, i.e. $P_1 \perp P_2$, then the globally optimal solution can be obtained from simply optimizing at each individual phase. The issue of local optimum may arise if there is active dependency (nonconvex) between phases.

To better select the next action (i.e. the component) at each state, we may also take the advantage of the data being analyzed at this state. Intuitively, we may expect some component should work the best for some types of inputs, but another component may work better for other types, which defines an **adaptive** policy. If we represent the data objects in their original forms, we have to make the assumption of independence, which does not allow generalization. Instead, we consider feature-based representation, or formally, we may define a feature function $\phi$ for an intermediate object or an initial input.

Although we can infer the phase of the current state solely from the intermediate object (e.g. recover $i$ from $x^{c1i}$) by analyzing what processes have already been applied to the data and what analysis steps remain to be done, it is usually challenging and unnecessary to do so, since we can simply combine the tabular representation for meta information with the feature-based representation for the content to represent a state. The first approach is to define each possible value that represents some meta information as an indicator feature, and expand the feature function $\phi$ with these additional dimensions ($N$ for current phase representation, $\sum_{i=1}^NM_i$ for current component representation, and $\sum_{k=1}^N\prod_{i=1}^kM_i$ for partial trace representation). Another approach is to define an individual data-feature function for each meta information-based state, e.g. we can have a $\phi_i$ for each phase if we combine with current phase representation, and a $\phi_{i,c_i}$ for each component if we combine with current component representation.

In this work, we focus on four state representations using meta information only: **current phase, current component, partial trace, and all of three** (ALL). An interesting future work is to study how to represent the input and the intermediate object as feature vectors.
5.3.5 Reward Measuring & Modeling

In the MDPs we defined in the earlier subsections, we use notations \( r_{i,j}^{\text{im}}(x^{e_{1:i-1}}) \) and \( r_{e_{1:N}}^{\text{te}}(x^{e_{1:N}}) \) to denote rewards received along each transition and at termination. In this subsection, we discuss how to measure rewards in the context of configuration space exploration, and how to model the reward functions.

First, we are able to measure the runtime performance, e.g. the execution time, the memory usage, I/O usage, etc., immediately after a component \( s_{i,j} \) has processed an input \( x^{e_{1:i-1}} \). We may use a single scalar value \( r_{i,j}^{\text{im}}(x^{e_{1:i-1}}) \) to summarize all the runtime performance measures of this atomic execution, which defines the immediate reward of an action in the MDP. However, we cannot measure the task performance, such as average precision in an information retrieval task or reciprocal rank in a factoid question answering task, until an \( N \)-step system \( s = (s_{1,c1}, \ldots, s_{N,cN}) \) has processed the input \( x \) and produced the final output \( x^{e_{1:N}} \), which defines the terminal reward \( r_{e_{1:N}}^{\text{te}}(x^{e_{1:N}}) \). We see that there is no need to measure or model the task performance at each phase and/or for each component. If the quality of the task accomplishment is measured by both the quality of the output and the system efficiency, we may also integrate runtime performance factors into the single terminal task performance measurement.

For both immediate reward \( r_{i,j}^{\text{im}} \) and terminal reward \( r_{e_{1:N}}^{\text{te}} \), we may make an assumption of uniform input, and thus the reward is sampled from a distribution parameterized by \( \xi_{i,j}^{\text{im}} \) or \( \xi_{e_{1:N}}^{\text{te}} \) only (e.g. the mean \( \gamma_{i,j}^{\text{im}} \) and the standard deviation \( \sigma_{e_{1:N}}^{\text{te}} \) for a Gaussian distribution). Formally,

\[
r_{i,j}^{\text{im}}(x^{e_{1:i-1}}) \sim \text{Dist}(\xi_{i,j}^{\text{im}}),
\]
\[
r_{e_{1:N}}^{\text{te}}(x^{e_{1:N}}) \sim \text{Dist}(\xi_{e_{1:N}}^{\text{te}}).
\]

The training step involves parameter estimation for \( \xi_{i,j}^{\text{im}} \) and \( \xi_{e_{1:N}}^{\text{te}} \). We can also leverage the technique of feature-based representation for the input to modeling rewards.

\[
r_{i,j}^{\text{im}}(x^{e_{1:i-1}}) \sim \text{Dist}(\phi(x^{e_{1:i-1}}); \xi_{i,j}^{\text{im}}),
\]
\[
r_{e_{1:N}}^{\text{te}}(x^{e_{1:N}}) \sim \text{Dist}(\phi(x^{e_{1:N}}); \xi_{e_{1:N}}^{\text{te}}).
\]

Here, we make no assumption that the evaluation metric for the task performance and/or execution environment, including CPU, network, and other resource availability, does not change throughout the whole evaluation / exploration process, which suggests that the reward functions are not just uncertain but also nonstationary even if the input and the components are fixed. For example, if a configuration space exploration process is performed on a share computing environment, the runtime performance measure may fluctuate depending on the availability of the resources. In another example, a common practice to improve analysis component efficiency is to introduce a caching layer into the analysis component, e.g. Lucene\textsuperscript{12} search engine caches recent frequent search results in memory, and a Web service wrapper may cache the responses from a remote server in a local database. If the task requires to consistently process unseen inputs, the runtime performance tends to remain at a low level. But if the task input set is finite or follows a Zipf’s law, the configuration space exploration process may see repeating inputs frequently, then the

\textsuperscript{12}http://lucene.apache.org/
assumption of nonstationarity allows to capture this phenomenon and quickly learn the “true” asymptotic performance.

5.4 Policy Optimization for CSE-MDP

In this section, we first describe how to apply the general value-based policy optimization methods in Section 5.4.1. Then, we introduce three alternative solutions in Sections 5.4.2 – 5.4.4 aiming to improve the optimization efficiency. In Section 5.4.2, we introduce an adaptive learning rate for off-policy learning in a nonstationary environment, and propose two-step option selection strategies. In Section 5.4.3, we further parameterize the option selection policy and apply policy-gradient methods. In Section 5.4.4, we propose an additive and a multiplicative benefit model to describe the configuration evaluation dynamics.

5.4.1 Value-Based Methods for CSE

Tabular Monte Carlo (MC) control (Eq. 5.4) and linear Monte Carlo control (Eq. 5.5) are simple yet effective value-based policy optimization algorithms.

Tabular MC control requires categorical state representations. We consider three representations: current phase (CP), current component (CC), and partial trace (PT) for tabular MC control. To utilize the linear MC control algorithm for policy optimization, categorical representation of each state is converted to an indicator vector. For example,

- **Categorical:** CURRENT_PHASE:2
- **Indicator vector:** CURRENT_PHASE=2:1

Using linear MC control with one-hot indicator vector representation is identical to using tabular MC control with categorical representation except for an optional intercept term in the linear combination. We don’t attempt to use the categorical representation of all the meta information (ALL), due to the sparsity issue, whose indicator vector representation however can be directly integrated within the linear MC control algorithm as well as other policy optimization algorithms that take parameterized state- and/or action-value functions. Both categorical and feature vector state representation can be extended to state-action representation, by replacing the name of the category or the feature by its concatenation with an action name. For example, if the agent has processed the input with components \( s_{1,1} \) and \( s_{2,5} \) during the first two phases and arrives at a new state, which can be represented as

\[
\begin{align*}
\text{CURRENT\_PHASE} &= 3:1, \\
\text{CURRENT\_COMPONENT} &= 2,5:1, \\
\text{PARTIAL\_TRACE} &= 1,5:1,
\end{align*}
\]

then the action of taking the component \( s_{3,4} \) can be conceptually represented as

\[
\begin{align*}
\text{CURRENT\_PHASE} &= 3\&\text{ACTION}=3,4:1, \\
\text{CURRENT\_COMPONENT} &= 2,5\&\text{ACTION}=3,4:1, \\
\text{PARTIAL\_TRACE} &= 1,5\&\text{ACTION}=3,4:1.
\end{align*}
\]
In the standard setting, we use $\epsilon$-greedy to explore the possible components at each phase in tabular MC control and linear MC control algorithms. We use the stochastic gradient descent method to learn the linear MC control. The learning rate $\alpha$ of both tabular MC control and linear MC control is defined depending on the nature of the environment, often as $\frac{1}{n}$ for stationary environment, or a constant value for nonstationary environment. In configuration space exploration MDP, the stationarity depends on not only the actual environment (task performance judgment and/or runtime execution environment), but also the state representation. For example, if we use a representation that contains more comprehensive history information, such as partial trace (PT), then the agent only visits the same state again if it repeats executing exactly the same components along this partial trace. In this case, the state senses the environment more stationary. If we use only the current phase (CP) to represent each state, the agent may visit the same state multiple times along different traces, resulting in possibly completely different returns. In this case, the state senses the environment nonstationary, which allows each state to quickly forget the old experience of episodes that tend to have more low-value states and actions, and learn the true value functions from the latest experiences from best-to-date systems.

We compare the incremental learning of linear MC control with its batch learning variant. The latter always learns the weight vectors using all the episodes in the history, which makes this method inefficient in practice. In the stationary case, this method serves as the upper bound of the incremental linear MC control. However, in the nonstationary case, using old episodes may be harmful. We use lasso, a $L_1$-regularized linear regression model, for the batch linear MC control.

### 5.4.2 Off-Policy Learning Methods for CSE

In this section, we discuss how to improve exploration efficiency of value-based policy optimization algorithms. The multi-armed bandit (MAB) problem aims to balance exploration and exploitation for an unknown MDP that contains only a single state, and thus pulling an exploratory arm does not affect the return of pulling a greedy arm. As a result, in the best arm identification problem [7], we can simply apply a maximal exploration strategy, such as random in Successive Rejects (SR) algorithm or UCB-E, to aggressively explore the arm space. This does not hold for a multi-step MDP, e.g. CSE-MDP. First, the total return reflects a combined effect of multiple actions in a multi-step MDP. Intuitively, to improve the learning efficiency, we need to identify the step or component that attributes the most to the gain or loss of the return. In the first part of this section, we propose a modification to the off-policy MC control method for nonstationary environments to solve this issue.

Most on-policy learning methods, such as tabular or linear MC policy control algorithms, make an assumption that the policy the agent uses to generate episodes and estimate the value of each state and action is the current optimal policy, which forbids the agent to aggressively explore the state/action space but only take steady steps each at a time, and results an efficiency problem. Off-policy learning methods use two policies: target policy (usually denoted by $\pi(a|s)$) and behavior policy (denoted $\mu(a|s)$) for the same MDP. The former is learned about and that becomes the optimal policy, and the latter is more exploratory and is used to generate behavior, in order to explore all actions.

Off-policy method aims to estimate the value functions for target policy given the returns
collected from the episodes following the behavior policy. **Weighted importance sampling** method can be used to adjust the learning rate for each state-value function or action-value function, proportionally to the ratio of the probabilities under the two policies

\[
\rho = \prod_k \frac{\pi(a_k|s_k)}{\mu(a_k|s_k)}
\]

and the **off-policy Monte Carlo policy evaluation** modifies the Eq. 5.2 to

\[
n(s) \leftarrow n(s) + \frac{1}{\mu(a|s)}
\]

\[
v(s) \leftarrow \rho^{n(s)} G
\]

(5.17)

This definition works for the stationary case, as the static counter \(n(s)\) won’t forget the old experience. To adapt the off-policy MC policy evaluation to nonstationary environments, we propose to replace the static counter \(n\) by an exponentially discounted counter \(n_{ED}\). It can be formally defined as

\[
n_{ED} \leftarrow \zeta n_{ED} + \frac{1}{\mu(a|s)}
\]

(5.18)

where \(\zeta\) is the **damping factor** \((0 < \zeta < 1)\). If the agent chooses to use the same behavior policy and target policy, then setting the damping factor to \(\zeta\) is effectively equivalent to setting the learning rate \(\alpha\) to \(1 - \zeta\) in the degenerated on-policy MC control algorithm. \(n_{ED}\) can be initialized in various ways. If \(n_{ED}\) is initialized as 0, then the first update of each state-value function uses a learning rate of 1, as in the stationary case. If \(n_{ED}\) is initialized greater than 0, then the first update of each state-value function uses a learning rate less than 1, which is more often considered for the nonstationary case. As we have no prior knowledge about the range of \(\mu(a|s)\) for each problem, it is difficult to specify a universal value to initialize the counter for every problem. Instead, we can use a relative value \(p\) that is multiplied to the first observed \(\mu(a|s)\) value to initialize the counter \(n_{ED}(s)\), i.e. \(n_{ED}^{(0)} \leftarrow p \mu(a|s)\), which is referred to as **initial padding length**. If we have a learning rate \(\alpha\) in mind, in addition to setting the damping factor to \(1 - \alpha\), we can also set \(p = \frac{1}{\alpha}\). This exponentially discounted counter can also be used in other MC-based policy optimization algorithm, e.g. linear MC control.

If we apply \(\epsilon\)-greedy strategy with a relatively small \(\epsilon\), to select option independently at each step, then this off-policy MC control is able to identify the step(s) or component(s) that attribute the most to the return and thus assign a higher learning rate than the rest of the components. However, this method still has a limitation when the exploration process is concurrently aggressive at all steps, e.g. if we set a large \(\epsilon\) in \(\epsilon\)-greedy or a small \(c_{UCB}\) in UCB (Eqs. 5.10 and 5.11), where all the components are assigned a small learning rate. In the second part of this section, we discuss possible exploration schedules to coordinate the greedy/exploratory action selections across all steps.

One solution is to follow a two-step (phase-component) option selection strategy, where it first selects which phase(s) to explore, and then selects which component to explore within the
selected phase(s), while keeps using the greedy components at all other phases. We can apply similar exploration strategies to phase selection as in component selection, e.g. $\epsilon$-greedy, softmax, UCB, etc. The simplest fully coordinated phase selection method is to randomly pick one phase to explore, which guarantees that at most one step is exploratory. At the phase to be explored, we can freely leverage any aggressive exploration strategy. Therefore, we can have a family of two-step option selection strategies, e.g. using random selection at the phase level and another random selection at the component level, denoted by random/random (RR), or using random selection at the phase level and a UCB1 selection at the component level, denoted by random/UCB1 (RU), etc.

Some exploration strategies, such as UCB, take the advantage of counts of state or state-action visits to calculate the index that reflects how often each action is selected for exploration. In fact, we can use a different state representation for exploration from that used for value estimation. For example, we may choose to use current phase (CP) as the state representation for value estimation to avoid sparsity issue, and choose to use partial trace (PT) as the state representation for exploration to reduce repeating episodes and maximize exploration. We use a suffix PT, e.g. in RU/PT, to denote a different exploration state representation in the policy optimization process.

5.4.3 Policy-Based Methods for CSE

Policy-based methods, e.g. REINFORCE (Eq. 5.7), directly estimate the tendency of choosing one action over other actions. One advantage of the policy-based methods over the value-based methods and the model-based methods is that, once trained, the learned policy is not relative to the task that trains it, which gives us two major practical benefits.

First, rapid domain adaptation becomes easier with policy-based methods. Specifically, if an old task defines a state $s$ and a set of valid actions $\{a\}$ for that state, and a policy has trained for each state-action pair: $\pi(a|s)$, then we can directly leverage the same policy $\pi(a|s)$ for a new task, as long as it reuses the state $s$ and the actions $\{a\}$. For example, a pipeline for document retrieval using natural language questions (a configuration evaluation MDP) defines a “question parsing” phase (i.e. “question parsing” state, if we use the current phase for state representation), with two parsers A and B integrated into the phase. Then, we use average precision, a task performance metric, as the reward signal to train the policy for this phase. When we build a pipeline for factoid question answering, we also define the phase “question parsing” and reuse the parsers A and B, and then we can reuse the policy previously trained for the document retrieval task to predict the performance of A and B, although the new task may use a different metric, e.g. reciprocal rank. In contrast, the state-value function $v(s)$ or the action-value function $q(s, a)$ learned from value-based methods, are prediction of original metric (e.g. average precision), and thus cannot be directly used for the new task.

Moreover, the user can easily specify his/her preference or prior knowledge about the performance in a policy for initialization, whereas the user must know the task performance metric as well as the performance of other components in the pipeline in order to correctly specify the state-value function or action-value function for value-based method. This can also help overcome the issue of sparsity and nonconvex (i.e. local optimum) with some configuration spaces, possibly due to active dependency between phases, as discussed in Section 5.3. For example, let us assume
that two components $c_{1,1}$ and $c_{1,2}$ are integrated into the first phase, and two other components $c_{2,1}$ and $c_{2,2}$ are integrated into the second phase. If $c_{1,1}$ and $c_{2,1}$ combined gives the best result, and $c_{1,2}$ and $c_{2,2}$ combined gives the second best result, and two other combinations have a zero reward, then once $c_{1,2}$ and $c_{2,2}$ are selected as the current optimal policy, then it would be hard for a conservative exploration method to discover the global optimum, since it with a much higher probability chooses to explore a single phase within an episode. To overcome this issue, the user may use high-level prior knowledge into the exploration process. If the user realizes that there is active dependency between adjacent phases, then he/she can use current component (CC) instead of current phase (CP) for state representation, as we discussed in Section 5.3. Before the exploration process starts, the user can initialize the value table to reflect such dependency. For the above example, we may initialize the weight as follows (we note that we use current component for the state representation):

<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
<th>Initial Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_{1,1}$</td>
<td>$c_{2,1}$</td>
<td>0.1</td>
</tr>
<tr>
<td>$c_{1,2}$</td>
<td>$c_{2,2}$</td>
<td>0.05</td>
</tr>
<tr>
<td>$c_{1,1}$</td>
<td>$c_{2,2}$</td>
<td>−0.1</td>
</tr>
<tr>
<td>$c_{1,2}$</td>
<td>$c_{2,1}$</td>
<td>−0.1</td>
</tr>
</tbody>
</table>

Then, the actual active dependency between the components can be learned by the policy gradient strategy.

We use softmax policy and the same set of state-action representations as in Section 5.4.1. The learning rate $\alpha$ can be set as a constant or a decayed value, e.g. the inverse of the state-action visit count. As it is well known that the policy-gradient method can easily converge to a local minimum, a decayed learning rate encourages alternative options to compete with the current optimal option. We also consider actor-critic methods (Eq. 5.8) [13] to use the action value predicted by a value-based method as the target to train the policy, and advantage actor-critic methods, which also consider the state-value function as a baseline. For action value and state value estimation, we use both tabular MC control and linear MC control.

### 5.4.4 Model-Based Methods for CSE

A CSE-MDP defines a fixed $N$-step process, unless a component encounters some error and must terminate early. Value-based methods learn the expected total reward of each state and action from experiences, but they cannot explain how this reward is credited from the first principle or predict the reward of a test episode that ends at a state never seen in the training process. Model-based methods aim to answer this problem. We assume each action generates a hidden reward (referred to as benefit $b$), which is a measure of its share or contribution to the final performance, independent of other components in the pipeline. Then, the total reward accumulates the component-level benefits.

We introduce two benefit accumulation models: additive model and its logarithmic variant multiplicative model.
Additive model. It assumes a zero reward should be credited to an empty pipeline, since it cannot solve the task. Each execution adds some new information, taking a step, of size equal to benefit (ideally $b \geq 0$), towards building the bridge from the input to the output to fulfill the task. The total reward measures how far we have gone after $N$ such steps, which is the sum of these benefits.

Similar to the value-based method, we define the benefit as the dot product of the state-action representation $\psi(s, a)$ and the global weight vector $w$, i.e. $b(s, a) = w^T \psi(s, a)$. The additive model defines the return is the sum of all benefits from the total of $N$ phases. The weight vector can be easily learned from minimizing mean-squared error, via stochastic gradient descent.

Multiplicative model. It assumes a full reward (1) should be credited to an empty pipeline, since the original information is completely preserved and no information has been lost. Each execution adds some new information, which introduces some error due to misinterpretation, and causes the problem representation to diverge from its original input. The benefit measures how much information is preserved at each execution (ideally $0 \leq b \leq 1$). The errors from all earlier steps cascade down to the $N$-th task, where the total reward is calculated by measuring how much original information is preserved by multiplying all the benefits accumulated in this episode.

We define the benefit as the exponential of dot product of the state-action representation $\psi(s, a)$ and the global weight vector $w$, i.e. $b(s, a) = \exp(w^T \psi(s, a))$, and thus the solution to the multiplicative model is simply the solution to the additive model after we take the logarithm of the return\(^\text{[13]}\).

Since a model-based method does not directly learn the expected total reward for each individual state-action pair, instead it can only predict the expected reward for a complete episode. In order to select option at each time step and report the best policy, it requires to estimate the expected reward of a state-action pair on the fly. We apply a general solution, similar to Monte Carlo tree search [28], where we first perform a certain number ($N$) of simulated episodes all starting from the current state and taking the action as the immediate next step, and then we use the average of the top $k$ returns as the expected total reward. To further reduce the variance of the estimation and improve efficiency, we introduce a damping factor $\gamma$ and calculate the expected reward by combining both the memorized reward calculated during the last visit and the newly estimated reward derived from the sampling based method described above, weighted by the damping factor $\gamma$ and $1 - \gamma$ respectively. If a small damping factor is used, the estimated action value is predominantly determined by the simulated episodes sampled from the learned MDP, which may have a large variance due to the randomness of the sampling process. If a large damping factor is used, the estimated action value mostly uses its old value, and therefore, erroneous estimated values, due to lack of training instances collected in the first several episodes at the beginning, have a longer-than-normal-term effect in misleading the selection of optimal options. We select a $\gamma$ that can balance both effects.

\(^{13}\)One can also define directly define the benefit as in the additive model, $b(s, a) = w^T \psi(s, a)$, and its SGD update rule $\Delta w = \alpha \text{error} \cdot \frac{\psi(s, a)}{w^T \psi(s, a)}$ can hardly converge.
The multiplicative model has a drawback that if the execution terminates early due to some error or exception, then it tends to overestimate the terminal reward. In the extreme case, the empty pipeline is always the optimal, since it guarantees to receive the full reward. Therefore, we explicitly model how likely an error occurs in the pipeline that may cause early termination. We make the assumption that the occurrence of an error can be consequence of a combined effect of multiple prior executions. For example, a question answering pipeline uses a simple token based abstract query generator, which creates a query term for every token in the question, including the question mark, and then it uses a bag-of-phrase query string constructor to create the query string, which concatenates the query terms into a single string, separated by a space. Finally, it sends this query string to the Lucene search engine, which throws a ParseException due to the unparsable question mark. We should “blame” the use of all these three steps, rather than any single step, because we can easily avoid this error by replacing any of these three steps: (1) using a concept based abstract query generator, which does not include the question mark, or (2) using a query string constructor tailored for Lucene, which can escape the special characters, or (3) send this query to a more robust search engine, e.g. a commercial search engine. Therefore, we should consider the entire partial episode when we make the prediction of an erroneous episode.

We formulate the problem of predicting whether a partial episode can cause error as a binary classification problem. Again, we derive the vector representation of a partial episode from concatenating the state-action vectors in the episode. We use logistic regression for the error model, and we integrate the error model with the additive benefit model or multiplicative benefit model. Specifically, in the training process, if an error occurs, we report the failure to the error model and retrain the logistic regression model using the additional “positive” example (with respect to the “error”). Otherwise, we report a success to the error model and retrain the logistic regression model using the additional “negative” example, and also report the task performance result to the benefit model and update it accordingly. For performance prediction in the option selection phase, we first invoke the error model to predict whether advancing the episode with an addition execution may cause an error. If the prediction is negative (i.e. no error may occur), we use the benefit model to predict its estimated action-value, otherwise, we directly assign the action-value as zero. We note that an action with an action-value of zero may still be selected and performed in an exploratory step.

5.5 Simulated Experiments for CSE

In this section, we report the experimental results using a simulated environment. We first describe the simulated environment, and the experiment settings. Then, we compare the learning performance of learning strategies, including value-based methods, off-policy methods, policy-based methods, and model-based methods. Finally, we compare the learning performance in different environments by varying the simulated environments.
Table 5.5: Description of simulated problem set and configuration space

<table>
<thead>
<tr>
<th>Problem set</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>distribution beta</td>
</tr>
<tr>
<td></td>
<td>mean: 0.5</td>
</tr>
<tr>
<td></td>
<td>variance: 0.01</td>
</tr>
<tr>
<td>Evaluation</td>
<td>multiplication of input and task performances of all processing steps</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Configuration space</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Length of the pipeline</td>
<td>10</td>
</tr>
<tr>
<td># of components at phase $i$</td>
<td>10, for $i = 1, \ldots, 10$</td>
</tr>
<tr>
<td>Task performance of component $c_{i,j}$</td>
<td>distribution beta</td>
</tr>
<tr>
<td></td>
<td>mean: $\frac{j}{10}$, for all $i$ and $j = 0, \ldots, 9$</td>
</tr>
<tr>
<td></td>
<td>variance: 0.01 for all $i, j$</td>
</tr>
</tbody>
</table>

### 5.5.1 Simulated Environment

The simulated environment consists of a simulated problem set and a simulated configuration space. The problem set contains infinite number of inputs, which are drawn independently from a beta distribution, with mean of 0.5 and variance 0.01 (i.e. $a = 12, b = 12$ in the standard beta distribution form). The mean simulates the difficulty of the task (the higher the mean, the easier the task), and the variance simulates the diversity of the task (the higher the variance, the more diverse the task). The task performance is calculated by multiplying the input value and the task performances of all processing steps. The configuration space contains a total of 100 configured components: 10 components at each of the 10 phases. The task performance of each component $c_{i,j}$, i.e. the $j$-th component at phase $i$, is drawn independently from another beta distribution, with mean of $\frac{j}{10}$ for $j = 1, \ldots, 9$ and fixed variance of 0.01. The higher the mean, the better the component performs, and the lower the variance, the more consistent the performance of the component is. We introduce a component that always throws an artificial `IntendedException` for each 0-th component $c_{i,0}$ for all $i$. The simulated environment is summarized in Table 5.5. We make an assumption here that the task performance is independent of each other. We may also consider dependent components, e.g. the task performance of $c_{i,j} = 1$ if and only if $c_{i',j'}$ is used and 0 otherwise, which we leave for future work.

The configuration space has a total of $10^{10}$ different combinations. In the simulated experiments, we observe the learning performance of the first 2000 episodes, i.e. only 0.00002% of all possible pipelines are tested if we apply a random sampling strategy (without replacement), which is much lower than the condition (100% for CSE) used in Yang et al. [220].
5.5.2 Experiment Settings

Due to the randomness of the strategies, in this section, we repeat each experiment setting 10 times, and report the mean, standard deviation, minimum, and maximum value of the 10 runs. We report two learning curves for each setting: the behavior performance curve, i.e. the actual total reward received at the end of each episode, and the expected optimal performance curve, which is calculated from dividing the expected performance of the optimal-to-date policy by the maximum expected performance, given that the optimal policy is known. The expected optimal performance is a normalized score between 0 and 1, which allows us to compare the performance across different environments. For example, the optimal policy for the simulated environment is to select \( c_{i,9} \) for all \( i \), which has an expected task performance of \( 0.9^{10} \), and a policy that selects \( c_{1,8} \) and \( c_{i,9} \) for all \( i \) except \( i \neq 1 \) has an expected optimal performance of \( \frac{0.8 \times 0.9^9}{0.9^{10}} \). The distinction between the behavior performance curve and the expected optimal performance curve is important, especially when we compare the off-policy learning methods.

Each data point in the learning curve is smoothened by a sliding-window of 1000 raw data points, from the past 100 consecutive episodes from 10 random runs. The standard error range is plotted as a colored area surrounding the curve. We use the area-under-the-curve (AUC) metric to summarize the learning curve into a single real-valued number, which enables us to further compare the performance of the strategies with different parameters (e.g. \( \epsilon \) in the \( \epsilon \)-greedy and the learning rate \( \alpha \)). We define AUC-BP@N and AUC-EOP@N for the behavior performance curve and the expected optimal performance curve respectively, terminating at the \( N \)-th episode, which is opposite of the regret. We use \( N = 2000 \). We also conduct significance test for each strategy and its comparand using t-test. We report the comparand and the significance level. “Three stars” (⋆⋆⋆) represents a significance level < 0.0001, “two stars” (⋆⋆) represents < 0.001, “one star” (⋆) represents < 0.01, “one dagger” (†) represents < 0.05, and “ns” otherwise.

5.5.3 Comparison of Policy Optimization Strategies

In this subsection, we compare the learning performance of value-based methods, off-policy methods, policy-based methods, and proposed model-based methods.

Value-based methods

We use the tabular MC control strategy (TQ), incremental linear MC control strategy (FAi), and batch linear MC control strategy (FAb) with current phase (CP) representation, and current component (CC), partial trace (PT) representations or all meta information vector representation (ALL). The incremental linear MC control uses stochastic gradient method, and the batch linear MC control uses lasso with coefficient = 1e-4, which is executed every 500 updates, i.e. at least 50 full episodes, to reduce the computational overhead. We set the learning rate and the \( \epsilon \) with the conventional values 0.1. We summarize the value-based methods used for comparison in Table 5.6.

The results are reported in Table 5.7 and Figure 5.3. First, we discover that the baseline method TQ/CP performs significantly better than Random, which randomly selects options at each phase, and still maintains an action-value table, which is used to suggest the optimal policy, as in the
Table 5.6: Description of value-based methods

<table>
<thead>
<tr>
<th>Strategy</th>
<th>State representation</th>
<th>Model update / value estimation</th>
<th>Learning rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>TQ/CP</td>
<td>current phase (categorical)</td>
<td>MC tabular value estimation</td>
<td>constant ((\alpha = 0.1))</td>
</tr>
<tr>
<td>TQ/CC</td>
<td>current component (categorical)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TQ/PT</td>
<td>partial trace (categorical)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FAi/CP</td>
<td>current phase (indicator vector)</td>
<td>linear function approximation (incremental update using stochastic gradient method)</td>
<td></td>
</tr>
<tr>
<td>FAi/ALL</td>
<td>current phase, current component, partial trace (indicator vector)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FAb/CP</td>
<td>current phase (indicator vector)</td>
<td>linear function approximation (batch update using lasso)</td>
<td>N/A</td>
</tr>
<tr>
<td>FAb/ALL</td>
<td>current phase, current component, partial trace (binary feature)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Shared configuration

<table>
<thead>
<tr>
<th>Reward calculator</th>
<th>Option selector</th>
</tr>
</thead>
<tbody>
<tr>
<td>total reward (0 for failed episode)</td>
<td>(\epsilon)-greedy ((\epsilon = 0.1))</td>
</tr>
</tbody>
</table>

Table 5.7: Results of value-based methods

<table>
<thead>
<tr>
<th>Strategy</th>
<th>AUC-BP@2K</th>
<th>AUC-EOP@2K</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mean (\pm) sd</td>
<td>min</td>
</tr>
<tr>
<td>Random</td>
<td>.0002 (\pm) .0000</td>
<td>.0001</td>
</tr>
<tr>
<td>TQ/CP</td>
<td>.0352 (\pm) .0057(***)</td>
<td>.0256</td>
</tr>
<tr>
<td>TQ/CC</td>
<td>.0157 (\pm) .0053(***)</td>
<td>.0069</td>
</tr>
<tr>
<td>TQ/PT</td>
<td>.0086 (\pm) .0074(*)</td>
<td>.0022</td>
</tr>
<tr>
<td>FAi/CP</td>
<td>.0356 (\pm) .0065(ns)</td>
<td>.0245</td>
</tr>
<tr>
<td>FAi/ALL</td>
<td>.0218 (\pm) .0069(***)</td>
<td>.0154</td>
</tr>
<tr>
<td>FAb/CP</td>
<td>.0388 (\pm) .0110(ns)</td>
<td>.0205</td>
</tr>
<tr>
<td>FAb/ALL</td>
<td>.0318 (\pm) .0102(*)</td>
<td>.0184</td>
</tr>
</tbody>
</table>

We also find that TQ/CP and FAi/CP perform alike to each other. In fact, they are almost equivalent, with the exception that FAi uses an additional intercept term. If we use CC or PT to represent each state, then due to lack of sufficient training data, they perform more poorly than CP. We may also observe the advantage of CC or PT if we relax the assumption of component independence in the simulation of the configuration space. The batch version (lasso regression for LSE optimization) performs better than the incremental version (via stochastic gradient descent), weakly significantly in the FA/ALL setting, but not significantly in the FA/CP setting. However, the batch linear MC control does not start working until sufficient (with respect to the dimensionality of the vector representation space) training instances are available. The FAb/ALL cannot make
any single prediction in the first 200 episodes, i.e. the options are selected randomly in these 200 episodes. Using a richer feature representation for states has a slower convergence rate. In fact, we observe that a failed episode always caused a huge turbulence to value estimation. We hypothesize that an off-policy learning variant of FAi/ALL might relieve this issue.

We vary the learning rate function for the tabular MC control (TQ) and incremental linear MC control, where we compare the inverse of visit count (denoted as $\frac{1}{N}$) and the constant $\alpha$ ranging from 0.01 to 0.8, and report the results in Figure 5.4. We observe that the inverse of the visit count consistently achieves the same best performance as a carefully picked constant learning rate: 0.1 for TQ/CP, 0.2 for TQ/CC, and 0.4 for TQ/PT, 0.05 for FAi/CP, and 0.1 for FAi/ALL. We also observe that a more comprehensive and sparser representation (e.g., CC, PT, or ALL) tends to
prefer a larger (more aggressive) learning rate, given the maximum of 2000 experiment episodes.

We vary the option selection strategies. We compare UCB1, inverse of visit count \((\frac{1}{N})\) with \(\varepsilon\)-greedy with \(\varepsilon\) ranging from 0.01 to 0.8, and show the results in Figure 5.5. We found that the best behavior performance is achieved when tabular MC control (TQ) uses \(\varepsilon = 0.1\) and linear MC control (FAi) uses a slightly smaller \(\varepsilon\) of 0.05. However, the expected optimal performance prefers a slightly larger \(\varepsilon\): TQ/CP uses 0.1, TQ/CC and TQ/PT use 0.2, and FAi also uses 0.1. Intuitively, with a larger \(\varepsilon\), although a smaller reward may be received from the actual experience, the agent is able to more quickly learn the performance of the component in the configuration space to predict the optimal system.

**Off-policy methods**

We apply the exponentially discounted counter for off-policy learning in the nonstationary environments. We compare the off-policy tabular MC control strategy, the off-policy linear MC control strategy, two-step option selection strategies: random/random (RR), random/UCB1 (RU), and option selection using partial trace as the state representation (RU/PT). Based on grid search of the optimal damping factor and initial padding length, we set damping factor \(\zeta = 0.95\) and initial padding length = 1 for original off-policy variant of the tabular MC control (TQ/*-OFF), damping factor \(\zeta = 0.99\) and initial padding length = 1 for two-step tabular MC control (TQ/*-RR, -RU, and -RU/PT), damping factor \(\zeta = 0.995\) and initial padding length = 5 for all off-policy linear MC control settings (FA/*-OFF, -RR, -RU, and -RU/PT). The probability of selecting a random component in the behavior policy equals to \(\frac{1}{M_i}\), where \(M_i\) is the number of components in the phase, and the probability of selecting an option with the maximum UCB is estimated from the past episodes in the history. The probability of selection a phase-component pair using a two-step selection strategy equals to the multiplication of selecting the phase and the probability of selecting the component. The optimal/greedy option selection strategy is defined as an \(\varepsilon\)-greedy with
Table 5.8: Description of off-policy methods

<table>
<thead>
<tr>
<th>Strategist</th>
<th>Learning rate</th>
<th>Option selector (behavior policy)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TQ*/-OFF</td>
<td>initial padding length = 1,</td>
<td>(\epsilon)-greedy ((\epsilon = 0.1))</td>
</tr>
<tr>
<td>TQ*/-RR</td>
<td>damping factor (\zeta = 0.95)</td>
<td>random/random two-step selection</td>
</tr>
<tr>
<td>TQ*/-RU</td>
<td>initial padding length = 1,</td>
<td>random/UCB1 two-step selection</td>
</tr>
<tr>
<td>TQ*/-RU/PT</td>
<td>damping factor (\zeta = 0.99)</td>
<td>random/UCB1 two-step selection with UCB1 uses a MDP with the state space defined by partial trace</td>
</tr>
</tbody>
</table>

FA*/-OFF          | initial padding length = 5,               | \(\epsilon\)-greedy \((\epsilon = 0.1)\)                           |
| FA*/-RR          | damping factor \(\zeta = 0.995\)          | random/random two-step selection                                     |
| FA*/-RU/PT       | damping factor \(\zeta = 0.995\)          | random/UCB1 two-step selection with UCB1 uses a MDP with the state space defined by partial trace |

Shared configuration

Optimal option selector (greedy / estimation policy)

\(\epsilon\)-greedy \((\epsilon = 0.001)\)

\(\epsilon = 0.001\). The off-policy methods are summarized in Table 5.8.

We report the comparison results of off-policy methods in Table 5.9 and Figures 5.6 and 5.7. First, we find that the off-policy learning method improves the tabular MC control method significantly when the state is represented by CP or CC, and all off-policy two-step selection strategies improve the linear MC control method significantly (at least \(\star\star\), meaning the confidence level \(> 0.99\)), regardless of the state representation.

From the expected optimal performance curves in Figure 5.6, we can see that the off-policy tabular MC control strategies may face a convergence problem, especially when they use the random-UCB1 (RU) option selection method, which does not happen with the off-policy linear MC control strategies, as we can see from Figure 5.7. This phenomenon is an indicator of non-convergence, due to a large (effective) learning rate \((\alpha \text{ or } 1 - \zeta)\) and/or large exploration rate \((\epsilon)\), the two parameters of the exploration strategy. Whenever we observe this phenomenon, we should avoid this strategy, at least for this particular budget and this task. In this case, we find that it is because the off-policy linear MC control uses a larger initial padding length and damping factor, which both result in an effectively smaller learning rate. When initial padding length = 5 and damping factor \(\zeta = 0.995\), the learning rate starts with 0.183 and asymptotically decreases to 0.005. On the contrary, the off-policy tabular MC control uses a relatively large effective learning rate (0.1), and thus the optimal policy changes after almost every episode, when we combine with an aggressive exploration method. In fact, one can imagine that a large learning rate can help the model quickly learn a local optimum in an earlier stage, but soon disallow it to converge. Due to the same reason, reinforcement learning problems sometimes consider a decayed learning rate combined with a decayed exploration rate (if there is an implicit budget constraint), where people believe that converging to a local optimum without further exploration is better than not converging at all. In this thesis, we focus mostly on constant learning rate and constant exploration rate (or constant
Table 5.9: Results of off-policy methods

<table>
<thead>
<tr>
<th>Strategy</th>
<th>AUC-BP@2K</th>
<th></th>
<th></th>
<th>AUC-EOP@2K</th>
<th></th>
<th></th>
<th>Comparand</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mean ± sd</td>
<td>min</td>
<td>max</td>
<td>mean ± sd</td>
<td>min</td>
<td>max</td>
<td></td>
</tr>
<tr>
<td>TQ/CP-OFF</td>
<td>0.0510 ± 0.0081***</td>
<td>0.0345</td>
<td>0.0612</td>
<td>0.4678 ± 0.0792***</td>
<td>0.3131</td>
<td>0.5810</td>
<td>TQ/CP</td>
</tr>
<tr>
<td>TQ/CC-OFF</td>
<td>0.0258 ± 0.0087**</td>
<td>0.0079</td>
<td>0.0390</td>
<td>0.2720 ± 0.0902**</td>
<td>0.0800</td>
<td>0.3994</td>
<td>TQ/CC</td>
</tr>
<tr>
<td>TQ/PT-OFF</td>
<td>0.0066 ± 0.0031ns</td>
<td>0.0027</td>
<td>0.0112</td>
<td>0.0820 ± 0.0399ns</td>
<td>0.0315</td>
<td>0.1482</td>
<td>TQ/PT</td>
</tr>
<tr>
<td>FA/CP-OFF</td>
<td>0.0267 ± 0.0113†</td>
<td>0.0061</td>
<td>0.0427</td>
<td>0.2350 ± 0.1054†</td>
<td>0.0473</td>
<td>0.3944</td>
<td>FA/CP</td>
</tr>
<tr>
<td>FA/ALL-OFF</td>
<td>0.0376 ± 0.0121**</td>
<td>0.0250</td>
<td>0.0616</td>
<td>0.3852 ± 0.1266**</td>
<td>0.2479</td>
<td>0.6327</td>
<td>FA/ALL</td>
</tr>
<tr>
<td></td>
<td>0.0509 ± 0.0100**</td>
<td>0.0347</td>
<td>0.0641</td>
<td>0.5576 ± 0.1129***</td>
<td>0.3753</td>
<td>0.7144</td>
<td>TQ/CP</td>
</tr>
<tr>
<td>TQ/CC-RR</td>
<td>0.0090 ± 0.0041**</td>
<td>0.0041</td>
<td>0.0192</td>
<td>0.1783 ± 0.0814ns</td>
<td>0.0949</td>
<td>0.4033</td>
<td>TQ/CC</td>
</tr>
<tr>
<td>TQ/PT-RR</td>
<td>0.0043 ± 0.0041ns</td>
<td>0.0016</td>
<td>0.0147</td>
<td>0.1157 ± 0.1141ns</td>
<td>0.0402</td>
<td>0.3976</td>
<td>TQ/PT</td>
</tr>
<tr>
<td>FA/CP-RR</td>
<td>0.0445 ± 0.0106*</td>
<td>0.0304</td>
<td>0.0655</td>
<td>0.4764 ± 0.1272**</td>
<td>0.3109</td>
<td>0.7102</td>
<td>FA/CP</td>
</tr>
<tr>
<td>FA/ALL-RR</td>
<td>0.0403 ± 0.0065***</td>
<td>0.0274</td>
<td>0.0492</td>
<td>0.4915 ± 0.0811***</td>
<td>0.3347</td>
<td>0.6084</td>
<td>FA/ALL</td>
</tr>
<tr>
<td>TQ/CP-RU</td>
<td>0.0509 ± 0.0143*</td>
<td>0.0255</td>
<td>0.0761</td>
<td>0.3889 ± 0.1029ns</td>
<td>0.2034</td>
<td>0.5749</td>
<td>TQ/CP</td>
</tr>
<tr>
<td>TQ/CC-RU</td>
<td>0.0055 ± 0.0036***</td>
<td>0.0019</td>
<td>0.0141</td>
<td>0.1166 ± 0.0698ns</td>
<td>0.0423</td>
<td>0.2781</td>
<td>TQ/CC</td>
</tr>
<tr>
<td>TQ/PT-RU</td>
<td>0.0046 ± 0.0029ns</td>
<td>0.0012</td>
<td>0.0104</td>
<td>0.1293 ± 0.0716ns</td>
<td>0.0372</td>
<td>0.2569</td>
<td>TQ/PT</td>
</tr>
<tr>
<td>FA/CP-RU</td>
<td>0.0885 ± 0.0055***</td>
<td>0.0798</td>
<td>0.0960</td>
<td>0.6963 ± 0.0412***</td>
<td>0.6253</td>
<td>0.7533</td>
<td>FA/CP</td>
</tr>
<tr>
<td>FA/ALL-RU</td>
<td>0.0727 ± 0.0121***</td>
<td>0.0472</td>
<td>0.0903</td>
<td>0.5721 ± 0.0849***</td>
<td>0.3889</td>
<td>0.7007</td>
<td>FA/ALL</td>
</tr>
<tr>
<td>TQ/CP-RU/PT</td>
<td>0.0579 ± 0.0140***</td>
<td>0.0396</td>
<td>0.0819</td>
<td>0.4542 ± 0.0958**</td>
<td>0.3217</td>
<td>0.6370</td>
<td>TQ/CP</td>
</tr>
<tr>
<td>TQ/CC-RU/PT</td>
<td>0.0102 ± 0.0038*</td>
<td>0.0035</td>
<td>0.0149</td>
<td>0.1593 ± 0.0515ns</td>
<td>0.0642</td>
<td>0.2304</td>
<td>TQ/CC</td>
</tr>
<tr>
<td>TQ/PT-RU/PT</td>
<td>0.0053 ± 0.0034ns</td>
<td>0.0009</td>
<td>0.0141</td>
<td>0.1526 ± 0.0881ns</td>
<td>0.0274</td>
<td>0.3759</td>
<td>TQ/PT</td>
</tr>
<tr>
<td>FA/CP-RU/PT</td>
<td>0.0750 ± 0.0060***</td>
<td>0.0674</td>
<td>0.0839</td>
<td>0.6261 ± 0.0604***</td>
<td>0.5470</td>
<td>0.7355</td>
<td>FA/CP</td>
</tr>
<tr>
<td>FA/ALL-RU/PT</td>
<td>0.0747 ± 0.0094***</td>
<td>0.0619</td>
<td>0.0888</td>
<td>0.5920 ± 0.0824***</td>
<td>0.4939</td>
<td>0.7462</td>
<td>FA/ALL</td>
</tr>
</tbody>
</table>

exploration decay rate). As a next step, we may consider to apply an adaptive learning rate, such as AdaGrad [52], Adam [98], etc., and further investigate an adaptive exploration rate following the similar idea.

The overall best performance from off-policy methods is given by FA/CP-RU. Two best-performing strategies in the category of off-policy linear MC control, FA/CP-RU and FA/CP-RU/PT, use RU for option selection and use current phase (CP) for state representation. Compared with the RR method, the RU method can intentionally avoid greedy options, as these options have 10 times more visits than non-greedy options.

We analyze the impact of the parameters of the exponentially discounted counter, including damping factor and initial padding length, on the performance of off-policy methods. We first compare the use of exponentially discounted counter for off-policy learning in the nonstationary environments (i.e. Eq 5.18), where $\zeta$ ranges from 0.999 to 0.9 and the initial padding length remains the same (1 or 5), with the conventional non-discounted counter for off-policy learning in the stationary environments (i.e. Eq 5.17, denoted as $\frac{1}{N}$), and report the results in Figures 5.8 and 5.9. We find that the nonstationary off-policy methods, which use exponentially discounted
counter, perform better than the stationary off-policy methods. Most off-policy methods prefer a damping factor $\zeta$ between 0.998 to 0.98, and they start performing worse when damping factor is decreased to and below 0.95.

Then, we compare the impact of initial padding length used in the exponentially discounted counter on the performance of the off-policy methods, and report the results in Figures 5.10 and 5.11. A large initial padding length leads to a small effective learning rate used in the first several updates. When deciding the value for the initial padding length in off-policy tabular MC control, we find TQ/CP-OFF prefers 0, TQ/CP-RR prefers 5, TQ/CP-RU prefers 10, and TQ/CP-RU/PT prefers 5. Two best-performing strategies, FA/CP-RU and FA/CP-RU/PT, are insensitive to the
initial padding length, whereas FA/ALL-RU and FA/ALL-RU/PT, those that use ALL meta information to represent a state, prefer a larger initial padding (> 20) or smaller effective learning rate, which coincides with our observation of the on-policy value-based methods. The strategies using \( \epsilon \)-greedy (-OFF) or RR to select options prefer an initial padding length between 2 and 50.

Finally, we vary \( \epsilon \) in the off-policy methods that use \( \epsilon \)-greedy strategy to select options, and report the results in Figure 5.12. We find that most best-performing strategies have the highest expected optimal performance when \( \epsilon \) is set 0.1, which is again slightly higher than the \( \epsilon \) that maximizes the behavior performance (0.05).

108
Policy-based methods

We use current phase (CP), current component (CC), partial trace (PT), or all meta information (ALL) to represent states. We implement the policy-gradient MC control method (REINFORCE), actor-critic methods, advantage actor-critic methods, where we use the indicator vector representation for state-action pairs in the REINFORCE algorithm as well as the actors in the actor-critic methods. We use a tabular action-/state-value estimator or a linear action-/state-value estimator as the critic of each actor-critic method. As discussed in Section 5.4.2, we use an inverse decayed learning rate for REINFORCE algorithm, and a small constant learning rate ($\alpha = 0.01$) for
Table 5.10: Descriptions of policy-based methods

<table>
<thead>
<tr>
<th>Strategy</th>
<th>State representation</th>
<th>Model update / value estimation</th>
<th>Learning rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>REINF/CP</td>
<td>current phase (indicator vector)</td>
<td>REINFORCE</td>
<td>inverse decayed $(\alpha = 1/n)$</td>
</tr>
<tr>
<td>REINF/CC</td>
<td>current component (indicator vector)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REINF/PT</td>
<td>partial trace (indicator vector)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TQAC/CP</td>
<td>current phase (indicator vector/categorical)</td>
<td>actor-critic method (critic is a tabular action-value estimator)</td>
<td></td>
</tr>
<tr>
<td>TQAC/CC</td>
<td>current component (indicator vector/categorical)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TQAC/PT</td>
<td>partial trace (indicator vector/categorical)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TAAC/CP</td>
<td>current phase (indicator vector/categorical)</td>
<td>advantage actor-critic method (critic and baseline are tabular action- / state-value estimators)</td>
<td></td>
</tr>
<tr>
<td>TAAC/CC</td>
<td>current component (indicator vector/categorical)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TAAC/PT</td>
<td>partial trace (indicator vector/categorical)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LQAC/CP</td>
<td>current phase (indicator vector)</td>
<td>actor-critic method (critic is a tabular action-value estimator)</td>
<td></td>
</tr>
<tr>
<td>LQAC/CC</td>
<td>current component (indicator vector)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LQAC/PT</td>
<td>partial trace (indicator vector)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LQAC/ALL</td>
<td>current phase, current component, partial trace (binary feature)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LAAC/CP</td>
<td>current phase (indicator vector)</td>
<td>advantage actor-critic method (critic and baseline are linear action- / state-value estimators)</td>
<td></td>
</tr>
<tr>
<td>LAAC/CC</td>
<td>current component (indicator vector)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LAAC/PT</td>
<td>partial trace (indicator vector)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LAAC/ALL</td>
<td>current phase, current component, partial trace (binary feature)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Shared configuration

<table>
<thead>
<tr>
<th>Policy parameterization</th>
<th>Reward calculator</th>
<th>Option selector</th>
</tr>
</thead>
<tbody>
<tr>
<td>softmax</td>
<td>total reward (0 for failed episode)</td>
<td>$\epsilon$-greedy ($\epsilon = 0.1$)</td>
</tr>
</tbody>
</table>
actor-critic methods. For critics, we set the learning rate $\beta = 0.1$. We use the \( \epsilon \)-greedy strategy to explore the configuration space. The policy-based methods that are compared in this section are summarized in Table 5.10.

We report the comparison results in Table 5.11 and Figure 5.13. First, we find that the policy-based methods perform worse than the value-based methods, such as tabular MC control and linear MC control. The REINFORCE algorithm and the advantage actor-(linear-)critic (LAAC) method have the highest performance among the policy-based methods, and the advantage actor-critic method using either a tabular or linear critic outperforms its standard actor-critic method counterpart. The performance of both algorithms benefit from using all meta information (ALL) instead of current phase (CP) representation. Most high-performing policy-based methods exhibit high
Table 5.11: Results of policy-based methods

<table>
<thead>
<tr>
<th>Strategy</th>
<th>AUC-BP@2K mean ± sd</th>
<th>min</th>
<th>max</th>
<th>AUC-EOP@2K mean ± sd</th>
<th>min</th>
<th>max</th>
<th>Comparand</th>
</tr>
</thead>
<tbody>
<tr>
<td>REINF/CP</td>
<td>0.0173 ± 0.0104  *</td>
<td>0.009</td>
<td>0.0360</td>
<td>1.503 ± 0.0931 *</td>
<td>0.0049</td>
<td>0.3195</td>
<td>Random</td>
</tr>
<tr>
<td>REINF/CC</td>
<td>0.0137 ± 0.0055 ns</td>
<td>0.0040</td>
<td>0.0228</td>
<td>1.386 ± 0.0628 ns</td>
<td>0.0332</td>
<td>0.2505</td>
<td>REINF/CP</td>
</tr>
<tr>
<td>REINF/PT</td>
<td>0.0033 ± 0.0027 ns</td>
<td>0.0008</td>
<td>0.0111</td>
<td>0.0371 ± 0.0326 ns</td>
<td>0.0081</td>
<td>0.1296</td>
<td>REINF/CC</td>
</tr>
<tr>
<td>REINF/ALL</td>
<td>0.0284 ± 0.0129 †</td>
<td>0.024</td>
<td>0.457</td>
<td>0.2597 ± 0.1223 †</td>
<td>0.0166</td>
<td>0.4198</td>
<td>REINF/CP</td>
</tr>
<tr>
<td>TQAC/CP</td>
<td>0.0008 ± 0.0011  ‡</td>
<td>0.0000</td>
<td>0.0029</td>
<td>0.0055 ± 0.0077 ‡</td>
<td>0.0000</td>
<td>0.0210</td>
<td>REINF/CP</td>
</tr>
<tr>
<td>TQAC/CC</td>
<td>0.0010 ± 0.0012 ns</td>
<td>0.0001</td>
<td>0.0043</td>
<td>0.060 ± 0.0103 ns</td>
<td>0.0002</td>
<td>0.0359</td>
<td>TQAC/CP</td>
</tr>
<tr>
<td>TQAC/PT</td>
<td>0.0005 ± 0.0005 ns</td>
<td>0.0002</td>
<td>0.0020</td>
<td>0.0029 ± 0.0036 ns</td>
<td>0.0003</td>
<td>0.0118</td>
<td>TQAC/CC</td>
</tr>
<tr>
<td>TAAC/CP</td>
<td>0.0158 ± 0.0053 ns</td>
<td>0.0077</td>
<td>0.0267</td>
<td>1.340 ± 0.0475 ns</td>
<td>0.0614</td>
<td>0.2359</td>
<td>TQAC/CP</td>
</tr>
<tr>
<td>TAAC/CC</td>
<td>0.0190 ± 0.0111 ns</td>
<td>0.0075</td>
<td>0.0425</td>
<td>0.1984 ± 0.1220 ns</td>
<td>0.0671</td>
<td>0.4599</td>
<td>TAAC/CP</td>
</tr>
<tr>
<td>TAAC/PT</td>
<td>0.0063 ± 0.0052 ns</td>
<td>0.0009</td>
<td>0.0189</td>
<td>0.0818 ± 0.0699 *</td>
<td>0.0092</td>
<td>0.2531</td>
<td>TAAC/CC</td>
</tr>
<tr>
<td>LQAC/CP</td>
<td>0.0025 ± 0.0056 ns</td>
<td>0.0000</td>
<td>0.0191</td>
<td>0.195 ± 0.460 ns</td>
<td>0.0000</td>
<td>0.1560</td>
<td>TQAC/CP</td>
</tr>
<tr>
<td>LQAC/CC</td>
<td>0.0020 ± 0.0026 ns</td>
<td>0.0002</td>
<td>0.0085</td>
<td>0.0173 ± 0.0251 ns</td>
<td>0.0007</td>
<td>0.0748</td>
<td>LQAC/CP</td>
</tr>
<tr>
<td>LQAC/PT</td>
<td>0.0004 ± 0.0003 ns</td>
<td>0.0001</td>
<td>0.0009</td>
<td>0.0036 ± 0.0032 ns</td>
<td>0.0003</td>
<td>0.0090</td>
<td>LQAC/CC</td>
</tr>
<tr>
<td>LQAC/ALL</td>
<td>0.0008 ± 0.0010 ns</td>
<td>0.0000</td>
<td>0.0029</td>
<td>0.0056 ± 0.0075 ns</td>
<td>0.0001</td>
<td>0.0212</td>
<td>LQAC/CP</td>
</tr>
<tr>
<td>LAAC/CP</td>
<td>0.0241 ± 0.0130 ns</td>
<td>0.0091</td>
<td>0.5551</td>
<td>0.2121 ± 0.1261 **</td>
<td>0.0721</td>
<td>0.5211</td>
<td>LQAC/CP</td>
</tr>
<tr>
<td>LAAC/CC</td>
<td>0.0082 ± 0.0045 ns</td>
<td>0.0013</td>
<td>0.0173</td>
<td>0.0836 ± 0.0477 *</td>
<td>0.0132</td>
<td>0.1679</td>
<td>LAAC/CP</td>
</tr>
<tr>
<td>LAAC/PT</td>
<td>0.0075 ± 0.0047 ns</td>
<td>0.0009</td>
<td>0.0147</td>
<td>0.1003 ± 0.0642 ns</td>
<td>0.0112</td>
<td>0.1984</td>
<td>LAAC/CC</td>
</tr>
<tr>
<td>LAAC/ALL</td>
<td>0.0271 ± 0.0184 ns</td>
<td>0.0040</td>
<td>0.0666</td>
<td>0.2428 ± 0.1769 ns</td>
<td>0.0324</td>
<td>0.6372</td>
<td>LAAC/CP</td>
</tr>
</tbody>
</table>

vance, where the maximum performance among each 10 random runs beats that of a baseline value-based method, which suggests that policy-based methods may require more trials with random initialization and a proper strategy to select the optimal policy. We can also see from Figure 5.13 that the policy-based methods have better convergence performance than other methods, although they tend to converge to local optima, which is also a well-known phenomenon of policy-gradient methods [189].

We vary learning rate function in the REINFORCE algorithm and the actors in the actor-critic methods. We compare a constant learning rate $\alpha$ ranging from 0.001 to 0.1 with an inverse decayed learning rate ($\frac{1}{N}$), and plot in Figure 5.14. We can clearly see that the REINFORCE algorithm prefers an inverse decayed learning rate, regardless of the state representation, and so does the actor-(linear-)critic method using all meta information (LAAC/ALL), the highest-performing actor-critic method, while other actor-critic methods tend to prefer a constant learning rate between 0.002 and 0.01. During the exploration process, the policy optimization algorithm tends to select a current (local) optimal option more often than other options, no matter which option selection strategy is used. Due to the fact that, unlike many other reinforcement learning environments, the total reward in configuration space exploration is always nonnegative, the REINFORCE algorithm, which uses the original reward as target, always increases the confidence of selecting an option if
this option is used in an episode. Therefore, the exploration process is easily stuck in a local optimum option with a constant learning rate. In contrast, a decayed learning rate often slows down the process from falling into the local optimum. An interesting next step is to modify the reward function that can also generates negative signals, as the advantage actor-critic offsets the original total reward by a baseline (state-value function).

We compare different option selection strategies in Figure 5.15. We see that most policy-based methods prefer a constant $\epsilon$ between 0.05 and 0.4, and perform consistently well when $\epsilon$ is set between 0.1 and 0.2.
### Table 5.12: Descriptions of model-based methods

<table>
<thead>
<tr>
<th>Strategist</th>
<th>State representation</th>
<th>Model update</th>
<th>Learning rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>B(E)Mi+/CP</td>
<td>current phase (indicator vector)</td>
<td>linear regression</td>
<td>constant</td>
</tr>
<tr>
<td>B(E)Mi+/CC</td>
<td>current component (indicator vector)</td>
<td></td>
<td>((\alpha = 0.05))</td>
</tr>
<tr>
<td>B(E)Mi+/PT</td>
<td>partial trace (indicator vector)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B(E)Mi+/ALL</td>
<td>current phase, current component, partial trace</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B(E)Mi*/CP</td>
<td>current phase (indicator vector)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B(E)Mi*/CC</td>
<td>current component (indicator vector)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B(E)Mi*/PT</td>
<td>partial trace (indicator vector)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B(E)Mi*/ALL</td>
<td>current phase, current component, partial trace</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B(E)Mb+/CP</td>
<td>current phase (indicator vector)</td>
<td>linear regression (batch update using lasso regression, coefficient = 1e-4, executed every 10 updates, i.e. 10 full episodes)</td>
<td>N/A</td>
</tr>
<tr>
<td>B(E)Mb+/CC</td>
<td>current component (indicator vector)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B(E)Mb+/PT</td>
<td>partial trace (indicator vector)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B(E)Mb+/ALL</td>
<td>current phase, current component, partial trace</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Value estimation**

- average of top-\(k\) (\(= 10\)) returns estimated from \(N\) (\(= 200\)) MC samples starting from the current state

**Reward calculator**

- total reward (0 for failed episode)

**Option selector**

- \(\epsilon\)-greedy (\(\epsilon = 0.1\))

---

**Model-based methods**

We compare the model-based methods, including the additive model, multiplicative model, as well as their variant models with error prediction. Again, we use current phase (CP), current component (CC), partial trace (PT), and all meta information (ALL), for state representation in indicator vector forms. We train the additive benefit model (BMi+) and the multiplicative benefit model (BMi*) with the stochastic gradient descent algorithm, which uses a constant learning rate \((\alpha = 0.05)\). We compare the incremental update algorithm with the batch update method (BMb+) in training the additive benefit model, which uses lasso, with coefficient = 1e-4, executed every 10 updates, i.e. 10 full episodes. When estimating the expected reward for model-based methods, we use a damping factor \(\gamma = 0.5\), and select to use the top \(k = 10\) returns from a total of \(N = 200\) samples. We train a support vector classifier, using \(L_1\) regularizer \(L_2\) loss, to predict the error, and repeat after every 10 updates. We use BEMi+, BEMb+, and BEMi* to denote the strategies that integrate an error prediction model. We summarize the model-based methods in Table 5.12.
We report the performance results in Table 5.13 and Figure 5.16. We find that given the allowed number of training episodes (2000), we achieve the best performance when using current phase (CP) as state representation. The additive model (BMi+/CP) performs significantly better than the baseline TQ/CP method and the multiplicative model (BMi*/CP), due to lack of the error prediction module. Policies of length shorter than the number of phases, which cannot accomplish the task, are often estimated to have higher performance by the multiplicative model. Although the batch update algorithms (BMb+) have higher performance than the incremental update counterparts, they suffer from substantial processing overtime, esp. for high-sparsity feature space. Taking BMi(b)+/CC and BMi(b)+/PT as examples, we timed the execution of the first 1000 episodes (10 steps per episode). On average, BMi+/CC and BMi+/PT took 4.2 – 7.5 milliseconds per step, while BMb+/CC and BMb+/PT took 810.6 – 944.8 milliseconds per step. Also, we can see from Figure

Table 5.13: Results of model-based methods

<table>
<thead>
<tr>
<th>Strategy</th>
<th>AUC-BP@2K</th>
<th>AUC-EOP@2K</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mean ± sd</td>
<td>min</td>
</tr>
<tr>
<td>BMi+/CP</td>
<td>0.503 ± 0.0141</td>
<td>0.327</td>
</tr>
<tr>
<td>BMi+/CC</td>
<td>0.280 ± 0.0070***</td>
<td>0.118</td>
</tr>
<tr>
<td>BMi+/PT</td>
<td>0.013 ± 0.006***</td>
<td>0.026</td>
</tr>
<tr>
<td>BMi+/ALL</td>
<td>0.038 ± 0.0048*</td>
<td>0.0296</td>
</tr>
<tr>
<td>BMi*/CP</td>
<td>0.006 ± 0.006***</td>
<td>0.0001</td>
</tr>
<tr>
<td>BMi*/CC</td>
<td>0.0008 ± 0.0003ns</td>
<td>0.004</td>
</tr>
<tr>
<td>BMi*/PT</td>
<td>0.0055 ± 0.0003*</td>
<td>0.0002</td>
</tr>
<tr>
<td>BMi*/ALL</td>
<td>0.0008 ± 0.0006ns</td>
<td>0.0001</td>
</tr>
<tr>
<td>BEMi+/CP</td>
<td>0.0296 ± 0.0130**</td>
<td>0.0048</td>
</tr>
<tr>
<td>BEMi+/CC</td>
<td>0.121 ± 0.0044**</td>
<td>0.0666</td>
</tr>
<tr>
<td>BEMi+/PT</td>
<td>0.023 ± 0.0007***</td>
<td>0.0017</td>
</tr>
<tr>
<td>BEMi+/ALL</td>
<td>0.033 ± 0.0021***</td>
<td>0.0007</td>
</tr>
<tr>
<td>BEMi*/CP</td>
<td>0.032 ± 0.0100ns</td>
<td>0.0150</td>
</tr>
<tr>
<td>BEMi*/CC</td>
<td>0.0091 ± 0.0034***</td>
<td>0.0024</td>
</tr>
<tr>
<td>BEMi*/PT</td>
<td>0.021 ± 0.0016***</td>
<td>0.0006</td>
</tr>
<tr>
<td>BEMi*/ALL</td>
<td>0.0085 ± 0.0044***</td>
<td>0.0019</td>
</tr>
<tr>
<td>BMb+/CP</td>
<td>0.555 ± 0.0055***</td>
<td>0.0047</td>
</tr>
<tr>
<td>BMb+/CC</td>
<td>0.0107 ± 0.0025***</td>
<td>0.0071</td>
</tr>
<tr>
<td>BMb+/PT</td>
<td>0.047 ± 0.0019***</td>
<td>0.0027</td>
</tr>
<tr>
<td>BMb+/ALL</td>
<td>0.0261 ± 0.0064***</td>
<td>0.0150</td>
</tr>
<tr>
<td>BEMb+/CP</td>
<td>0.0443 ± 0.0184***</td>
<td>0.0143</td>
</tr>
<tr>
<td>BEMb+/CC</td>
<td>0.0082 ± 0.0026*</td>
<td>0.0045</td>
</tr>
<tr>
<td>BEMb+/PT</td>
<td>0.046 ± 0.0016ns</td>
<td>0.0023</td>
</tr>
<tr>
<td>BEMb+/ALL</td>
<td>0.0320 ± 0.0114ns</td>
<td>0.0070</td>
</tr>
</tbody>
</table>
5.16 that a strategy using batch update such as BEMb+/CP tends to have a preparation period (around 200 episodes for BEMb+/CP), before it starts making prediction, and it finally catches up with the BEMi+/CP at around 500 episodes. When we integrate the error prediction module, the additive BEMi+/CP strategy slightly outperforms the BMi+/CP strategy, and the multiplicative BEMi*/CP strategy greatly outperforms the BMi*/CP strategy, which confirms our hypothesis about the performance of additive model and multiplicative model.

We observe the impact of the learning rate function for the model-based strategies using an incremental update method on their learning performance, where we use a constant learning rate $\alpha$ ranging from 0.002 to 0.2 or an inverse decayed learning rate $(\frac{1}{N})$. We plot them in Figure 5.17. We can see that the two best-performing strategies BEMi+/CP and BEMi*/CP prefer a constant
learning rate between 0.02 and 0.05. Figure 5.18 shows the learning performance of various model-based methods using different option selection strategies, where we set a constant $\epsilon$ from 0.01 to 0.8, an inverse decayed ($\frac{1}{N}$), or replace with UCB1. Similar to other strategies such as value-based methods or policy-based methods, model-based methods prefer a constant $\epsilon$ between 0.1 and 0.2. We see that an aggressive exploration strategy (e.g. $\epsilon > 0.4$) should be avoided when selecting options in model-based methods as well. In fact, if an aggressive exploration strategy is used in a value-based method, the behavior policy is “too off” the estimation policy to accurately estimate the values. In a model-based method, we can accurately learn the benefit of an exploratory option without any modification to the policy optimization method (e.g. off-policy learning adjustment), as long as the episode contains mostly greedy options whose benefits have already been accurately estimated. However, if an aggressive exploration strategy is used in a model-based method, the learning algorithm is quickly immersed in a very sparse space, which makes it too slow to accurately estimate the benefits within the limit of the episode allowance.

Finally, we vary the parameters in the sampling based value estimation method for model-based strategies, which include the damping factor $\gamma$, number of samples $N$ and the number of top estimated values for averaging $k$. We plot them in Figures 5.19, 5.20, and 5.21 respectively. First, we see that the best-performing model-based strategy, BEMi+/CP, prefers a damping factor $\gamma = 0.5$. From Figure 5.20, we see that most model-based strategies slightly prefer a larger $N$. However, the performance with even the smallest $N = 100$ is still comparable to the performance with $N = 300$. Also, we can see from Figure 5.21 that most model-based strategies prefer a large $k$, but many times also work well with a small $k$. Actually, making this decision is similar to selecting the $k$ in the $k$-nearest neighbor algorithm, which is usually task-specific and determined based on data observation.
Temporal difference methods

Temporal difference methods, such as Sarsa(λ) and Q-learning, tend to have better asymptotic convergence property than the Monte Carlo methods but slower convergence rate. Finally, in this subsection, we compare the Monte Carlo method, which is used in this section and the following case studies, with Sarsa(λ) and Q-learning using the same simulated problem set and configuration space, but executed each for 5000 episodes, in contrast to 2000 episodes in later sections. The learning curves of these methods are shown in Figure 5.22 and the change of performance in terms of AUC-EOP@5000 are reported in Figure 5.23.

We see that these methods have a better performance than a random method, and their per-
performance is also improved throughout the entire exploration process consisting of 5000 episodes. However, the performance is much lower than that of the Monte Carlo method. The reason is that the simulated configuration space is convex, and the performance of each component is independent of other components, which, a simple setting, unsurprisingly prefers a strategy that uses no bootstrapping to smooth the value estimation.

**Summary**

We find both the off-policy methods that use the exponentially discounted counter for nonstationary environments and the model-based methods outperform the conventional value-based methods.
Most high-performing strategies prefer the $\epsilon$-greedy (with $\epsilon = 0.1$) method for option selection, except for two-step option selection, and prefer a small learning rate (usually below 0.05), due to the large variance and the nonstationarity of the environment.

### 5.5.4 Comparison of Learning Performance in Different Environments

In this subsection, we vary the environments: the problem set and the configuration space from their default setting (as in Table 5.5), and compare the learning curves of the most representative and best-performing strategies, which include TQ/CP, FA/CP, TQ/CP-OFF, FA/CP-RU, REINF/ALL, LAAC/ALL, BEMi+/CP and BEMi*/CP.

#### Difficulty and diversity of the problem

We first observe the impact of the difficulty of the problem by varying the input mean. In the previous section, we set the input mean as 0.5 by using parameters $a = 12$, $b = 12$ in the beta distribution. Here, we use $a/b = 0.8/7.2$, $3/12$, $6/14$, $9.2/13.8$, $12/12$, $13.8/9.2$, $14/6$, $12/3$, $7.2/0.8$ to define a series of beta distributions that have means at 0.1, ..., 0.9 and the same variance 0.01. We plot the learning performance in terms of AUC-BP@2000 and AUC-EOP@2000 for each given input mean in Figure 5.24.

We note that the behavior performance curves reflect the actual return received at the end of each episode. Therefore, we see that as the input mean decreases from 0.9 to 0.1, the AUC-BP number also unsurprisingly decreases near-linearly to 0. Expected optimal performance is a normalized metric, and the AUC-EOP is also a number between 0 and 1, reflecting the steepness of the learning curve. We can see that when the input mean decreases from 0.4, the best-performing learning strategy, FA/CP-RU, starts performing worse than when the input mean is above 0.4.

$^{14}$Theoretical study is needed in order to confirm the linearity of the correlation between the two variables.
When the input mean decreases from 0.2 to 0.1, almost all the strategies start “struggling” about this task and show a 30% of decrease in the performance in terms of AUC-EOP. The reason that the problem becomes harder when the mean decreases to 0.1 is actually that the relative ratio between the mean and the variance determines the uncertainty.

Next, we change the diversity of the problem by varying the input variance. We set the parameters a/b = 99.5/99.5, 49.5/49.5, 24.5/24.5, 12/12, 5.75/5.75, 2.625/2.625, 1.0625/1.0625 respectively, which produce a series of symmetric beta distributions with mean fixed at 0.5 and variances of 0.00125, 0.0025, 0.005, 0.01, 0.02, 0.04, and 0.08 respectively. We plot it in Figure 5.25. We see that as the variance increases, the performance of most strategies drops slightly. In fact, a consequence of high-variance input set is that the probability of an easy input (high value) being executed by a suboptimal configuration and thus receiving a high return becomes greater, which may easily confuse the policy optimization strategy. The results show that the MC sampling based strategies can overcome this issue to a certain level.

**Length of the pipeline**

In earlier experiments, we use a configuration space that contains 100 components equally integrated at 10 phases. Some analytics tasks may require fewer or more phases. Here, we vary the length of the pipeline from 2 to 20, where the same 10 configured components are integrated at each phase, and plot the results in Figure 5.26. The AUC-BP curves show that as the number of phases increases, the behavior performance drops to near 0, because the terminal reward is calculated from multiplying all the component task performance, which is at most 0.9. From the AUC-EOP curves, we can see when the task only requires 2 processing steps, most learning strategies can achieve an AUC-EOP of 0.8 or above, or even close to 1, which suggests that these learning strategies can quickly find the globally optimal policy using much fewer episodes than.
2000. Then, as the number of phases increases, the performance of most learning strategies drops near-linearly but with different slopes.

The best-performing learning strategy, FA/CP-RU, also has the slowest slope and can achieve an AUC-EOP of above 0.3 even when the length of the pipeline increases to 20. The AUC-EOP curves also suggest that we may consider to train the learning strategies using more episodes if we have a longer pipeline.

Size of the configuration space

As the development of the information system continues, more and more compatible components may be integrated into the framework, and thus increase the number of options at a single phase or multiple phases. We first vary the number of components at a single phase, Phase 6, where we change \( M_6 \) from the minimum of 2 components to the maximum of 20 components. The mean of the task performance of the \( j \)-th component at Phase 6 is \( \frac{j}{M_6} \) for \( j = 0, \ldots, M_6 - 1 \), which means if we have only two components, then there is a simulated component that always causes an \texttt{IntendedException} and the other component has a task performance mean of 0.5, whereas if we have 20 components, then there are 19 components that have non-zero task performance mean, with the two best components have task performance means of 0.95 and 0.9. Then, we simultaneously change the number of components in every other phase (i.e. 5 out of 10 total phases) from 2 to 20. Similarly, the mean of the task performance of the \( j \)-th component at the varied phase \( i \) is \( \frac{j}{M_i} \) for \( j = 0, \ldots, M_i - 1 \). We show the performance results of varying the number of components in a single phase and in multiple phases in Figures 5.27 and 5.28.

Both variations exhibit a similar performance trend that as the number of components decreases from 20 to 6, the learning strategies achieve higher performance in terms of AUC-EOP, since the size of the configuration space becomes smaller. However, as the number of components further
Figure 5.26: Varying the length of the pipeline

decreases to 4 or 2, the performance of four learning strategies, TQ/CP, TQ/CP-OFF, REINF/ALL, and LAAC/ALL, keeps increasing, while the performance of the other four learning strategies, FAi/CP, FAi/CP-RU, BEMi+/CP, and BEMi*/CP drops. We find the reason of the performance drop of FAi strategies is that a positive intercept is often learned as the result of our definition of the reward function and a smaller positive weight is estimated for each option exhibiting above-average performance and a smaller negative weight is estimated otherwise. An update in the stochastic gradient descent algorithm calculates the delta (increment) based on the difference between the sum of the weight of the component and the intercept and the return, and therefore, if the return is often zero (half of the components in half phases make errors if the number of components is 2), and the intercept is larger than the weight, the delta can be much greater than the original weight and thus causes huge turbulence.

Task performance of the components

In the previous experiments, we design a configuration space that contains components whose task performance mean follows a uniform distribution, i.e. 0.1, . . . , 0.9. Now, we consider six different distributions of task performance means, where each ellipsis can be expanded with a step size of 0.05:

• all low performance (Lo): 0.0, . . . , 0.45.
• all moderate performance (Mo): 0.25, . . . , 0.70.
• all high performance (Hi): 0.50, . . . , 0.95.
• two extremes (LH): 0.0, . . . , 0.25 and 0.70, . . . , 0.95.
• one error component among high performance (1LNH): 0, 0.55, . . . , 0.95.
• one high performance among low performance (NL1H): 0, . . . , 0.40, 0.95.
We show the performance in bar plots in Figure 5.29. First, we find that the learning strategies can relatively easily find the optimal policy when the task performance means follow the distributions of all high performance (Hi) and two extremes (LH), and face more challenges when the task performance means are generally low (Lo). Figure 5.29 also suggests that we may consider to use different learning strategy for different distributions of task performance mean. Overall, FAi/CP-RU achieves the best performance with most distributions, and it also has much smaller variance than other learning strategies, especially in the case of one high performance among low performance (NL1H). For the case of all low performance (Lo), BEMi+/CP has the second best performance after FAi/CP-RU, and for the case of all moderate performance (Mo), BEMi*/CP and BEMi+/CP score the second and the third. Then, for the case of all high performance (Hi), BEMi*/CP has the highest performance, and BEMi+/CP and FAi/CP-RU each drop one place to the second and the third place respectively. For the case of one error component among high performance (1LNH), BEMi+/CP has the highest performance, and slightly outperforms FAi/CP-RU.

Finally, we vary the uncertainty of the task performance of each component, i.e. the task performance variance. In the previous experiments, we define the task performance variance as 0.01 universally, and now we test different task performance variances, including 0.00125 (more certain), 0.0025, 0.005, 0.01, 0.02, 0.04, and 0.08 (more uncertain), while fix the means as 0, . . . , 0.9, which can be derived from a set of symmetric beta distributions with \(a = b = 99.5, 49.5, 24.5, 12, 5.75, 2.625, 1.0625\) respectively. To better illustrate the distributions of the task performance with different variances, we plot the probability density function of each distribution with variance set as 0.01, 0.00125, or 0.08 respectively, in Figure 5.30.

We show the results in Figure 5.31, and we can see that most strategies remain at their (near-) highest performance even when the variance increases from 0.00125 to 0.005. However, when the variance keeps increasing to 0.08, the performance all start to drop, but the best-performing strategies remain the best among others.
In the previous sections, we consider the non-budgeted CSE problem, which holds a similar assumption as in conventional multi-armed bandit problem and reinforcement problem that the policy optimization process terminates only when it converges, and our goal is to find the optimal policy as quickly as possible. In the simulated experiment, we find that many policy optimization strategies can converge to a near-optimal solution only within 2000 episodes for a configuration space that has $10^{10}$ combinations. In reality, we may have to terminate the policy optimization process before it converges, due to a more complex configuration space, e.g. one containing more than $10^{10}$ combinations or a limited budget that does not allow to execute 2000 episodes. In this section, we study two aspects of the budgeted CSE problem – cost modeling for each execution and budgeted policy optimization strategies that consider the overall budget and the cost estimated for each execution.

### 5.6.1 Cost Modeling

In order to better allocate the budget, the budgeted policy optimization strategies need to be able to accurately predict the cost of executing a component, which is measured by its runtime performance (Section 5.3), including the cost of memory, storage, network, and often more importantly, time. Cost is not modeled in traditional budgeted multi-armed bandit (MAB) problems, since they assume that pulling an arm incurs a known and fixed cost. In our problem, the cost of executing a configured component on a certain input is unknown until the execution is done. Moreover, the cost is also a variable in a nonstationary environment. Ding et al. [50] study the problem of MAB with budget constraint and variable costs, where both rewards and costs need to be estimated during exploration, and they prove the theoretical bounds of the proposed greedy algorithm.
Given a MDP definition, we use the same set of techniques in modeling the value function to model the cost for each state and state-action pair. For example, if we use current phase (CP) for state representation in the CSE-MDP, and we apply a tabular value estimator, which uses the total reward as the target to update the value functions of the states and the state-action pairs in the episode, we can also use this tabular estimator to update the cost estimation of the each state and state-action pair after the runtime performance is reported immediately after the execution. If the state has a vector representation, then a linear estimator can be used instead.

The standard multi-armed bandit problem or the non-budgeted configuration space exploration problem orders the options by their values (i.e. rewards). When we further consider the cost, we can define alternative greedy heuristics other than the reward order, e.g. density (reward-cost ratio) order, for more efficient exploration.
5.6.2 Budgeted Policy Optimization

When we have an additional constraint on the total exploration budget, we may have two approaches: (1) applying the same procedure as for the non-budgeted multi-armed bandit problem with costs and using the budget only as a stopping criterion, where we can consider to use the reward order or the density order as the greedy heuristic, and (2) planning the exploration schedule with respect to the total budget and the remaining budget after each execution.

In fact, assuming that both the reward and the cost can be correctly predicted, the budgeted multi-armed bandit problem is degenerated to the budgeted policy optimization problem with known cost, which is essentially an NP-hard problem – unbounded knapsack problem and commonly solved greedily using the density order as the greedy heuristic. To balance the exploration and exploitation, a UCB-E strategy (Eq. 5.11) \[7\] is proposed to encourage exploration, i.e. selection of a greedy option is less frequent and requires the reward estimation to have a higher confidence than UCB1 (Eq. 5.10). This extension tolerates a poorer behavior policy, and in return, anticipates a better expected optimal policy during the entire exploration process.

The multi-armed bandit problem aims to maximize the expected total rewards during the entire exploration phase. The second approach is often studied in the best arm identification problem \[7\] 64 95 or pure exploration problem \[29\], which aims to find the best arm given the budget constraint, a setting more relevant to our budgeted configuration space exploration problem. In other words, a multi-armed bandit problem defines AUC-BP (equivalently the opposite of regret) or AUC-EOP as a performance metric for the policy optimization strategy, which counts every exploration episode, whereas best arm identification problem only cares the optimal policy in the last exploration episode. Successive Rejects \[7\] is an example algorithm making use of the budget while planning the exploration schedule, which divides the total budget into a number of successive
reject phases\textsuperscript{15} of unequal size, and eliminates the worse option at the end of each successive reject phase. It is tailored to the best arm identification setting, and neither optimizes the behavior performance (low AUC-BP) or the expected optimal policy (low AUC-EOP) until the last successive reject phase. We may expect this algorithm to have a learning curve (both AUC-BP and AUC-EOP) that slowly increases in the first several successive reject phases and then quickly reaches a (possibly local) optimum in the end.

We may apply these approaches to the budgeted configuration space exploration problem. The original Successive Rejects algorithm deals with the problem with fixed and equal cost (1 for each arm pulling), and defines a fixed exploration schedule (continuously selecting arm \( i \) for a certain rounds), which prevents effective exploration over combinations across phases in configuration space exploration problem. Therefore, we modify the Successive Rejects algorithm by setting \( n \) as the total budget, and \( K \) as the number of components in each phase, then we define that the total budget for each reject phase \( k \) is \((n_k - n_{k-1})(K - k + 1)\), and select an option randomly from the remaining options.

We see that exploration is greatly encouraged in the budgeted multi-armed bandit problem, and even more in the best arm identification problem. We recall that, in Section \[5.4.2\] the (budgeted) configuration space exploration problem, although formalized alike the best arm identification problem, has a multi-step nature as defined in MDP, which means an over-aggressive exploration that uses a behavior policy “too off” the estimation policy may cause problem in value estimation. Therefore, we adopt the same techniques proposed in Section \[5.4.2\] off-policy and two-step option selection, for configuration space exploration, and define additional two-step option selection strategies: random/UCB-E (RUE) and random/Successive Rejects (RJ). We define that we always use the greedy component at the phase that is not selected for exploration, meaning that if we use density as the greedy heuristic, we will always pick the highest-density component at most phases\textsuperscript{16}. When combining random phase selection with Successive Rejects algorithm for component selection, we select the greedy component at a greedy phase, and a random component at an exploratory phase. We eliminate one option from each phase after the budget for a successive reject phase is exhausted and the phase is being explored. If the ratio of the length of the pipeline to the total budget is large, there may exist phases that are not selected for exploration until a next successive reject phase, and as a result, the RJ exploration strategy may eliminate more than one options at a time.

In summary, a budgeted policy optimization strategy should consist of four modules: a state representation method, a value estimation method, an option selection method, and a greedy heuristic, where the non-budgeted policy optimization strategy always uses reward order as the greedy heuristic.

\textsuperscript{15} Audibert et al. \[7\] use the term phase to represent a series of exploration attempts without any option eliminated, which is referred to as successive reject phase in this work, to distinguish from our definition of phase in a multi-step information processing pipeline.

\textsuperscript{16} We may also compare with an alternative two-step option selection method: always picking the highest-reward/value options at greedy phases.
Table 5.14: Description of simulated problem set and configuration space

<table>
<thead>
<tr>
<th>Problem set</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>distribution</td>
</tr>
<tr>
<td></td>
<td>mean</td>
</tr>
<tr>
<td></td>
<td>variance</td>
</tr>
<tr>
<td>Evaluation</td>
<td>multiplication</td>
</tr>
<tr>
<td></td>
<td>of input and task</td>
</tr>
<tr>
<td></td>
<td>performances</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Configuration space</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Length of the pipeline</td>
<td>10</td>
</tr>
<tr>
<td># of components at phase $i$</td>
<td>10, for $i = 1, \ldots, 10$</td>
</tr>
<tr>
<td>Task performance of component $c_{i,j}$</td>
<td>distribution</td>
</tr>
<tr>
<td></td>
<td>mean</td>
</tr>
<tr>
<td></td>
<td>variance</td>
</tr>
<tr>
<td>Runtime performance of component $c_{i,j}$</td>
<td>distribution</td>
</tr>
<tr>
<td></td>
<td>mean</td>
</tr>
<tr>
<td></td>
<td>variance</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Budget</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>600, 1,800, 3,600, 7,200, 14,400, 28,800, 86,400, 172,800, 345,600</td>
<td></td>
</tr>
<tr>
<td>(i.e. 10m, 30m, 1h, 2h, 4h, 8h, 1d, 2d, 4d)</td>
<td></td>
</tr>
</tbody>
</table>

5.7 Simulated Experiments for BCSE

In this section, we report the experimental results using a simulated environment. We first describe the simulated environment that also defines runtime performance (cost) for each component. Then, we compare the learning performance of budgeted policy optimization strategies, including value-based methods and model-based methods with various option selection strategies, using original value function or density. Finally, we compare the learning performance in different environments by varying the simulated environments.

5.7.1 Simulated Environment

In this section, we use a simulated environment to compare budgeted configuration space exploration strategies, which uses the same problem set and the same configuration space as in Section 5.5. We also define the runtime performance for each component $c_{i,j}$. First, we shuffle the components in a phase and obtain a new index $l_j$ for each component, to make sure that there is no correlation between a component’s runtime performance and its task performance. Then, we define the runtime performance of the $l_j$-th component is drawn from a gamma distribution with parameter $k = 10$ and $\theta = \frac{1}{10} (\exp \frac{l_j}{2})$, i.e. mean at $\exp \frac{l_j}{2}$ and variance of $\frac{1}{10} (\exp \frac{l_j}{2})^2$. The component that has the lowest expected cost has a mean of 1 and variance of 0.1, and the one that has the
highest expected cost has a mean of 90.02 and variance of 810.31. If we interpret the cost value is
the total temporal duration of executing a component (including the original processing time plus
the overhead induced by the policy optimization strategy), then processing an input in every phase
costs the fastest component 1 second on average and the slowest component 1 minute 30 seconds
on average. We define nine different budgets from 600 to 345,600, which equal to 10 minutes to
4 days using a temporal duration interpretation. These budgets simulate the wait times a de-
veloper may afford for the integration test to complete. We summarize the settings for the simulated
environment in Table 5.14.

5.7.2 Experiment Settings

We repeat each experiment setting 10 times, and report the mean of the 10 runs. As discussed in
Section 5.6, we evaluate the performance of the budgeted configuration space exploration strategies
in terms of the final optimal policy at termination when the budget is exhausted, and thus, we report
the expected optimal performance, averaged across 10 runs, and the significance level over the
comparand using t-test for each strategy and budget. In addition, we also report the progress,
i.e. the number of training episodes collected when a certain budget is spent. We vary the budget
from 600 (10m) to 345,600 (4d), and observe the performance of budgeted policy optimization
strategy. The expected optimal performance curve is also the learning curve of a strategy that uses
the budget only as the stopping criterion (such as UCBE and RUE), since the exploration schedule
is identical for all budgets. It is not the learning curve of a strategy that considers the budget in
the exploration schedule, such as the Successive Rejects algorithm. Since we select the budgets
roughly following a geometric series, we use logarithmic scale for y-axis in both plots.

In this section, we use the current phase (CP) for state representation (for both value estimation
and cost modeling). For the value estimation module, we may use the tabular MC control strategy
(TQ), the linear MC control strategy (FA), and the additive model based learning with error pre-
diction (BEM+, or shorten as M+ in this subsection). We solve the linear regression problem (for
linear MC control and additive benefit model) incrementally using SGD. The baseline option se-
lection method is $\epsilon$-greedy (denoted by $\epsilon$G for on-policy learning and OFF for off-policy learning
with exponentially discounted counter), where the greedy option is defined as the option that has
the highest value (or reward, denoted by $/R$). A variant strategy can use the density to select op-
tions, which is denoted by $/D$. We can also consider other independent option selection strategies,
including UCB-E (UCBE), Successive Rejects (J), or two-step option selection strategies, includ-
ing random/random (RR), random/UCB1 (RUI), random/UCB-E (RUE), and random/Successive
Rejects (RJ). When we use any of the two-step option selection strategies, we use the off-policy
learning. We use the same values to the parameters (e.g. $\epsilon$, damping factor $\zeta$, initial padding length,
etc.) as in Section 5.5 to initialize the strategies. We set the scaling factor $\epsilon_{UCB}$ in Eqs. 5.10 and
5.11 as 10.

5.7.3 Comparison of Budgeted Policy Optimization Strategies

In this subsection, we compare the learning performance of budgeted learning strategies.
Independent option selection

We first compare the performance of the budgeted policy optimization strategies that use independent option selection methods for the simulated configuration space constrained by various budgets. We report the results in Table 5.15 and Figures 5.32 to 5.34.

We find that the performance of most budgeted policy optimization strategies is improved and the variance is also reduced if we increase the budget, because they can spend the extra budget to collect more training episodes for more accurate value estimation or regression. Also, we can see from the progress curves that if we use the density order (/D) instead of the reward order (/R) as the independent option selection methods for the simulated configuration space constrained by various budgets. We report the results in Table 5.15 and Figures 5.32 to 5.34.

Table 5.15: Results of budgeted learning strategies with independent option selection

<table>
<thead>
<tr>
<th>Strategy</th>
<th>10m</th>
<th>30m</th>
<th>1h</th>
<th>2h</th>
<th>4h</th>
<th>8h</th>
<th>1d</th>
<th>2d</th>
<th>4d</th>
<th>Comparand</th>
</tr>
</thead>
<tbody>
<tr>
<td>TQ-εG/R</td>
<td>0.004</td>
<td>0.021</td>
<td>0.026</td>
<td>0.079</td>
<td>0.164</td>
<td>0.180</td>
<td>0.090</td>
<td>0.244</td>
<td>0.611</td>
<td>-</td>
</tr>
<tr>
<td>TQ-εG/D</td>
<td>0.029</td>
<td>0.091</td>
<td>0.010</td>
<td>0.017</td>
<td>0.071</td>
<td>0.200</td>
<td>0.092</td>
<td>0.492</td>
<td>0.631</td>
<td>0.632</td>
</tr>
<tr>
<td>TQ-OFF/R</td>
<td>0.011</td>
<td>0.042</td>
<td>0.006</td>
<td>0.013</td>
<td>0.047</td>
<td>0.127</td>
<td>0.415</td>
<td>0.649</td>
<td>0.667</td>
<td>0.015</td>
</tr>
<tr>
<td>TQ-OFF/D</td>
<td>0.025</td>
<td>0.012</td>
<td>0.016</td>
<td>0.028</td>
<td>0.050</td>
<td>0.054</td>
<td>0.171</td>
<td>0.198</td>
<td>0.179</td>
<td>0.025</td>
</tr>
<tr>
<td>TQ-UCBE/R</td>
<td>0.010</td>
<td>0.009</td>
<td>0.018</td>
<td>0.029</td>
<td>0.048</td>
<td>0.150</td>
<td>0.407</td>
<td>0.407</td>
<td>0.344</td>
<td>0.005</td>
</tr>
<tr>
<td>TQ-UCBE/D</td>
<td>0.025</td>
<td>0.007</td>
<td>0.035</td>
<td>0.053</td>
<td>0.116</td>
<td>0.242</td>
<td>0.328</td>
<td>0.440</td>
<td>0.411</td>
<td>0.015</td>
</tr>
<tr>
<td>TQ-J</td>
<td>0.017</td>
<td>0.010</td>
<td>0.034</td>
<td>0.062</td>
<td>0.073</td>
<td>0.266</td>
<td>0.576</td>
<td>0.576</td>
<td>0.639</td>
<td>0.015</td>
</tr>
<tr>
<td>FA-εG/R</td>
<td>0.026</td>
<td>0.011</td>
<td>0.026*</td>
<td>0.044</td>
<td>0.085*</td>
<td>0.143</td>
<td>0.231</td>
<td>0.306</td>
<td>0.366</td>
<td>0.025</td>
</tr>
<tr>
<td>FA-εG/D</td>
<td>0.026</td>
<td>0.009</td>
<td>0.030</td>
<td>0.069</td>
<td>0.158</td>
<td>0.258</td>
<td>0.317</td>
<td>0.424</td>
<td>0.420</td>
<td>0.005</td>
</tr>
<tr>
<td>FA-OFF/R</td>
<td>0.022</td>
<td>0.011</td>
<td>0.016</td>
<td>0.028</td>
<td>0.043</td>
<td>0.093</td>
<td>0.231</td>
<td>0.275</td>
<td>0.282</td>
<td>0.005</td>
</tr>
<tr>
<td>FA-OFF/D</td>
<td>0.026</td>
<td>0.007</td>
<td>0.005</td>
<td>0.045</td>
<td>0.114</td>
<td>0.015</td>
<td>0.021</td>
<td>0.067</td>
<td>0.004</td>
<td>0.005</td>
</tr>
<tr>
<td>FA-UCBE/R</td>
<td>0.011</td>
<td>0.016</td>
<td>0.029</td>
<td>0.019</td>
<td>0.038</td>
<td>0.049</td>
<td>0.075</td>
<td>0.115</td>
<td>0.110</td>
<td>0.005</td>
</tr>
<tr>
<td>FA-UCBE/D</td>
<td>0.016</td>
<td>0.003</td>
<td>0.005</td>
<td>0.007</td>
<td>0.013</td>
<td>0.017</td>
<td>0.028</td>
<td>0.013</td>
<td>0.022</td>
<td>0.005</td>
</tr>
<tr>
<td>FA-J</td>
<td>0.003</td>
<td>0.003</td>
<td>0.017</td>
<td>0.046</td>
<td>0.124</td>
<td>0.170</td>
<td>0.306</td>
<td>0.431</td>
<td>0.363</td>
<td>0.005</td>
</tr>
<tr>
<td>M+εG/R</td>
<td>0.001</td>
<td>0.008</td>
<td>0.015</td>
<td>0.012</td>
<td>0.018</td>
<td>0.052</td>
<td>0.240</td>
<td>0.456</td>
<td>0.592</td>
<td>0.005</td>
</tr>
<tr>
<td>M+εG/D</td>
<td>0.003</td>
<td>0.008</td>
<td>0.018</td>
<td>0.014</td>
<td>0.104</td>
<td>0.284</td>
<td>0.451</td>
<td>0.508</td>
<td>0.538</td>
<td>0.015</td>
</tr>
<tr>
<td>M+UCBE/R</td>
<td>0.036</td>
<td>0.003</td>
<td>0.005</td>
<td>0.007</td>
<td>0.030</td>
<td>0.074</td>
<td>0.422</td>
<td>0.545</td>
<td>0.662</td>
<td>0.025</td>
</tr>
<tr>
<td>M+UCBE/D</td>
<td>0.016</td>
<td>0.006</td>
<td>0.044</td>
<td>0.027</td>
<td>0.167</td>
<td>0.308</td>
<td>0.455</td>
<td>0.380</td>
<td>0.391</td>
<td>0.015</td>
</tr>
<tr>
<td>M+J</td>
<td>0.017</td>
<td>0.004</td>
<td>0.009</td>
<td>0.032</td>
<td>0.042</td>
<td>0.155</td>
<td>0.201</td>
<td>0.271</td>
<td>0.329</td>
<td>0.015</td>
</tr>
</tbody>
</table>

Independent option selection

We first compare the performance of the budgeted policy optimization strategies that use independent option selection methods for the simulated configuration space constrained by various budgets. We report the results in Table 5.15 and Figures 5.32 to 5.34.

We find that the performance of most budgeted policy optimization strategies is improved and the variance is also reduced if we increase the budget, because they can spend the extra budget to collect more training episodes for more accurate value estimation or regression. Also, we can see from the progress curves that if we use the density order (/D) instead of the reward order (/R) as the greedy heuristic, the same budgeted policy optimization strategies, including the TQ-εG, FA-εG, M+εG, TQ-UCBE, and M+UCBE, are able to collect more episodes given the same budget, and can thus greatly or even significantly († to ‡, meaning a confidence level of 0.95 to 0.999 and above) improve their performance, especially in the early exploration stage (2h to 2d). In fact, based on the results in Table 5.15, the density ordered heuristic has significantly outperformed the reward ordered heuristic in 16 out of 15 measurements (64%) if the budget is between 2h to 2d, compared to 3 out of 15 measurements (20%) if the budget is equal to or below 1h and 0 out of 5 (0%) if the budget is 4d. Once the reward ordered heuristic discovers a policy whose expected performance is above 0.6, it can hardly be outperformed by a density ordered heuristic. This moti-
vates us to consider an $\epsilon$-first algorithm [196], which uses a portion of the budget (equal to $\epsilon$) for an aggressive or even a pure exploration, e.g. by using a density ordered heuristic, and then switches to a more conservative exploration, e.g. by using a conventional $\epsilon$-greedy strategy to exhaust the remaining budget. When the off-policy learning is used with density ordered heuristic, TQ-OFF/D and FA-OFF/D perform much worse than TQ-OFF/R and FA-OFF/R, because the option that has the highest density is considered greedy by the $\epsilon$-greedy option selection method but considered exploratory by the value estimator, which in return assigns a small learning rate. Successive Rejects algorithm (-J) is often optimal or close to optimal, especially when used with a TQ value estimator.
Two-step option selection

Next, we compare the performance of the budgeted policy optimization strategies that use two-step option selection (random/) for the simulated configuration space constrained by various budgets, which include RR, RU1, RUE, and RJ. We report the results in Table 5.16 and Figures 5.35 to 5.37.

First, we can hardly notice the difference of the number of training episodes collected during the exploration process between the reward ordered heuristic and the density ordered heuristic, because all phases but one select the greedy components regardless of the greedy heuristic and only the exploratory phase needs to prioritize the options according to the specified greedy heuristic, either density or reward ordered. Due to the same reason, if a two-step option selection method is
an interesting future work, one can consider to predict the sufficiency of budget 
the exploration starts.

17

17

This work, the sufficiency of budget is always analyzed 
ex post facto based on the performance of strategies. As 
an interesting future work, one can consider to predict the sufficiency of budget a priori before or immediately after 
the exploration starts.

| Table 5.16: Results of budgeted learning strategies with 2-step option selection |
|----------------------|--------|-------|------|------|------|------|------|------|----------------------|
| Strategy             | 10m    | 30m   | 1h   | 2h   | 4h   | 8h   | 1d   | 2d   | 4d       | Comparand  |
|----------------------|--------|-------|------|------|------|------|------|------|----------------------|
| TQ-RR/R              | .006^ns | .002^ns | .005ns | .003ns | .007ns | .006ns | .017** | .039** | .5655*** | .7161†       | TQ-UCBE/R  |
| TQ-RR/D              | .005^ns | .009^ns | .012ns | .024ns | .130^ns | .245ns | .418^ns | .546^ns | .667^ns | TQ-UCBE/D   |            |
| TQ-RU1/R             | .01^ns  | .01^ns  | .015ns | .032ns | .105^†  | .187^ns | .361^ns | .357^ns | .158^†  | TQ-UCBE/R   |            |
| TQ-RU1/D             | .0057^ns | .017^ns | .023ns | .061^ns | .121ns  | .30^ns  | .349ns  | .260^†  | .205^†  | TQ-UCBE/D   |            |
| TQ-RUE/R             | .0017^ns | .010^ns | .036^ns | .060^ns | .123^†  | .212ns  | .494^ns | .432^ns | .275^ns | TQ-UCBE/R   |            |
| TQ-RUE/D             | .0004^* | .0019^* | .008^†  | .027^ns | .058^ns | .168ns  | .462^ns | .426^ns | .151^ns | TQ-UCBE/D   |            |
| TQ-RJ                | .0003^ns | .0021^+ | .013ns  | .042ns  | .043^ns | .204^ns | .495^ns | .563^ns | .696^ns | TQ-J        |            |
| FA-RR/R              | .009^ns | .002^ns | .005^ns | .023^ns | .083^ns | .20^ns  | .593^ns | .641^ns | .641^ns | FA-UCBE/R   |            |
| FA-RR/D              | .0023^ns | .004^ns | .006^*  | .018^ns | .052^ns | .152^ns | .397^ns | .570^ns | .616^ns | FA-UCBE/D   |            |
| FA-RU1/R             | .0029^ns | .006^ns | .020^ns | .0750^ns | .149^†  | .226^ns | .390^ns | .390^ns | .367^ns | FA-UCBE/R   |            |
| FA-RU1/D             | .0021^ns | .006^ns | .029^ns | .063^ns | .1532^ns | .278^ns | .383^ns | .477^ns | .431^ns | FA-UCBE/D   |            |
| FA-RUE/R             | .0009^ns | .003^ns | .014^ns | .046^ns | .129^ns | .20^ns  | .445^ns | .448^ns | .415^ns | FA-UCBE/R   |            |
| FA-RUE/D             | .0208^ns | .0203^ns | .021^ns | .025^ns | .107^*  | .250^ns | .454^ns | .454^ns | .422^ns | FA-UCBE/D   |            |
| FA-RJ                | .0103^ns | .0052^ns | .015^ns | .010^ns | .102^ns | .153^ns | .359^ns | .722^ns | .720^ns | FA-J        |            |
| M+-RR/R              | .006^ns | .0013^ns | .008^ns | .015^ns | .040^ns | .188^*  | .415^*  | .608^†  | .750^ns | M+-UCBE/R   |            |
| M+-RR/D              | .0008^ns | .001^ns | .0033^ns | .040^ns | .074^ns | .293^*  | .541^ns | .758^ns | M+-UCBE/D |            |
| M+-RJ                | .0007^ns | .0041^ns | .012^ns | .024^ns | .066^ns | .170^*  | .488^ns | .667^ns | .710^ns | M+-UCBE/R   |            |
| M+-RJ/D              | .0014^ns | .0023^ns | .003^ns | .042^ns | .163^ns | .406^ns | .725^ns | .750^ns | .750^ns | M+-UCBE/D   |            |
| M+-RUE/R             | .0015^ns | .0116^ns | .0209^ns | .030^†  | .062^ns | .126^ns | .545^ns | .734^†  | .694^ns | M+-UCBE/R   |            |
| M+-RUE/D             | .0053^ns | .0071^ns | .0104^ns | .018^ns | .042^ns | .097^ns | .425^ns | .638^ns | .795^ns | M+-UCBE/D   |            |
| M+-RJ                | .0023^ns | .002^ns | .0072^ns | .007^†  | .035^ns | .129^ns | .358^ns | .478^ns | .771^ns | M+-J        |            |

used, there is no consistently significant difference between the performance of the reward ordered 
heuristic and that of the density ordered heuristic.

Now, we compare between the two-step option selection methods. If the budget is sufficient 
(> 1d† 17) we find the RR method performs better than RU1 and RUE, no matter which value 
estimator is used, and the RUE method often outperforms the RU1 method. In fact, as the budget 
becomes more sufficient and the number of episodes increases, the RU1 and RUE methods become 
less exploratory (with a decayed exploration rate) and stick with the highest-density component 
for an exploratory phase, whereas the RR method keeps exploring options at the same constant 
rate throughout the entire exploration process. If the budget is insufficient (1h – 8h), then the 
RU1 method is more exploratory than the RR method, and thus tends to perform better. The RJ 
method is again very reliable, which often achieves the highest performance or close to the highest 
performance, especially when the budget is relative sufficient.

17In this work, the sufficiency of budget is always analyzed ex post facto based on the performance of strategies. As 
an interesting future work, one can consider to predict the sufficiency of budget a priori before or immediately after 
the exploration starts.
the two-step option selection methods can many times improve the performance of the budgeted policy optimization strategies when the budget is sufficient. An interesting next step is to study a variant of the two-step option selection method that always selects the highest-reward component in the greedy phases.

Scaling factor of UCB

We have defined a scaling factor $c_{\text{UCB}}$ to the estimated value term in both Eqs. 5.10 and 5.11, which controls the decay rate. Intuitively, a smaller $c_{\text{UCB}}$ allows the exploration process to remain for a longer period of time, while a larger $c_{\text{UCB}}$ selects a greedy option more frequently. We vary the
scaling factor from 0.001 to 1000, while fix the budget to 86,400 (1d), and plot the performance of the budgeted policy optimization strategies that use UCB-E, random/UCB1, and random/UCB-E for option selection in Figures 5.38, 5.39, and 5.40 respectively.

We can see from the progress curves in Figure 5.38, when the independent option selection method UCB-E is used and the scaling factor $c_{UCB}$ is set below 1, all the budgeted policy optimization strategies collect roughly the same number of training episodes (slightly above 500), although their performance differs. When the scaling factor $c_{UCB}$ increases from 1 to 1,000, the progress curves diverge: the strategies that use the density ordered heuristic collect more episodes (up to 2,000) than when $c_{UCB}$ is set below 1, and the strategies that use the reward ordered heuristic collect fewer episodes (below 500). However, their performance curves follow the same pattern. The
performance first gets improved when \( c_{UCB} \) increases to 10 or 100, then drops when \( c_{UCB} \) continues to increase to 100 or 1000. When the two-step open selection methods RU1 and RUE are used, the number of episodes collected for training slightly increases when \( c_{UCB} \) increases. However, we can hardly make any conclusion about the impact of \( c_{UCB} \) on their performance due to large variance.

**Summary**

When the budget is insufficient, one may consider an aggressive exploration method and the density ordered heuristic, e.g. UCBE/D, \( \epsilon \)G/D, or RR/D. When the budget is relatively sufficient, one may use a conservative exploration method, which includes (1) a two-step option selection method, (2) a reward ordered heuristic, i.e. simply a non-budgeted policy optimization strategy (e.g. RR/R or RJ), (3) a small (effective) learning rate (our TQ-RU strategies are counter-examples).

### 5.7.4 Comparison of Learning Performance in Different Environments

In this subsection, we vary the environments: the runtime performance of the components and the correlation between the component’s runtime performance and the task performance, and compare the learning curves of the budgeted policy optimization strategies.

**Runtime performance of the components**

We use the gamma distribution, parameterized by \( k = 10 \) and \( \theta = \frac{1}{10}(\exp \frac{t}{2}) \), to simulate the runtime performance of each component. We now add a scaling factor \( c_{RPM} \) to \( \theta \), which defines a gamma distribution with the mean at \( c_{RPM} \exp \frac{t}{2} \) and the variance of \( \frac{1}{10}(c_{RPM} \exp \frac{t}{2})^2 \). Since we fix the parameter \( k \), the gamma distributions with different \( c_{RPM} \) values have the same shape. We plot in Figure [5.41] the probability density functions of gamma distributions with \( c_{RPM} = 0.25, 1, \) and

![Figure 5.40: Varying scaling factor \( c_{UCB} \) for strategies w/ random/UCB-E](image)
4, which represent the components’ runtime performance in a phase. When \( c_{\text{RPM}} = 0.25 \), we see that the expected cost of a component is smaller than \( c_{\text{RPM}} = 1 \), where the highest-cost component has an expected cost of 22.5, and thus we expect more episodes can be collected given the same budget. In contrast, when \( c_{\text{RPM}} = 4 \), the highest-cost component has an expected cost of 360.1. We vary \( c_{\text{RPM}} \) from 0.25 to 4, fix the budget at 345,600 (1d), and compare the performance of budgeted policy optimization strategies. The comparison results of these strategies are reported in Figures 5.42 to 5.47, grouped by their value estimation method (TQ, FA, or M+) and option selection method (independent option selection or two-step option selection).

For the strategies that use an independent option selection method, we see from Figures 5.42 to 5.44 that, when we increase \( c_{\text{RPM}} \) (i.e. the mean execution cost), all strategies tend to collect fewer training episodes, and the expected optimal performance of most strategies drops. We can see that the performance drops at a constant rate, and those that use the density ordered heuristic tend to drop more slowly (smaller decline rate) than that use the reward ordered heuristic. For example, we can find the TQ-\( \epsilon \)G/R curve and TQ-\( \epsilon \)G/D curve intersect between \( c_{\text{RPM}} = 0.5 \) and 1,
Figure 5.43: Performance of FA strategies w/ independent option selection when $c_{RPM}$ varies

Figure 5.44: Performance of M+ strategies w/ independent option selection when $c_{RPM}$ varies

FA-$\epsilon$G/R curve and FA-$\epsilon$G/D curve intersect between $c_{RPM} = 1$ and 2, M+-$\epsilon$G/R curve and M+-$\epsilon$G/D curve intersect between $c_{RPM} = 0.5$ and 1, which again suggests that we may consider to use the density ordered heuristic when the budget is insufficient relative to the cost of the components in the configuration space, and vice versa.

For the strategies that use a two-step option selection method and tabular MC control (TQ) for value estimation, we see from Figures 5.45 that, as we decrease $c_{RPM}$ from 4 to 0.25, all the strategies are able to collect more episodes, especially those that use the random/random (RR) option selection method. In fact, the RJ method also uses RR for option selection despite that it also eliminates options. We can also see that the performance of the strategies that use the random/UCB option selection methods (including RU1 and RUE) may start to drop as $c_{RPM}$ decreases from 1,
which becomes more noticeable when $c_{RPM}$ decreases to 0.25. We find that this phenomenon is the consequence of the convergence problem of the off-policy tabular MC control strategies that we discussed in Section 5.5 (for Figure 5.6), which happens when the strategy uses a large effective learning rate, also employs an aggressive exploration method, and experiences sufficiently large number of episodes. TQ-RJ is the best budgeted policy optimization strategy when the average cost of a component is low (0.25) and sufficient episodes can be collected, and the worst when the average cost is high (4), which suggests that it could be risky to use the TQ-RJ method since a true optimal option may be mistakenly eliminated due to insufficient budget and thus unreliable value estimation.

The phenomenon that a budgeted policy optimization strategy using a random-UCB method
and the density ordered heuristic is less noticeable when we use FA for value estimation, as shown in Figure 5.46. However, we can still see the performance of FA-RU1/D and FA-RUE/D drops at $c_{RPM} = 0.25$. In general, as the $c_{RPM}$ decreases from 4 to 0.5, the performance of the budgeted policy optimization strategies using FA increases. When we use the additive model based estimator (M+), we find from Figure 5.47 that the performance continuously increases as $c_{RPM}$ decreases, even when $c_{RPM} = 0.25$, which makes M+ the best budgeted policy optimization strategy family when the average cost of component is relatively low. From both Figures 5.46 and 5.47, we can also see that the FA-RJ and M+-RJ strategies are optimal when $c_{RPM}$ is low (0.25), and become worse when $c_{RPM}$ is high (4) among all the strategies.

**Correlation between task performance and runtime performance**

The simulated configuration space defines a pair of distributions for each component to represent its task performance and runtime performance respectively, and there is no correlation between them, i.e. a high-quality component may also run faster than a low-quality component in some phase, but a high-quality component may run more slowly in another phase. We are interested to observe whether the correlation has some impact in the performance of the budgeted policy optimization strategies, especially those that use a density ordered heuristic.

We use the Spearman’s rank correlation coefficient to measure the correlation between the task performance mean (TPM) and the runtime performance mean (RPM) of all the components in a phase. The performance correlation $\rho_{TRM,RPM}$ is a number between $-1$ and 1. When $\rho_{TRM,RPM} = -1$, a higher-quality component always performs faster; when $\rho_{TRM,RPM} = 1$, a higher-quality component always performs more slowly; when $\rho_{TRM,RPM} = 0$, there is no relation between the runtime performance and the task performance. We divide the range into 10 subranges of equal length: $[-1.0, -0.8], \ldots, [0.8, 1.0]$, and accordingly create 10 simulated configuration spaces, where each configuration space contains only phases whose performance correlation is within the same sub-

![Figure 5.47: Performance of M+ strategies w/ two-step option selection when $c_{RPM}$ varies](image)
range. We use the centers of each subrange $-0.9, \ldots, 0.9$ to represent the corresponding simulated configuration space, and plot the performance of the budgeted policy optimization strategies when $\rho_{\text{TRM, RPM}}$ varies in Figures 5.48 to 5.53.

First, we can see from Figures 5.48, 5.49, and 5.50 that almost all the budgeted policy optimization strategies are able to collect more training episodes as $\rho_{\text{TRM, RPM}}$ decreases from 0.9 to $-0.9$. In fact, when $\rho_{\text{TRM, RPM}}$ is negative and small, better-quality components, which are more likely selected in the greedy phases, tend to cost less, and therefore, more episodes can be collected given the same budget. The performance of most top budgeted policy optimization strategies also increases as $\rho_{\text{TRM, RPM}}$ decreases from 0.9 to $-0.9$, including TQ-OFF/R, TQ-J, FA-$\epsilon$G/R, FA-$\epsilon$G/D, FA-OFF/R, FA-J, M+$\epsilon$G/R, M+$\epsilon$G/D, and M+-UCBE/R, all of which benefit from additional
episodes collected during the exploration process. On the contrary, TQ-\(\epsilon\)G/D cannot collect more episodes, as \(\rho_{\text{TRM, RPM}}\) further decreases from −0.3 to −0.9, and thus, its performance also starts to drop when \(\rho_{\text{TRM, RPM}}\) is below −0.5. We can also see that collecting more episodes does not necessarily improve the performance. For example, the OFF/D strategies (using \(\epsilon\)-greedy to select option, off-policy learning to update the model, and density ordered heuristic) can collect more episodes, however whose performance is consistently low. We can again observe a learning curve of a strategy using the density ordered heuristic intersects with another learning curve of a strategy that uses all the same modules but the reward ordered heuristic, e.g. FA-\(\epsilon\)G/D vs FA-\(\epsilon\)G/R and M+-\(\epsilon\)G/D vs M+-\(\epsilon\)G/R, where the ones that use the density ordered heuristic tend to have a faster decline rate, indicating that they are more sensitive to the correlation between the runtime performance and the task performance.

We can see from Figures 5.51, 5.52, and 5.53 that all the strategies collect more episodes as \(\rho_{\text{TRM, RPM}}\) increases from 0.9 to −0.9, and the performance of most strategies increases as \(\rho_{\text{TRM, RPM}}\) decreases from 0.9 to −0.9. The exception is again the budgeted policy optimization strategies using the TQ method for value estimation and using random/UCB for option selection, whose performance starts to drop as \(\rho_{\text{TRM, RPM}}\) decreases from −0.1 to −0.9. Similar to the phenomena we observe in Figures 5.6 and 5.45, this is again a consequence of the convergence problem of the off-policy tabular MC control.

### 5.8 Generalization to Analytics Space Exploration

The configuration space exploration strategies discussed in this chapter can be easily extended to full analytics space exploration, which further considers nonlinear decision processes. In fact, we can define an analytics engine evaluation MDP and an analytics space exploration MDP, analogous to the configuration evaluation MDP and configuration space evaluation MDP. Once we define the
MDPs, we can apply the same (budgeted) policy optimization strategies to the MDP to find the optimal information system.

### 5.8.1 Analytics Engine Evaluation MDP & Analytics Space Exploration MDP

When we consider the decision processes, we have defined two functions – problem reduction \( \rho \) and solution synthesis \( \sigma \) in Section 2.2. Similarly, we introduce the entry state \( \langle \rho \rangle \) and the exit state \( \langle \sigma \rangle \) for each decision process corresponding to the problem reduction process and solution synthesis process.

Given any arbitrary decision process with predefined decision factors, we may fix the execution
order of all the factors (e.g. in the alphabetical order of their names or identifiers), and then the execution trace of an analytics engine is unique, which describes the transition function of the analytics engine evaluation MDP. Compared with the configuration evaluation MDP defined in Section 5.3, the analytics engine evaluation MDP additionally defines (1) a fixed transition from the problem reduction state of a decision factor to the state of its first sub decision factor, (2) a fixed transition from the solution synthesis state of a sub decision factor or the last analysis step to the problem deduction state of the next sub decision factor or the first analysis step except for the last one, (3) a fixed transition from the solution synthesis state of the last sub decision factor to the solution synthesis state of its parent decision factor. Therefore, only a single deterministic policy can be derived from the analytics engine evaluation MDP.

Analogously, if we have multiple analytics procedures and further multiple analytics engines implementing these procedures, we may define the analytics space evaluation MDP, where we further allow the previously defined transitions from/to a problem reduction state or solution synthesis state (i.e. the transition types 1 and 3 in the previous paragraph) to a set of possible problem reduction states or solution synthesis states. In particular, when we arrive at a problem reduction state, we need to select a reduction policy from the set of decision processes that can solve this analytics task, which defines the set of valid actions for the problem reduction state. Once we take the action, we arrive at the state corresponding to the first decision factor. Similarly, when we arrive at a solution synthesis state, we need to select a solution synthesis policy from the set of solution synthesis models, which defines the set of valid actions for the solution synthesis state. A policy on analytics space evaluation MDP defines how to select decision process and solution synthesis model at each state.
5.8.2 Policy Optimization & Space Representation for Analytics Space Evaluation MDP

We may use the same iterative update methods described in Sections 5.4 and 5.6 to learn the optimal policy, where we randomly sample the decision process and solution synthesis model at each problem reduction state and solution synthesis state in the same way as we sample the components at each analysis step. We may also use the same value estimation method to learn the value function defined on each state or state-action pair. However, since the intrinsic mechanism of how a final decision is made is more complicated than a single output of a pipeline, we can hardly define a universal model-based method as we have done with the additive model and the multiplicative model in Section 5.4.4. In Chapters 8 and 9, we focus on evaluating the value-based methods including tabular MC control and linear MC control.

In Section 5.3.4, we propose to extract two types of information from each intermediate object to represent the state in the configuration evaluation MDP or the configuration space exploration MDP, namely meta information such as current phase (CP), current component (CC), and partial trace (PT), and intermediate content features. In an analytics space, since a sub analysis task (e.g. document retrieval) can be used by multiple decision factors to address different aspects of the same analytics problem, an analysis step can only be uniquely identified by both the factor and the analysis step. In other words, we must introduce an additional meta information type: current factor (CF) to distinguish two analysis steps in different sub analytics engines of the system that share the same phase number (with respect to each sub analytics engine), and/or use the same analysis component, and/or have the same partial execution trace. This allows the analytics meta learning framework to learn multiple values for the same analysis component that has different performance when integrated at different decision processes, and discover the best configuration (component and its parameter setting) customized to each factor. For example, a concept identification algorithm may work the best for some factor description but not for others, or some condition is considered as being met only if we can find more evidence (higher document frequency) than other conditions. However, it is not always necessary to distinguish the same analysis steps in two different factors. For example, a parser that exhibits a low performance on processing natural language descriptions for some decision factors tends to perform poorly on the descriptions of other decision factors that share a similar writing style.

5.9 Conclusion

In this section, we study the exploration strategies based on MDP modeling and policy optimization. Specifically, we model the configuration space exploration process as an MDP (referred to as the CSE-MDP), and describe how to use the policy optimization methods such as value-based methods and policy-based methods for configuration space exploration, and introduce a number of extensions, including off-policy learning methods for nonstationary environments and two model-based methods: additive model and multiplicative model. We also consider the budgeted policy optimization problem, where we discuss how to model the cost of each execution. We report the experimental results using simulated environments, where we compare the learning performance
of the various policy optimization strategies and budgeted policy optimization strategies, and then, we vary the environment to observe the impact of different problem sets and configuration spaces on the performance of these strategies.

There can be many extensions to the work we present in this chapter. For example, this work may be extended to consider continuous MDP, which can incorporate real-valued parameters directly without discretization. We can use any policy optimization method that works with vector representations of states and state-actions in continuous MDPs, such as the linear MC control method. Furthermore, we can also study how to represent the input and the intermediate object as feature vectors, and eventually automatically learn the representation for these objects. We are also interested in further extending our proposed strategies. For example, we may modify the reward function such that it can also generate negative signals for policy-based strategies, and study a variant of the two-step option selection method that always selects the highest-reward component in the greedy phases.
Chapter 6

Analytics Meta Learning Implementation

In Section 2.4, we describe the scenarios when a user may consider using the analytics meta learning when building his/her own intelligent information system, where we discuss the tasks for a framework developer and a component developer respectively.

This chapter focuses on the implementation detail of the intelligent architecture layer that assists developers in building information systems using analytics meta learning methodology. In Section 6.1, we discuss the high-level design and engineering principles of the architecture layer, which are not specific to any particular architecture framework such as UIMA and should be applicable to any in-house architecture framework. We then in Sections 6.2 and 6.3, we describe our implementation of the ECD (Extended Configuration Description) framework and the AML framework extension based on Apache UIMA and uimaFIT\(^1\), which focus on the analytics space specification and optimization / exploration respectively. We conclude in Section 6.4.

6.1 Design Principles

To facilitate easy integration with existing components, we require a system that is simple to configure and powerful enough to sift through thousands to trillions of option combinations to determine which represent the best system configuration. In this section, we present the general principles in implementing such architecture frameworks for in-house system development, which include the mandatory modules: **declarative descriptors**, **configurable evaluation**, and **configurable exploration strategy**, as well as optional but recommended modules: **automatic data persistence**, **global resource caching**, **distributed architecture**, and **graphical user interface**.

**Declarative descriptors.** To leverage the analytics meta learning framework, users specify how the analysis components should be organized into an information system, which parameter values may be sensible to the problem and thus need to be specified for each configuration, what is the input benchmark set, and what measurements should be applied. As we can see from the evolution of cognitive systems in Section 2.1, only if we can describe the elements effectively, can an algorithm be applied to manipulate and reason about them.

\(^1\)https://uima.apache.org/uimafit.html
Compared to mixing the high-level processing workflow and the low-level analysis logics in the same source code, e.g. programmatically itemizing the steps of an information pipeline in a `main()` method or via uimaFIT, and conveniently configuring the components via Java annotations, we recommend separate workflow and configuration representations, such as XML or YAML based process modeling languages. It is because such representation language is generally independent of implementation details (e.g. the programming language, the system requirement, etc.), and it can be more easily accessed and interpreted by the architecture framework.

**Configurable evaluation.** Analytics meta learning supports the evaluation of component performance based on user-specified evaluation metrics and gold-standard outputs at each step; these measurements are also used to estimate the value for each component and sub process. Analytics meta learning also requires the capability to calculate statistical significance of the performance difference between systems for a given task. This is important when attempting to disprove the null hypothesis that the best baseline system is as good as alternative systems, and crucial to understanding each component’s contribution (e.g. Section 4.2.4) and prioritizing the candidate systems when adapting to new tasks (e.g. Section 4.2).

Ideally, the framework should provide a set of commonly used evaluation metrics, such as precision, recall, F-measure. In addition, the framework should also provide APIs to allow users to extend the built-in evaluation metrics to meet various specific needs, which usually include two abstract methods that respectively evaluate for each instance and aggregate across the entire development set or the most recent $K$ test instances. For example,

```java
Map<Measure, Double> calculate(
    Collection<T> resultEvaluatees, Collection<T> gsEvaluatees);
```

```java
Map<Measure, Double> aggregate(
    Map<Measure, ? extends Collection<Double>> measure2values);
```

**Configurable exploration strategy.** As we have seen in Sections 5.7 and 5.7 and we will see in Chapters 7 to 9, each exploration strategy or budgeted exploration strategy has pros and cons. An experienced integration tester may pick one based on his/her observed or estimated characteristics of the problem, the analysis components, and/or the budget. Therefore, the exploration strategies should be configured by the user and extended by the integration tester; strategies described in Chapter 5 should be provided.

**Automatic data persistence.** In Sections 4.1 and 5.3, we have mentioned that if the intermediate results are kept in a repository accessible from the system, i.e. the persistenced configuration space exploration MDP, then the framework can avoid repeating executions for the components along the prefix, and also has more freedom to select which intermediate object and component to execute. In fact, a data persistence layer also allows the system to pause, resume and continuously integrate new modules during the exploration process, and is useful for error analysis, system performance analysis, and reproduction of previous experimental results.

To achieve automatic data persistence, the framework should accurately and efficiently seri-
alize arbitrary data objects output from analysis components and deserialize from the storage before sending them to the next analysis component, and the storage should be able to quickly locate the Binary Large OBject (BLOB) from the key pair of input and the trace prefix. Since we expect this process to be transparent to the module developers, i.e. they do not need to add serialization and deserialization code to each component analysis, we recommend to utilize existing serialization frameworks such as Apache Thrift\footnote{http://thrift.apache.org/} or data processing architecture frameworks that support data serialization.

**Global resource caching.** Online resources are sometimes temporarily unavailable, and are occasionally updated, changing their contents; to support reproducibility of results, one can implement a generic resource caching strategy inside the analytics meta learning framework. In addition to ensuring the reproducibility of results when external KBs are queried, global resource caching can also speed up the exploration process by avoiding replicated service calls across systems. The cost modeling methods described in Section\footnote{https://github.com/oaqa/uima-ecd} are designed for nonstationary environments, meaning that the framework is able to accurately estimate the actual runtime performance of a component with a caching layer.

**Distributed architecture.** In Chapters\footnote{4} and \footnote{5} we focus on the discussion of single-machine solutions to the analytics meta learning problem, i.e. only one configured component is processing one input instance at any time. In fact, we can also make the exploration process run in parallel, where the configured components are deployed into the cluster beforehand; the execution, fault tolerance and bookkeeping are managed by a master server.

**Graphical user interface.** A Web GUI could help the developers easily observe the current status of the exploration process and the performance of the produced systems. We recommend three different use cases for the Web GUI: *experiment inspection*, *ad hoc problem solving*, and *trend analysis*.

- The *experiment inspection* function helps users monitor configuration exploration progress and evaluation results, and view current and past persisted intermediate data for error analysis.
- The *ad hoc problem solving* function enables users to submit an *ad-hoc* analytics task similar to those used in the benchmark set and inspect the output from the current best system.
- The *trend analysis* view allows users to visualize performance change across experiments over a selected period of time.

### 6.2 ECD Framework

In this section, we describe the ECD (Extended Configuration Descriptor) framework\footnote{https://github.com/oaqa/uima-ecd} an extension to the UIMA framework that supports specification and evaluation of an analytics space. We focus on the implementation of a YAML-based declarative descriptor language, performance
evaluation API, and the data persistence layer – JDBC providers.

6.2.1 YAML-based Declarative Descriptors

The UIMA framework defines three main component types of an information system, namely CollectionReader, Annotator, and CasConsumer. A user should create a Java class to describe the logic of a component, and create an XML descriptor, usually from the UIMA Eclipse plug-in, to configure the component. Then, the user creates XML aggregate analysis engine descriptors and/or a collection processing engine descriptor to construct the end-to-end pipeline that integrates the component-level descriptors. The extended configuration descriptor (ECD) aims to find the balances between ease of creating and flexibility. It inherits the UIMA framework and uses the XML type system and Java components, but uses a YAML descriptor to allow easy manual editing and specification of multiple options. The user creates a single YAML component descriptor per Java class to specify all the possible parameter values, and a single YAML system descriptor per analytics space to specify all the procedures and analysis components.

YAML (Yet Another Markup Language or YAML Ain’t Markup Language) is a human-readable data serialization format, has a simple but rich syntax to represent and automatically infer various commonly used data types. We use three basic components of YAML.

**Associative arrays.** Keys are separated from values by a colon and a space, which can be naturally adopted to represent parameter values. For example,

```yaml
class: classpath.LingPipeNameEntityRecognizer
ModelFilePath: /ne-en-bio-genia.TokenShapeChunker
```

**Block literals.** We can define our own ECD-specific syntax for special parameters, e.g. the Resource loader syntax.

```yaml
builders: |
    - inherit: yamlpath.map-report-component
```

**Lists.** A list can be represented in the block format or inline format. The block format uses a hyphen and space to begin a new item in list, and the inline format specifies all the items in a single line, delimited by a comma and space and enclosed in brackets. The former is convenient for specifying component-level options, and the latter is convenient for listing all the parameter-level options. For example,

```yaml
- inherit: pos.lingpipe-hmm
- inherit: pos.stanford-maxent
```

or

```yaml
[0.1, 0.2, 0.3, 0.4]
```

Similar to the XML analysis engine descriptor (including collection reader or pipeline annotator) in UIMA SDK, the building blocks of a component-level ECD are a YAML associative array of the Java class that implements the component and the parameters that the component requires with their (possibly multiple) values.

* The simplest example of a component-level descriptor only defines the class path of the com-
ponent, which neither requires a parameter specification nor awaits an inheriting descriptor to specify the missing parameter values (effectively abstract). We can use the following descriptor.

```plaintext
class: edu.cmu.lti.oaqa.ecd.example.FirstPhaseAnnotatorA1
```

- When we need to specify the parameter values, we can start a new line for each parameter and value, for example,

```plaintext
class: edu.cmu.lti.oaqa.ecd.example.FirstPhaseAnnotatorA1
  foo: bar
```

- We introduce the keyword `cross-opts` to indicate that a list of values are specified for each enclosed parameter. If a parameter has a value type of array or iterable, then when this parameter is listed under the `cross-opts` block, we should use a 2-dimensional array instead. An example of the use of `cross-opts` keyword is shown below:

```plaintext
class: edu.cmu.lti.oaqa.ecd.example.FirstPhaseAnnotatorA1
  foo: bar
  cross-opts:
    parameter-a: [value100, value200]
    parameter-b: [value300, value400]
```

A total of \(2 \times 2 = 4\) descriptors for all the combinations will be virtually generated by the ECD framework.

The ECD framework supports inheritance of descriptors, i.e. a component-level descriptor can refer to another component-level descriptor as its template and override its predefined parameter values. The base descriptor should be specified by the keyword `class` (as in the examples above), and the inheriting descriptor should use the keyword `inherit`.

**class** The `class` keyword looks for the specified class within the class-path, and is intended to be used as a shorthand for a component that does not have any configurable parameter. For example, the `bar.yaml` descriptor:

```plaintext
# bar.yaml
class: bar.Bar
fixed-param: a
```

**inherit** The `inherit` keyword looks for a resource file within the class-path on the path specified by the dotted syntax, i.e. `a.b.c` directs to the file located at `a/b/c.yaml`. For example, the `foo.yaml` descriptor

```plaintext
# foo.yaml
inherit: bar
var: [x, y]
```

is effectively

```plaintext
class: bar.Bar
var: [x, y]
fixed-param: a
```

153
Similar to the collection processing engine descriptor, an analytics space level ECD (or main ECD) is a YAML associative array of three elements, collection-reader, the actual pipeline, and the optional post-process pipeline.

**collection-reader.** The collection-reader element is a YAML associative array. Developers should define the parameters that this collection reader requires. An example collection-reader element is given below.

```
collection-reader:
  inherit: yamlpath.collection-reader
  # or class: classpath.CollectionReader
dataset: BIO-COMBINED
sequence-start: 160
sequence-end: 187
```

**pipeline.** A pipeline element is a YAML list, which defines the phases and the component options for each phase. An example with two phases, which have one or two component level options, is given below.

```
pipeline:
- inherit: ecd.phase
  name: tokenizer
  options: |
    - inherit: bioqa.keyterm.default
- inherit: ecd.phase
  name: pos-tagger
  options: |
    - inherit: bioqa.keyterm.pos.lingpipe-hmm
    - inherit: bioqa.keyterm.pos.stanford-maxent
```

A ecd.phase defines an analysis step (or phase), which is a special analysis engine that manages the configured components (listed under the keyword options). In addition to the mandatory keyword options, we also introduce optional keywords, including name, timeout, lazy-initialization, etc. The ecd.phase sequentially executes each option, and the ECDDriver performs exhaustive evaluation over all the inputs and components at all phases as a depth-first search.

We also defined advanced workflows. We extend ecd.phase to jdbc.phase to further support the persisted configuration space exploration, and extend to aml.phase to support policy optimization using sampled backup (which is described in Section 6.3 in detail).

**post-process.** The post-process element is similar to the CasConsumers, which are optional. We usually define the post analysis steps here, e.g. aggregation of per-instance evaluation results.
6.2.2 Performance Evaluation API

In BaseQA 1.0, we defined a set of specialized Evaluators, such as FMeasureEvaluator, PassageMAPEvaluator, etc. Since BaseQA 2.0, we have defined three different roles in the evaluation: EvaluateeProvider, Measure, EvalCalculator, and have introduced a single Evaluator class that can dynamically combine the roles to perform evaluation for various tasks, and optionally persist the evaluation results to a database via EvalPersistenceProvider. Decoupling of measure and evaluatee from the actual calculation code allows us to reuse the same logic for various similar evaluation tasks. For example, let us assume that we have a DocumentMAP evaluation component and we plan to adapt it to evaluate AnswerMAP, we only need to implement a new AnswerEvaluateeProvider, without reimplementing the MAP calculation logic. All these interfaces define a getName() method, which is used for the purpose of reporting and persisting evaluation results.

EvaluateeProvider. It defines where and how to locate the system output and the gold standard output for comparison. The gold standard output is usually stored in a separate GoldStandard view by the GoldStandardDecorator during the CollectionReader phase, pretending to be hidden from the processing pipeline.

```java
public interface EvaluateeProvider<T> extends Resource {
    Collection<T> getGoldStandard(JCas jcas);
    Collection<T> getResults(JCas jcas);
    Comparator<T> comparator();
    Function<T, String> uniqueIdMapper();
    String getName();
}
```

If there are multiple outputs from either the system or in the gold standard set, we also need to specify how to rank them for evaluation, which is defined by the comparator method. If a data object has a rank or score field, then we can simply define the ascending order (if use the rank field) or descending order (if use the score field). To determine whether a system output matches a gold standard output, we define the uniqueIdMapper method, which behaves similarly to the equals method but gives the developer more flexibility to define what can be considered a match. For example, an answer to a factoid question is often a named entity or a concept in a knowledge base, which usually has a number of synonyms. Therefore, we may only use the concept ID, the default or official name to represent the output.

Measure. It defines the name of the specific evaluation metric. A EvalCalculator can produce multiple evaluation measurements, i.e. defines a one-to-many mapping. For example, we prefer using a single EvalCalculator to calculate the precision, recall, and F-measure to creating a different EvalCalculator for each metric. We use Measure as the key to distinguish these measurements.

```java
public interface Measure {
    String getName();
}
```
EvalCalculator. It extends to the interface sketch we present in Section 6.1 by further considering the comparator and the uniqueIdMapper.

```java
public interface EvalCalculator<T> extends Resource {
    Map<Measure, Double> calculate(JCas jcas,
                                    Collection<T> resultEvaluatees,
                                    Collection<T> gsEvaluatees,
                                    Comparator<T> comparator,
                                    Function<T, String> uniqueIdMapper);
    Map<Measure, Double> accumulate(
                                    Map<Measure, ? extends Collection<Double>> measure2values);
    String getName();
}
```

6.2.3 Data Persistence – JDBC Providers

Various framework components require intensive data access. A CollectionReader may initialize CASes by reading input elements from local files, remote database, or Web services. For the purposes of training and evaluation, a GoldStandardDecorator may also be specified to further load the gold standard outputs of a given task from a file or database. A CasConsumer, on the other hand, may write CAS data back into local files, remote database, or sending it back to the service requester. An EvaluatorConsumer measures per-episode/instance performance and/or calculates the aggregated performance. Both results may be stored in a spreadsheet or a rela-
tional database. To standardize the data access, we introduce a set of PersistenceProvider
Java classes and the corresponding persistence-provider YAML descriptors, as shown in
Figure 6.1.

**ExperimentPersistenceProvider.** It stores experiment related meta information.

```java
public interface ExperimentPersistenceProvider extends Resource {
    void insertExperiment(String id, String name, String author,
                           String configuration, String resource);
    void updateExperimentMeta(String experimentId, int size);
    void updateExperimentMeta(String experimentId, int size,
                               Set<String> topics);
}
```

**PhasePersistenceProvider.** It performs automatic CAS persistence.

```java
public interface PhasePersistenceProvider extends Resource {
    void storeCas(byte[] bytes, ExecutionStatus success,
                   long endTime, String key) throws IOException;
    CasDeserializer deserialize(JCas jcas, String hash);
    void storeException(byte[] bytes, ExecutionStatus failure,
                         long endTime, String key);
    void insertExperimentMeta(String experimentId, int phaseNo,
                               int stageId, int size);
    void insertExecutionTrace(String optionId,
                               String sequenceId, String dataset,
                               Integer phaseNo2, String uuid,
                               long startTime, String hostName,
                               String trace, String key);
}
```

**LogPersistenceProvider.** It stores log information.

```java
public interface LogPersistenceProvider extends Resource {
    void log(String uuid, Trace trace, LogEntry type,
              String message);
}
```

**EvaluationPersistenceProvider.** It stores evaluation information. The new interface
has been introduced since BaseQA 2.0.

```java
public interface EvalPeristenceProvider extends Resource {
    void insertPartialMeasurements(Key key, String sequenceId,
                                    String calculatorName, String evaluateeName,
                                    Map<Measure, Double> measure2value);
    Table<Key, Measure, List<Double>> selectPartialMeasurements(
        ExperimentKey experiment, String calculatorName,
        String sequenceId, String dataset,
        Integer phaseNo2, String uuid,
        long startTime, String hostName,
        String trace, String key);
String evaluateeName);

void deletePartialMeasurements(Key key, String sequenceId,
   String calculatorName, String evaluateeName);

void insertAccumulatedMeasurements(Key key,
   String calculatorName, String evaluateeName,
   Map<Measure, Double> measure2value);

void deleteAccumulatedMeasurements(ExperimentKey experiment,
   String calculatorName, String evaluateeName);
}

GoldStandardPersistenceProvider. It is a task specific persistence provider, which retrieves gold standard results from a file or data set.

public interface GoldStandardPersistenceProvider
   extends Resource {
      List<Passage> populateRetrievalGS(String dataset,
         int sequenceId, JCas docGSView);
   }

We provide a dummy implementation for each persistence provider, which performs no data access when being invoked. For example, the ecd.phase employs the “dummy” implementation, called DefaultPhasePersistenceProvider, which does not need any configuration parameter indicating how and where to perform the data persistence:

# ecd.phase
   class: edu.cmu.lti.oaqa.ecd.phase.BasePhase
   persistence-provider: |
      class: edu.cmu.lti.oaqa.ecd.impl.DefaultPhasePersistenceProvider

Different occasions may have different requirements for the persistence provider. For example, a simple local embedded database such as SQLite may work the best for prototyping, and a full relational database management system such as MySQL has larger storage capacity and can safely handle concurrent accesses by multiple processes, and thus works the best for a large-scale exploration experiment. We provide implementations for the SQLite and MySQL databases in the jdbc-provider module and we need to specify the SQL statements and account information as parameter values in each persistence provider descriptor. For example,

# helloqa.db.local-persistence-provider
   class: edu.cmu.lti.oaqa.cse.driver.impl.JdbcExperimentPersistenceProvider
   insert-experiment-query: |
      INSERT IGNORE INTO experiment
         (id, name, author, configuration, resource) VALUES (?,?,?,?)
   update-experiment-meta-query: |
      UPDATE experiment SET size = ? WHERE id = ?
   update-experiment-topics-query: |
      UPDATE experiment SET size = ?, sequenceIds = ? WHERE id = ?

[https://github.com/oaqa/jdbc-providers/](https://github.com/oaqa/jdbc-providers/)
We present the database schema in Table A.1 of Section A.1.

6.3 AML Extension to the ECD Framework

In this section, we introduce the extension AML Framework to the ECD Framework, which defines collection readers, AML Phase, and exploration strategies required to perform analytics space exploration using the methods introduced in Chapter 5.

6.3.1 Random Access Collection Reader

The UIMA framework loads all the analysis tasks in the data set using a fixed order collection reader and processes each input using a fixed pipeline. The ECD framework also assumes to use a finite data collection, but it processes each input using all the combinations of components defined in the configuration/analytics space. The AML extension assumes infinite inputs to allow continuous exploration. It first loads the analysis tasks from a data collection and then randomly samples a task each time. In case the user intends to specify a limit on the maximum number of episodes that the exploration process can experience, we also introduce a parameter \( n \). A user should also specify a finite \( n \) for non-budgeted exploration, otherwise he/she has to stop the exploration by killing the process. \( n \) can be ignored or specified as a large integer if a budgeted exploration strategy is used, since the budget is used as the stopping criterion for this case. We introduce a set of random access collection readers to simulate this process.

6.3.2 AML Phase

AmlPhase manages the execution at each step. It first selects and executes an option from the list of candidate options, and in the mean time, it is also responsible to catch any exception thrown from the analysis components. Once the execution is complete, it evaluates the output and generates an immediate reward for this step. AmlPhase is an extension to the BasePhase used in the ECD framework, and thus it also defines similar parameters, such as persistent-provider, name, options, etc. In addition, it introduces a set of parameters specific for analytics space exploration, including aml-strategist, and task performance and runtime performance calculation providers.

aml-strategist. An exploration strategy should be specified. There are two types of interactions between the AmlPhase and the AmlStrategist in each step. The AmlPhase first presents the data (JCas) to the AmlStrategist and asks which option to take at this step, and then reports the immediate reward back to the AmlStrategist. Therefore, we define the AmlStrategist interface as follows.
public interface AmlStrategist extends Resource {
    int selectOption(JCas currentState, int phaseNo, int optionCount);
    void report(JCas nextState, double runtimePerformance, double taskPerformance, boolean isSuccessful, boolean isFirstPhase, boolean isLastPhase);
}

task-performance-calculator, -evaluatee, and -measure. Previously, with ECD Phase, we tended to specify the evaluation components at the end of the pipeline (e.g. in the post-process block). However, the CSE/AML MDP allows immediate rewards. Therefore, we introduce these three parameters for a user to optionally specify the evaluation component at each step. The definition of these interfaces are the same as in Section 6.2.

runtime-performance-evaluator. We also introduce the interface to measure the runtime performance, which defines only a single method:
    double evaluate(JCas jcas);

We have built in a runtime performance evaluator that measures the execution time.

public class ExecTimeEvaluator extends ConfigurableProvider implements RuntimePerformanceEvaluator {
    private static long prevTime;
    @Override
    public boolean initialize(ResourceSpecifier aSpecifier, Map<String, Object> aAdditionalParams) throws ResourceInitializationException {
        if (!super.initialize(aSpecifier, aAdditionalParams))
            return false;
        prevTime = System.currentTimeMillis();
        return true;
    }
    @Override
    public double evaluate(JCas jcas) {
        long curTime = System.currentTimeMillis();
        double result = curTime - prevTime;
        prevTime = curTime;
        return result;
    }
}

task- and runtime-performance-scale. The task performance and runtime performance are not measured on the same scale. For example, a task performance measured by F-measure has a range between 0 and 1, and a runtime performance measured as the execution time in milliseconds can have a much wider range. Since the AmlStrategist expects a single scalar reward at each step, we need to combine the task performance and the runtime
performance. We define these two parameters to rescale the values.

6.3.3 Exploration Strategists

We provide built-in strategists as well as wrappers to external reinforcement learning framework libraries.

Built-in Strategists

We implement AmlStrategist for value-based strategies, policy-based strategies, and model-based strategies described in Chapter 5. Each strategist can be configured with what state representation the learner uses and the exploration uses, the option selection method, a counter (the usual undiscounted counter or the exponentially discounted counter) to use, a q-model or benefit-model, a cost-model and a budget for budgeted exploration. We define a provider (including an interface, implementations and descriptors) for each module.

**learner- and exploration-state-rep-providers.** A StateRepProvider extract a set of key-value features for state representation, which defines only a single method:

```java
Map<String, String> represent(JCas jcas);
```

We develop the MetaInfoProvider to extract the current factor, current phase, current component, and partial trace from the input JCas. A user specifies which meta information types to use via a set of parameters. A list of StateRepProvider descriptors can be specified as the value to the learner-state-rep-providers parameter, where the key-value features are combined from the outputs of these providers. As discussed in Section 5.4.2, we may use a different state representation for exploration from the one used by the learner, which is specified via the exploration-state-rep-providers parameter.

**option-selector.** An OptionSelector should return an OptionSelectionDetail object to the AmlPhase that contains not only the selected option, but also the behaviorPolicyProbability and greedyPolicyProbability, which can only be inferred from the option selection process and are necessary for off-policy methods. Almost all available information is provided to the OptionSelector as arguments, including the estimated values and costs for the options, the current state representation, the phase number, the counter, the total budget and the remaining budget. We define the interface as follows.

```java
<R> OptionSelectionDetail selectOption(
    Collection<Integer> options,
    Map<Integer, Double> rewardScores,
    Map<Integer, Double> costScores,
    Map<String, R> state, int phaseNo,
    StateOptionCounter<R, ?> counter,
    double totalBudget, double remainingBudget);
```

We implement EpsilonGreedyOptionSelector,
DecayedEpsilonGreedyOptionSelector, SuccessiveOptionRejector, UCB1OptionSelector, UCBEOptionSelector, etc.

**q-model, policy, benefit-model.** Depending on the strategy, one may need to specify a q-model for a value-based method, a policy for a policy-based method, and a benefit-model for a model-based method. Moreover, the user may have to specify multiple models for a strategy. For example, a AdvantageActorCriticStrategy requires a policy as the ReinforceStrategy for actor, and a q-model for the critic. a BEM strategy also requires to specify an error model as well. We provide TabularQModel, SGDLinearQModel, LSELinearQModel, SoftmaxPolicy, SGDLinearBenefitModel, and LSELinearBenefitModel. These models also require configurations. For example, the user needs to specify the learner-counter and the learning rate function via the alpha-provider parameter (two learning rate functions to both the actor and the critic for the actor-critic strategies via the additional beta-provider) for the TabularQModel, SoftmaxPolicy, and the SGD based models. The LearningRateProvider is defined as follows.

```java
<R> double getLearningRate(Map<String, R> state, int option, StateOptionCounter<R, ?> counter);
```

The user also specifies a coefficient-calculator to adjust the learning rate according to the off-policy update rule. We implemented two learning rate providers: the ConstantLearningRateProvider for a constant learning rate and the InverseCountLearningRateProvider for the inverse decayed learning rate ($\frac{1}{N}$) or the nonstationary off-policy learning. LSE based batch learning models require to specify a linear regression method, and the error model requires to specify a classifier to train and predict errors. The cost-model can be defined similarly.

When the selectOption method is invoked, it first generates the state representation via the learner-state-rep-providers, and calculates the estimated reward via the getQ method of a QModel or the estimateQ method of a BenefitModel, and the estimated cost via getCost method of a cost model if specified. These values are sent to the option-selector to select the greedy or exploratory option in each step. When the report method is invoked, the strategist first calculates the reward by combining the task and runtime performance measured at this step, and then a Monte Carlo strategist maintains an Episode and appends the current step, including the state, the OptionSelectionDetail, and reward information at each step. Finally, it updates the QModel, ParameterizedPolicy, or BenefitModel only at the end of the episode (determined by !isSuccessful || isLastPhase).

**External RL Framework Wrapper**

The framework also supports integrating external reinforcement learning frameworks, such as BURLAP[^6] Such frameworks usually define their own driver, e.g. the runLearningEpisode method in the BURLAP framework, and forces the developers to represent their reinforcement learning models in a specific manner. For example, the BURLAP framework requires the developers to define a learner that can interact with an environment and learn policies over time. This learner is responsible for selecting actions to take in the environment, receiving feedback in the form of rewards, and updating its internal model of the environment based on this feedback. The framework also provides support for defining policies and value functions, which are used to estimate the expected future rewards for different actions. The user can then use the framework to experiment with different policies and compare the results. This makes it possible to evaluate the performance of different policies and identify the most effective one for a given task. Overall, the framework provides a powerful tool for researchers and practitioners to develop and test reinforcement learning algorithms in a variety of domains. |

[^6]: http://burlap.cs.brown.edu/
learning problem under their own Environment interface. The environment usually defines a method that takes an action and returns an outcome, i.e.

\[
\text{Outcome} < R \text{ execute (int option)};
\]

However, it is difficult to reimplement the entire UIMA framework or ECD framework under their Environment interface to allow execution of each analysis step by their runLearningEpisode method instead of the UIMA framework driver. In order to bridge the gap between two frameworks that each have their own drivers, we design the wrapper to create a separate thread for the external RL framework driver, and leverage the BlockingQueue data structure to manage the communication between the main thread (UIMA) and the RL framework thread.

In particular, we create two BlockingQueues of maximum capacity of one, one for options and the other for outcomes. When the UIMA thread calls the selectOption method the first time, it initializes a virtual Environment for the RL thread by defining the initial state to be the input JCas of the selectOption method, and then immediately starts the RL process by invoking its runLearningEpisode method. The RL thread then produces an option (via the argument of the execute method) based on the current state in the Environment and waits for the Environment to return an Outcome. Now, our virtual environment puts the option into the option queue, and puts the RL thread to sleep. The UIMA thread takes the option from the option queue and processes it using the analysis component referred by the selected option. Once the execution is done, the UIMA thread creates an Outcome object from the returned JCas object, and puts it into the outcome queue and turns itself into asleep, which also wakes up the RL thread. The RL thread takes the outcome to update its model and selects the next option. This process continues until the end of the episode, when the Environment is reset to the initial state.

6.4 Conclusion

This chapter focuses on the design and implementation details of the analytics meta learning framework. We first present the design principles in Section 6.1 and then we describe the UIMA extensions ECD and AML frameworks as an examples, which support the analytics space description and exploration respectively. The framework is used in the simulated experiments in Sections 5.5, 5.7 as well as case studies in Chapters 7–9.
Chapter 7

Case Study: Biomedical Question Answering Task

In the next three chapters, we focus on exemplifying the proposed methodology to tackle analytics tasks and reporting the performance of the proposed analytics meta learning method and its resulting systems. In each chapter, we first describe the task, and then follow the general solution framework proposed in Section 2.3 to solve each individual analytics task. In each respective section, we give our manual definition of the analytics procedure or describe the automatic analytics procedure definition process, describe the analysis component integration or development, and report the analytics space exploration results.

This chapter presents the use of analytics meta learning in construction of an information system that supports a series of real-life information seeking needs from biomedical experts, e.g. the Task Examples 1 to 5 in Chapter 1. We first review prior biomedical information systems in Section 7.1. We describe the task in detail in Section 7.2. In this case study, we manually define the analytics procedure following a general question answering pipeline [193]. In Section 7.3, we leverage a UIMA-based three-layered architecture that was previously developed for biomedical QA tasks including TREC Genomics questions [220], CLEF QA4MRE questions [150]; the architecture consists of a generic component layer (BaseQA), a biomedical component layer (BioQA) and a BioASQ-specific component layer. In Section 7.4, we describe the analysis components that we have developed for the task. Using the development set, we also study to design and train supervised models to retrieve relevant contents, including concepts, documents, snippets, and to answer factoid, list, and yes/no questions, instead of manually-constructed rules or predefined templates. For example, we predict answer type(s) from among the 133 semantic types in the UMLS Semantic Network, and utilize supervised models to merge answer scores obtained from various sources, a technique used by several systems in past years [147, 212]. We report our preliminary results in Section 7.5. In Section 7.6, we report the configuration exploration results for two tasks: relevant snippet retrieval and factoid QA tasks, and compare the best system produced by each of the 40 budgeted exploration strategies with the baseline system.

The system developed using the analytics meta learning methodology has participated in the
2015 and 2016 BioASQ QA challenges [222, 223]. Our hypothesis is that informatics challenges like BioASQ are best met through careful design of a flexible and extensible architecture, coupled with continuous, incremental experimentation and optimization over various combinations of existing state-of-the-art components, rather than relying on a single “magic” component or a single end-to-end model. The architecture framework and the components are available as open-source downloads.

7.1 Related Work

We focus on summarizing systems and approaches that have participated in the previous shared tasks related to biomedical question answering, e.g. TREC Genomics [79] and BioASQ [199], or have leveraged the task benchmarks outside the formal competition.

Traditionally, a typical biomedical information retrieval pipeline consists of three major analysis components: keyterm extraction and query expansion, document (or paragraph) retrieval, and passage extraction [165, 177, 190]. Synonyms, acronyms, and lexical variants are processed in the first phase; a retrieval model is applied in the second phase; and the similarity between the query and each retrieved passage is considered in the last phase. Researchers have tried to apply existing general retrieval systems [21, 167], relevance feedback to the traditional retrieval model [122, 181, 226], or a fusion or shrinkage of retrieval models [48]. Moreover, linguistic knowledge is also incorporated for the task, e.g. POS tags [130] and SVO structure [190]. Recently, researchers in biomedical information retrieval and question answering continue to leverage the TREC Genomics data set for evaluation [34, 86, 118, 131, 140, 198, 227]. Given the absence of an easy-to-use framework for building baseline systems for new tasks and exploring large parts of the configuration space, iterative research typically focuses on perturbations to a single module while keeping modules and parameters elsewhere in the system frozen. Stokes et al [185] follow the same assumption to combine components from task participants to find optimal configurations for a biomedical IR task, using an approach specific to the task and difficult to generalize.

7.2 Task Description

A biomedical question answering task defines infinite concrete analytics tasks, which can be defined by pairs of information seeking needs described in natural language (e.g. Task Examples 1 – 5 in Chapter 1) and expected exact answers (e.g., named entities in the case of factoid and list questions) and ideal answers (paragraph-sized summaries) [125]. For example, Task 1

How many TAp73 isoforms have been identified in humans?

should have an exact answer of either

seven or 7

and an ideal answer of

https://github.com/oaqa/bioasq

166
The TP73 gene, due to the presence of two promoters (P1 and P2) in its 5’ flanking region, encodes a fully transcriptionally active domain (TAp73) and the amino terminus deleted (ΔNp73). TAp73 possesses pro-apoptotic properties, while deltaNp73 has anti-apoptotic functions. Alternative 3’-end splicing results in generation of at least seven TAp73 distinctive isoforms (α, β, γ, etc).

In addition, the user would be interested in knowing the evidence that supports the answer, including relevant PubMed articles (documents):

PMID: 18256531
PMID: 22388545
PMID: 23159862
...

relevant text snippets within the documents

A member of the p53 family, p73, has several isoforms and differentially regulates transcription of genes involved in the control of the cell cycle and apoptosis.

(0 - 162 from the abstract of PMID: 18256531)

The expression of all 5 N-terminal isoforms (TAp73, DeltaNp73, DeltaN’p73, Ex2p73 and Ex2/3p73) was measured by real-time RT-PCR and p53 status was analyzed by immunohistochemistry. TAp73, DeltaNp73 and DeltaN’p73 were significantly upregulated in tumors.

(347 - 602 from the abstract of PMID: 16964385)
...

and relevant concepts from knowledge bases

MeSH Heading: Protein Isoforms (D020033)
...

7.2.1 Evaluation Metrics

As motivated in Chapter [1] any domain expert can hardly solve all the analytics tasks, not to mention an information system. In order to automate the system construction process, we need to define metrics to estimate to what extent the system is able to answer the questions. We adopt the evaluation metrics that have been adapted for similar scenarios to measure different types of output [79][199].

The relevance of documents and concepts is measured by precision@N, recall@N, F-measure@N, average precision@N (AP@N) at the task/instance level and averaged at the experiment level,
where are referred to as mean precision@N, mean recall@N, mean F-measure@N, mean average precision@N (MAP@N). In particular, they are defined as follows

\[
\text{precision}@N = \frac{\text{true positive}@N}{\text{true positive}@N + \text{false positive}@N} \\
\text{recall}@N = \frac{\text{true positive}@N}{\text{true positive}@N + \text{false negative}@N} \\
\text{F-1}@N = 2 \cdot \frac{\text{precision}@N \cdot \text{recall}@N}{\text{precision}@N + \text{recall}@N} \\
\text{AP}@N = \frac{\sum_{i=1}^{N} \text{precision}@N \cdot \text{rel}@N}{\min(|\text{relevant items}|, N)}
\]

where rel@N equals 1 if the N-th returned item is relevant and 0 otherwise. The relevance of snippet (or passage) is measured by Passage MAP and Passage2 MAP [79], which consider the character level overlapping between the system generated results and the gold standard outputs.

The correctness of the system generated exact answers are estimated using accuracy@N, reciprocal rank@N, precision, recall, F-measure, depending on the task (question) type, at the task/instance level and averaged to mean accuracy, mean reciprocal rank (MRR), mean precision, mean recall, mean F-measure, etc., at the experiment level. In particular,

- if the question or task description requires either a “yes” or “no” as an answer (i.e. yes/no question), then the task performance is judged using binary accuracy@1, i.e. 1 if correct and 0 if wrong at the task/instance level, and mean accuracy at the experiment level;
- if the question or task description requires a particular entity (e.g. a disease, drug, or gene) as an answer (i.e. factoid question), then the task performance is judged using accuracy@1 (strict accuracy) and accuracy@5 (lenient accuracy) [199], i.e. 1 if there exists a correct answer in the first five answer candidates, and reciprocal rank@N, i.e. the inverse of the rank of the first correct answer in the candidate list;
- if the question or task description requires a list of entities (e.g. a list of genes) as an answer (i.e. list question), then the task performance is judged using precision, recall, and F-measure.

If the question or task description can be responded only by a short text summarizing the most prominent relevant information, we call it summary question or non-factoid question. To judge the performance of generating ideal answers for yes/no, factoid, list, and summary questions, we use ROUGE-2 and ROUGE-SU4 [113], which counts the overlapping N-grams between the system generated summary and the reference or golden summary, which are defined formally as follows.

\[
\text{ROUGE-}N = \frac{|\text{overlapping } N\text{-grams}|}{|\text{N-grams in golden summary}|} \\
\text{ROUGE-}SU_N = \frac{|\text{overlapping skip bigrams limited to } N| + |\text{overlapping unigrams}|}{|\text{skip bigrams in golden summary limited to } N| + |\text{unigrams}|}
\]
Table 7.1: Comparison of biomedical question answering shared tasks: TREC Genomics and BioASQ

<table>
<thead>
<tr>
<th>Example</th>
<th>TREC Genomics</th>
<th>BioASQ QA</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is the role of PrnP in mad cow disease?</td>
<td>Is TREM2 associated with Alzheimer’s disease?</td>
<td></td>
</tr>
<tr>
<td>What [GENES] regulate puberty in humans?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coverage genomics</td>
<td>biomedical</td>
<td></td>
</tr>
<tr>
<td>Years</td>
<td>2006, 2007</td>
<td></td>
</tr>
<tr>
<td>Benchmark size</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>810 (as of 2014)</td>
<td>1307 (as of 2015)</td>
<td></td>
</tr>
<tr>
<td>1799 (as of 2016)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Question type</td>
<td>summary, factoid</td>
<td></td>
</tr>
<tr>
<td>Answer type</td>
<td>relevant documents and passages</td>
<td></td>
</tr>
<tr>
<td>Best reported results</td>
<td>Document MAP: ~0.50</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Passage MAP: ~0.17</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Document MAP: 0.26 – 0.30</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Passage MAP: 0.04 – 0.06</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Concept MAP: 0.43 – 0.68</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Triple MAP: 0.04 – 0.09</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Answer MRR: 0.05 – 0.28</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(as of 2014, scores vary across different test batches)</td>
<td></td>
</tr>
<tr>
<td>Corpus</td>
<td>PMC Open Access full texts</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PubMed abstracts, PMC Open Access full texts</td>
<td></td>
</tr>
</tbody>
</table>

7.2.2 Data Set

Besides evaluation metrics, another crucial element when using analytics meta learning for constructing information systems is a benchmark data set. In fact, a number of shared tasks have been organized to evaluate the performance of biomedical question answering (QA) systems. For example, TREC Genomics QA [79] focuses on genomics questions for a subdomain of biomedical research, and makes available a relatively small set of factoid and list questions (a total of 64 questions from two years). Recently, the CLEF QA4MRE task [152] organized a pilot track on multiple choice QA for questions related to Alzheimer’s disease. Compared to these shared tasks, the BioASQ challenge [199] covers a wider range of biomedical subdomains, and releases a larger topic set with much more detailed gold standard data, including relevant documents, snippets, concepts, triples and both exact and ideal answers. We compare them in Table 7.1.

We have demonstrated using TREC Genomics benchmark data set for biomedical passage retrieval system in Section 4.2. In this chapter, we use the BioASQ QA benchmark set for optimizing the biomedical question answering system.
Figure 7.1: Relevant content retrieval (Phase A) pipeline diagram. * represents a provider that requires accessing BioASQ Web services.

7.3 Analytics Procedure Definition

In this case study, we consider three fixed information processing pipelines – relevant content retrieval pipeline (Phase A), factoid and list QA pipeline, and yes/no QA pipeline (Phase B). Relevant content retrieval pipeline retrieves relevant concepts, triples, documents, and snippets, and Phase B pipelines generate answers for specific types of questions. We utilize the conventional information retrieval and question answering methods to design both pipelines, and we illustrate the relevant content retrieval pipeline in Figure 7.1, the factoid and list QA pipeline of Phase B in Figure 7.2, and the yes/no QA pipeline of Phase B in Figure 7.3.

To integrate the analysis components into the analytics procedures at each analysis step and facilitate the extensibility of the system, we design a three-layered architecture uses the analytics meta learning framework.

The first layer BaseQA[^3] is designed for domain-independent analysis components, and includes the basic input/output definition of analytics engine, intermediate data objects (such as answer type, question type, relevant passages, relevant concepts, etc.), analytics task evaluation

[^3]: https://github.com/oaqa/baseqa/
Figure 7.2: Factoid and list QA (Phase B) pipeline diagram. † represents a provider that requires accessing external Web services.
Figure 7.3: Yes/no QA (Phase B) pipeline diagram. † represents a provider that requires accessing external Web services.
components, and data processing components (e.g. LingPipe\footnote{http://alias-i.com/lingpipe/index.html} and Apache OpenNLP\footnote{https://opennlp.apache.org/} wrappers, Lucene\footnote{https://lucene.apache.org/}-based passage retrieval component, LibLinear\footnote{http://www.csie.ntu.edu.tw/~cjlin/liblinear/} wrapper, and models applicable to generic English questions). Although this case study focuses on the biomedical domain, it is the first similar shared task, to the best of our knowledge, to combine four types of questions and evaluate both exact and ideal answers along with other relevant elements (e.g. triples), so many aspects of the existing BaseQA framework were extended to accommodate BioASQ application development. We modified the intermediate object and input/output object definition (UIMA type system) according to the task requirements. For example, we added two new attributes \texttt{Begin-/EndSection} to each \texttt{Passage} type, and changed the \texttt{Begin-/EndPosition} attributes to \texttt{Begin-/EndPositionInSection}. We also provided a BioASQ-compatible JSON format reader and writer at the BaseQA level, which we believe can be widely used in various analytics tasks beyond BioASQ. We also implemented the general-purpose evaluation methods at BaseQA layer.

In the second layer (BioQA), we implemented biomedical resources that can be used in any biomedical analytics task (outside the context of BioASQ), including UMLS Terminology Services (UTS)\footnote{https://uts.nlm.nih.gov/home.html}-based synonym expansion component, a MetaMap annotation component, etc. For the components that are included in the BaseQA layer, we also created a descriptor for the component at the BioQA level by overriding the \texttt{model} value with a path to the specific model tailored for biomedical domain, where applicable. For example, the ClearNLP wrapper, which is provided at the BaseQA level with the default \texttt{general-en} model specified in the descriptor, has a new descriptor for the \texttt{bioinformatics-en} model, trained on the CRAFT treebank\footnote{http://www.csie.ntu.edu.tw/~cjlin/liblinear/} [206], defined at the BioQA level. Although the training and testing processes are performed on the BioASQ development set, the derived models can also be used for other biomedical questions, so we also place the models and training components in the BioQA layer.

A few BioASQ evaluation specific components were integrated in the third design layer; for example, GoPubMed services are only hosted for the purpose of the BioASQ challenge. The introduction of this task-specific layer will facilitate easy replacement of proprietary and restricted components when we adapt the system to other biomedical analytics tasks or deploy the system as a real-world application. The end-to-end training and testing pipelines are also defined in this layer. Similar to the resource-wrapper providers which we introduced for the TREC Genomics task\footnote{https://github.com/ziy/bioasq-gopubmed-client/}, we also created a caching layer, using Redis\footnote{http://redis.io/} for all outgoing GoPubMed service requests, along with a Java client for accessing either the official GoPubMed server or the caching server, specified by a properties file\footnote{https://github.com/ziy/bioasq-gopubmed-client/} which helps reduce the workload of the official server and reduce the experiment runtime when multiple developers are evaluating their components.
7.4 Analysis Component Construction: Learning to Answer Biomedical Questions

In this section, we describe the analysis components that have been integrated for analytics meta learning. First, in Section 7.4.1 we describe the concept identification methods, including named entity recognizers, TmTool wrappers and C-Value algorithm. Then, in Section 7.4.2 we focus on the analysis components that we initially integrated for the BioASQ 3B challenge, which include the document retrieval component, the snippet retrieval component, the concept retrieval component, and the triple retrieval component. We implemented a general reranking module for BioASQ 4B to rerank the relevant contents of any content type, which is detailed in Section 7.4.3. Next, we focus on the factoid and list QA pipeline in Sections 7.4.4 to 7.4.8 whose components include answer type prediction component, a number of strategies for candidate answer generation, candidate answer scoring component, an answer pruning component, and a collective answer reranking module. Finally, we describe the yes/no question answering pipeline in Section 7.4.9.

7.4.1 Concept Identification

We integrate and compare a number of off-the-shelf biomedical NLP tools and resources to process the biomedical texts, including the input question and retrieved relevant passages.

Named Entity Extractors (Developed for BioASQ 3B & 4B)

All of the relevant content retrieval, factoid and list QA, or yes/no QA pipelines start with a question parsing step. We integrate the ClearNLP parser to annotate the tokens, part of speech tags, and dependency relations for the question (corresponding to Listing A.1 ll.21–24, Listing A.2 ll.22–24, Listing A.3 ll.22–25, and Listing A.4 ll.22–25, in the Section A.2). We use three approaches to identify the concept mentions in the question for BioASQ 3B. We first use the MetaMap service\(^1\) to identify the concepts and used UTS\(^2\) to retrieve variant names for each concept (Listing A.1 ll.26–29, Listing A.2 ll.26–28, Listing A.3 ll.27–30, and Listing A.4 ll.27–30). Only the first concept mapping with the confidence score returned from the service is used for each question. We also use a statistics-based LingPipe named entity recognizer (NER) with the GENIA model (Listing A.1 ll.36–39, Listing A.2 ll.30–32, Listing A.3 ll.37–40, and Listing A.4 ll.37–40), where the label of the named entity that is assigned by LingPipe NER is used as the semantic type of the concept. We then consider all noun phrases in the question as candidate concepts. Therefore, we employ the Apache OpenNLP\(^3\) chunker to detect all noun phrases (NPs) and prepositional phrases (PPs) from each question, and extract all NPs and all NP-PP-NP occurrences (Listing A.2 ll.34–40).


\(^3\) [https://opennlp.apache.org/](https://opennlp.apache.org/)
We use the same set of token and concept identification tools used for the question (described in Section 7.4.4) to annotate all the relevant snippets provided as input for Phase B (corresponding to Listing A.2 ll.54–76, Listing A.3 ll.52–80, and Listing A.4 ll.47–75).

However, due to the excessive noise introduced from the Apache OpenNLP Chunker based method, which extracts all noun phrases, we did not use this approach in the official evaluation for BioASQ 4B. In addition, we integrate the TmTool biomedical concept identification RESTful service [211] for both the semantic type labeling of gold standard exact answers and question/snippet annotation, and also use C-Value [62], a frequent phrase mining method, to extract potential out-of-vocabulary multi-word terms.

TmTool for Annotating Questions, Snippets, and Answer Texts (Developed for BioASQ 4B)

The TmTool[14] provides a standard Web service interface to annotate biomedical concepts using a number of state-of-the-art biomedical NLP parsers, which includes GNormPlus/SR4GN (for genes and species), tmChem (for chemicals), DNorm (for diseases), and tmVar (for mutations). Although it can only identify biomedical concepts belonging to any of these categories, they account for a great portion of the concepts used in the BioASQ corpus. In addition, many of these parsers utilize morphological features to estimate the likelihood of a term being a biomedical concept, rather than relying on an existing ontology like MetaMap.

TmTool supports three data exchange formats: PubTator (tab-separated), BioC (XML) and PubAnnotation (JSON). Since the PubTator format does not support DNorm annotation, and BioC format sometimes causes a single-sentence request to timeout (no response after 20 minutes), we choose the robustest PubAnnotation format. We also find that the offsets returned from the TmTool RESTful service might not align well with original request texts, especially with tmChem trigger, and hence we implement an escape method to convert the texts into a TmTool compatible format by replacing some non-ASCII characters with their normalized forms, and removing special characters.

We use the TmTool to identify the biomedical concepts and annotate their semantic types from both the questions (in Phases A and B, Listing A.1 ll.31–34, Listing A.3 ll.32–35, and Listing A.4 ll.32–35) and the relevant snippets (in Phase B, Listing A.3 ll.62–65 and Listing A.4 ll.57–69) in the same manner as MetaMap.

C-Value for Extracting Multi-Word Terms from Snippets (Developed for BioASQ 4B)

We treat the relevant snippets provided for each question in Phase B as a corpus, and we hypothesize that if a multi-word phrase is frequent in the corpus, then it is likely a meaningful concept. In order to extract not only high-frequency terms but also high-quality terms, a C-Value criterion [62] is introduced, which subtracts the frequency of its super terms from a term’s own frequency. In this way, it returns the longer multi-word terms if two candidate terms overlap and have the same frequency. This approach only applies to a corpus, rather than a single sentence. Therefore, we only use this method to extract concepts from snippets (corresponding to Listing A.3 ll.72–75 and

In the future, we may consider to collect a corpus relevant to the question, in order to apply the same idea to questions.

7.4.2 Relevant Content Retrieval & Ranking (Used in BioASQ 3B Only)

For BioASQ 3B, we develop individual relevant content retrieval and post-ranking modules using either the official retrieval service or a local search engine. We describe the implementation details in this section.

Document retrieval and ranking

We implement a LETOR based document retrieval module [217] similar to its TREC Web Track 2014 participation and integrate it into the BioASQ 3B Phase A system.

**Offline Indexing of MEDLINE Baseline Corpus.** We use Lucene to index a MEDLINE baseline corpus using title, abstract and keywords fields, if available. We use the standard Lucene tokenizer combined with the Krovetz Stemmer, which is less aggressive compared to the Porter Stemmer. This is an important step, because many biomedical terms (in particular gene names) are not recognizable by stemmers, and the Porter stemmer is likely to truncate many of the words, causing increased confusion between the stemmed biomedical terms and common terms during search time. We also keep the stopwords in the index. The motivation is that since we only have the abstract text for the document, removing stopwords may result in less accurate field length statistics, thus affecting the performance of many language model based retrieval models.

**Hierarchical Retrieval Architecture.** The fact is that given a query, we have more retrievable documents than we can perform a deeper analysis for. However, to ensure better retrieval performance, in-depth analysis of the documents is necessary. Therefore, a hierarchical retrieval architecture is introduced here to find a good balance between performance and efficiency. In summary, each search task is processed by three stages:

1. **Obtaining an affordable high recall candidate set.** During the query time, we have removed all stopwords from the query, as they provide no useful information and will likely cause serious efficiency issues. We use the Dirichlet smoothing retrieval model implemented in Lucene to conduct this search. In our implementation, we consider only the top 10,000 ranked documents.

2. **Precision oriented reranking.** We incorporate the Negative Query Generation (NQG) model [123], which utilizes a negative language model by assuming that all documents in the corpus are non-relevant, thus making more accurate adjustments to query term weights and relevance calculations. After reranking with NQG, we can now further cut down the candidate set by considering only the top 100 documents in the ranked list.

3. **Deep document feature extraction and LEarning TO Rank (LETOR).** We use ranker scores (e.g. BM25, Jelinek-Mercer smoothing, Dirichlet smoothing, Indri two-stage smoothing, NQG, etc), similarity scores (e.g. Jaccard coefficient and Dice coefficient, etc.), raw features (e.g. document length, vocabulary size, etc.), and customized features (e.g. harmonic
Table 7.2: Snippet retrieval features

<table>
<thead>
<tr>
<th>No.</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BM25: We index all the candidate snippets using Lucene, and then use a query that contains not only words but also phrases and confidence scores of all the different query concepts returned by the MetaMap service.</td>
</tr>
<tr>
<td>2</td>
<td>Skip-bigram: Based on the dependency relations generated from the dependency parser for each question, we count the number of matched pairs and calculate the F-1 based skip-bigram score.</td>
</tr>
<tr>
<td>3</td>
<td>Textual alignment: Surface similarity of a snippet and a question. We also consider the relative order of the different words.</td>
</tr>
<tr>
<td>4</td>
<td>Some other question independent features, such as the length of the snippet.</td>
</tr>
</tbody>
</table>

means of the ranker scores across all fields, the distribution of the query terms across the documents, etc.). We simply score the $K$ documents with a pretrained LETOR model which was optimized for Precision@10. Here, we are using Random Forest, an ensemble method known for robustness against overfitting.

The details of the proposed document retrieval approach can be found in [217].

Snippet retrieval and ranking

We use a simple LETOR based snippet retrieval module in BioASQ 3B. This module analyzes the 10 most relevant documents returned from the upstream document retrieval component, which first identifies the extent of a snippet and then applies a LETOR approach for snippet retrieval.

Candidate Snippets Generation. The definition of “snippet” is the original piece of text extracted from the document. In our initial study, we found that the distribution of snippet length in the gold standard answers is similar to that of sentence length. Therefore, we apply a sentence segmenter to split the snippets and define each sentence as a snippet candidate.

Feature Extraction and LETOR. We define four types of features for LETOR in Table 7.2, and also apply the logistic regression classifier for scoring.

Concept retrieval and ranking

For BioASQ 3B, we use the official GoPubMed concept retrieval service. We first identify the text spans from each question and search these texts from various GoPubMed concept services. Since only a single list of concepts is returned, we also propose to merge and rank the concept lists returned from multiple sources.

Candidate Queryable Concept Generation. We use MetaMap to identify the UMLS concepts from the question, and our results indicate a significant improvement in recall. However, one of the major drawbacks of MetaMap is that it is poor at identifying gene and protein names. To overcome this issue, we use LingPipe NER with the model trained on the GeneTag corpus to recognize gene
names to enrich the retrieved metathesaurus concepts. We then use the combination of tokens retrieved from the MetaMap service and the LingPipe NER to query various biomedical ontologies.

**Concept Ranking and Merging.** We create a ranking model that can rank the search results from different ontologies. We use the federated search approach \[176\], which trains a relevance mapping logistic function that maps the relevance scores of each result from each ontology to a global relevance scale.

**Triple retrieval**

We only participated in the triple retrieval challenge in BioASQ 3B. Similar to concept retrieval, we rely on the BioASQ provided service to retrieve relevant triples. Therefore, our goal is to construct an effective query string. Beyond the baseline method that simply concatenates all the keywords from the concept retrieval result, we make three improvements:

- Append “[obj]” and “[sub]” identifiers to each keyword in the query string.
- Enumerate all letter case possibilities for keywords: lower case, upper case, and capitalized word.
- Add all words in the original question to the keyword set while excluding the stop words and SQL reserved keywords.

The first improvement is to help the triple query server understand that most of our keywords are used as objects or subjects. This finding is intuitive through observation; since most of the words are nouns or adjectives, they are unlikely used as predicates in triples. The second improvement is based on an observation from examination of gold standard answers, where triple results indicate case-sensitivity during triple matching. Therefore, we need to include all casing variants to ensure that keywords are matched during triple retrieval. The third improvement ensures that we do not omit keywords from the original question, to make the query more robust.

### 7.4.3 Relevant Content Retrieval & Reranking (Developed for BioASQ 4B)

For BioASQ 4B, we reimplement the relevant content retrieval pipeline, which contains a candidate retrieval phase and a candidate reranking phase for each type of content. In particular, for relevant document, concept, and snippet retrieval, we first retrieve a list of 100 candidate results, then we define a set of features to estimate the relevance of each candidate result and employ a standardized interface to incorporate these features to rerank the candidate contents, where each stage employs a different retrieval / reranking strategy.

First, we replace the GoPubMed services with local Lucene indexes as the response time is estimated to be at least 20 times faster, although the task performance could be slightly worse (.2762 in terms of MAP using the GoPubMed concept Web service vs. .2502 using the local Lucene index in our preliminary experiment for concept retrieval). The concept Lucene index is created by fusion of the biomedical ontologies used by the official GoPubMed services, where we define three text fields: concept name, synonyms, and definition, and two string fields: source (e.g. Gene Ontology, Disease Ontology, etc) and URI. The document Lucene index is created from the latest MEDLINE Baseline corpus using Lucene’s StandardAnalyzer. After a list of documents
Table 7.3: Retrieval result reranking via relevant classification features

<table>
<thead>
<tr>
<th>No.</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Concept</strong></td>
</tr>
<tr>
<td>1</td>
<td>the original score from the concept retrieval step</td>
</tr>
<tr>
<td>2</td>
<td>overlapping concept count between the retrieval results and mentions annotated in the question</td>
</tr>
<tr>
<td>3</td>
<td>retrieval scores using various query formulation strategies</td>
</tr>
<tr>
<td></td>
<td><strong>Document</strong></td>
</tr>
<tr>
<td>1</td>
<td>the original score from the document retrieval step</td>
</tr>
<tr>
<td>2</td>
<td>retrieval scores using various query formulation strategies</td>
</tr>
<tr>
<td></td>
<td><strong>Snippet</strong></td>
</tr>
<tr>
<td>1</td>
<td>the score of the containing document</td>
</tr>
<tr>
<td>2</td>
<td>meta information, including the section label (abstract or title), binned begin/end offsets, binned length of the snippet, etc.</td>
</tr>
<tr>
<td>3</td>
<td>retrieval scores using various query formulation strategies</td>
</tr>
</tbody>
</table>

are retrieved and segmented into sections and sentences, the snippet Lucene index is then built in memory on-the-fly at the sentence level. The search query is constructed by concatenating all synonyms of identified concepts (enclosed in quotes) and all tokens that are covered by neither a concept mention nor a stop word, where the most 5,000 common English words\(^{16}\) are used as the stop list. Then, the query searches all text fields. The relevant content retrieval components are integrated into the Phase A pipeline of BioASQ 4B (corresponding to Listing A.1 l.57–60, 68–71, 79–82). We also make the code of creating Lucene indexes for both the biomedical concepts and MEDLINE documents available\(^{17}\).

The standardized search result reranking interface allows each retrieval task to specify different scoring functions (features). The features that we use for concept, document, and snippet retrieval are listed in Table 7.3. For example, during concept search result reranking, we can check if each candidate concept is also identified in the question by a biomedical NER. During snippet reranking, we can also incorporate the meta information, such as section label (title, abstract, body text, etc.), offsets in the section, and length of each snippet. In the candidate retrieval step, we have used a query that combines all non stop words and concepts identified by all biomedical concept annotators, in order to guarantee high recall. However, it does not optimize the precision. For example, some concept annotators or synonym expansion services may falsely identify concepts and thus introduce noisy search terms, and also some search fields tend to be less informative than others. Therefore, in the reranking step, we employ various query formulation strategies, e.g. only

\(^{16}\) [http://www.wordfrequency.info/](http://www.wordfrequency.info/)

\(^{17}\) [https://github.com/YueChou/biomedical-concept-indexer](https://github.com/YueChou/biomedical-concept-indexer) and [https://github.com/ziy/medline-indexer](https://github.com/ziy/medline-indexer)
within certain text fields and/or only using a subset of concept annotators, and consider the search score and rank of each candidate search result as features. We down-sample the negative instances to balance the training set, and use the logistic regression classifier to learn to predict relevance for all the reranking tasks. The relevant content reranking components are integrated into the Phase A pipeline of BioASQ 4B (corresponding to Listing A.1 ll.62–65, 73–76, 84–88). In the future, we can also integrate learning-to-rank modules.

### 7.4.4 Answer Type Prediction (Developed for BioASQ 3B & 4B)

Previous work has studied how to define rules to extract a *lexical answer type* (or LAT) from questions to predict the answer type, e.g. IBM’s Watson system [105]. Classification based approaches have been proposed to predict answer type from the question using syntactic and/or semantic features. Preparation of training data involves defining an answer type taxonomy manually or leveraging existing ontologies (e.g. MUC), collecting training questions (e.g. TREC QA question set), and annotating gold standard answer type(s) [111] [144]. Weissenborn et al. [212] also define patterns for LAT extraction for BioASQ questions. However, the use of predefined rules or patterns presents two challenges. First, it requires a huge amount of human effort to define a pattern set that has not only sufficient coverage of existing questions but which also generalizes to unseen questions that are similar, but do not exactly follow the linguistic structure of known questions. Second, even if a novel question can be mapped to one of the predefined templates, the extracted LAT set can still be arbitrarily large, and how to interpret the LATs and use them for answer extraction and scoring might still be a challenging problem.

We leverage the UMLS Semantic Network and TmTool results and map the LATs to the ontological hierarchy to obtain one of the UMLS semantic types or TmTool annotated concept type as an *expected answer type*, which is used for type coercion checking. We take advantage of these ideas and further incorporate the Q/A pairs in the BioASQ data set for training a multi-class answer type classifier that predicts candidate answer type(s).

**Answer type definition**

We introduce two additional question types: **CHOICE** and **QUANTITY** in addition to the UMLS semantic types. **CHOICE** questions are those that have candidate answers expressed explicitly in the question, e.g.

*Is Rheumatoid Arthritis more common in men or women?*

We treat **CHOICE** questions as a special case because the candidate answers can be directly extracted from the question, and no further answer type prediction is needed. Since there exist an unlimited number of quantitative values which cannot be all covered in the UMLS semantic network, we also add the **QUANTITY** type to complement the existing qnco (Quantitative Concept) type.
Answer type extraction

To identify the gold standard labels for the existing Q/A pairs used for training, we apply UTS to retrieve the semantic types for each gold standard exact answer, where we first use the exact search type, and if no results are returned, we further relax the search type to words. Since UTS may return more than one concept type for each input concept, and each training question may contain more than one gold standard answer variant (these may be synonyms or answer concepts for list questions), only the most frequent concept type is used as the gold standard answer type. If multiple concept types have the same number of occurrences for all gold standard answer variants, we keep all of them as the gold standard labels for the question.

For example, among the 44 gold standard answer variants provided for the question

In which proteins is the chromodomain present?

we can retrieve the following semantic types from UTS:

- {name: Gene or Genome, id: T028, abbr: gngm, count: 19}
- {name: Amino Acid, Peptide, or Protein, id: T116, abbr: aapp, count: 10}
- {name: Biologically Active Substance, id: T123, abbr: bacs, count: 9}
- {name: Nucleic Acid, Nucleoside, or Nucleotide, id: T114, abbr: nnon, count: 1}
- {name: Cell Component, id: T026, abbr: celc, count: 1}
- {name: Quantitative Concept, id: T081, abbr: qnco, count: 1}
- {name: Molecular Function, id: T044, abbr: moft, count: 1}
- {name: Food, id: T168, abbr: food, count: 1}

We assign gngm as the answer type for the question.

We identify 82 out of the 406 factoid and list questions in the 3B development set do not have a single gold standard answer variant for which UTS can provide a semantic type. There are three major reasons for this phenomenon. First, some answer concepts are not included in the UMLS semantic network. For example, the question

Which histone marks are deposited by Set7?

has two answers:

H4K20 monomethylation

and

H3K4 monomethylation,

both of which cannot be mapped to a single semantic type. However, since there are questions which ask about “which histone marks” in the training set and include gold standard answers that can be mapped to a concept type, we can simply exclude the question from the training set. Second, some gold standard exact answers do not strictly follow the representation format. For example, the question

Which enzyme is deficient in Krabbe disease?
has a gold standard answer

Galactocerebrosidase is an enzyme that is deficient in Krabbe disease (also known as globoid-cell leukodystrophy). This leads to accumulation of psychosine (galactosylsphingosine) primarily in oligodendrocytes.

In fact, “Galactocerebrosidase” alone should be the gold standard exact answer. Third, some questions, e.g.

Which is the most important prognosis sub-classification in Chronic Lymphocytic Leukemia?

with a gold standard answer

The mutational status of the IGHV genes.

have an answer which is not a simple biomedical entity, and thus cannot be mapped to a single concept type. For the second two types of phenomena, we must further investigate the linguistic and semantic structure of the answers. Finally, we obtain gold standard labels from UTS for the 324 remaining questions.

Some of the issues that we face with UTS can be solved by the TmTool based concept identification and type identification, as we have discussed in Section 7.4.1 since the latter uses a probabilistic soft matching to identify gene names, rather than a dictionary. As we have expanded the semantic type set of concepts to include TmTool concept types for BioASQ 4B, we also use the TmTool to label the semantic types of the gold standard exact answers, which allows the answer type prediction module to predict these additional semantic types. In particular, we concatenate all the exact answers of each factoid or list question using commas, and send the concatenated string to the TmTool service, instead of each exact answer at a time. For example, if the gold standard exact answer is a list of strings: “NBEAL2”, “GFI1B”, “GATA1”, then a single string “NBEAL2, GFI1B, GATA1” is used as the request text body.

When we prepare the UTS request text for BioASQ 4B, we separated the texts in the parentheses in all gold standard exact answers as synonyms before gold standard semantic type labeling and answer type prediction training, e.g.

Hydrophilic Interaction Chromatography (HILIC)

is converted into

[Hydrophilic Interaction Chromatography, HILIC].

For BioASQ 4B, we also introduce the “null” type for the exact answer texts if neither of the two concept search providers (TmTool or UTS) can identify.

**Feature extraction & classification**

We extract a number of linguistic and semantic features from the tokens and concepts, as detailed in Table 7.4. Motivated by the common structure of choice questions, we include the second and
Table 7.4: Answer type prediction features

<table>
<thead>
<tr>
<th>No.</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>the lemma form of each token</td>
</tr>
<tr>
<td>2</td>
<td>if the question begins with “do” or “be”</td>
</tr>
<tr>
<td>3</td>
<td>if the question contains a token “or”</td>
</tr>
<tr>
<td>4</td>
<td>if the question contains a quantity question phrase</td>
</tr>
<tr>
<td>5</td>
<td>the semantic type of each concept</td>
</tr>
<tr>
<td>6</td>
<td>a (semantic type, dependency label) pair, where we use the dependency label of the head token in the concept bearing phrase as the second element</td>
</tr>
<tr>
<td>7</td>
<td>also a (semantic type, dependency label) pair, where we use the dependency label of the head of the head token in the concept bearing phrase as the second element</td>
</tr>
<tr>
<td>8</td>
<td>the lemma form of the first child of the root in the parse tree that is a noun and has a dependency relation of dep</td>
</tr>
</tbody>
</table>

third groups of features. Since two questions that contain the same concept mentions may still have a different question focus, if the concept mentions have different syntactic roles (subject, objective, modifier, etc.) in the sentence, we also add the sixth and seventh feature groups. The last feature models a simple rule-based lexical answer type extraction approach by identifying the key noun in the sentence (Listing A.2 ll.42–44 and Listing A.3 ll.42–45).

We use the logistic regression trainer from the LibLinear tool [55] to train a multi-class classifier, and use 10-fold cross prediction to predict a list of up to five most likely semantic labels for each question in the training set, which is used in the downstream training process. The prediction module and the pretrained model are integrated into the BioASQ 3B and 4B evaluation pipelines (corresponding to Listing A.2 ll.46–48 and Listing A.3 ll.87–90). The model can correctly identify answer types for most high-frequency sentence patterns, such as “which genes”, but it may fail for low-frequency question patterns, where UTS may not be able to resolve ambiguous cases (e.g. AUS is identified as a country name without the context).

### 7.4.5 Candidate Answer Generation (Developed for BioASQ 3B & 4B)

We first use the same set of token and concept identification tools used for the question (described in Section 7.4.4) to annotate all the relevant snippets provided as input for Phase B (corresponding to Listing A.2 ll.54–76, Listing A.3 ll.52–80, and Listing A.4 ll.47–75). We then integrate four components to generate candidate answers (corresponding to Listing A.2 ll.82–84 and Listing A.3 ll.92–95, and the component level descriptor is presented in Listing A.5).

**Concepts as Candidate Answers.** We create a candidate answer using each concept identified by one of three concept identification approaches described in Section 7.4.4 (corresponding to line 6 of Listing A.5). For BioASQ 3B Batch 3, we also filter out any concept mention that is exactly a stopword, a token or phrase in the question, or a concept that is also annotated in the question. We use a stopword list that combines the most 5,000 frequent English words and the list of Entrez (PubMed) stopwords.
**CHOICE Questions.** We first identify the “or” token in the question (line 4), and then identify its head token, which is most likely the first option in the list of candidate answers. Next, we find all the children of the first option token in the parse tree that have a dependency relation of conj, which are considered to be alternative options. We see this approach works well on most CHOICE questions, but still has problems in a few special cases. First, if two options have different prefixes but the same suffix, the suffix may be discarded in the first option, e.g.

*Is the long non-coding RNA malat-1 up or downregulated in cancer?*

We believe richer semantic interpretation of the question is needed to handle such cases. Another issue is that the head tokens can be semantically incomplete, such that a phrase which covers the head token should be used instead for the options; we extend the boundary of the candidate answer to at minimum a concept mention that covers the candidate answer occurrence (Line 7).

**QUANTITY Questions.** We identify all the tokens that have a POS tag of CD in all relevant snippets (line 5). This approach can reliably produce a complete set of quantitative mentions. However, it does not give us a way to semantically interpret the extracted numbers. For example, it could correctly identify

\[20,687, 24,500, \text{etc.}\]
as candidate numbers, but does not have the ability to “summarize” the numbers and produce a single answer, e.g.

*Between 20,000 and 25,000*

as required. Similar to CHOICE questions, another limitation is that this method can only identify a single token as a candidate answer, e.g.

\[3.0\]

where a semantically complete phrase, e.g.

\[3.0 \text{ mm}\]
is preferred. We apply the same approach used for CHOICE questions to include the CD-bearing phrase as the candidate answer (Line 7).

### 7.4.6 Candidate Answer Scoring (Developed for BioASQ 3B & 4B)

We predict a confidence score for each candidate answer (corresponding to Listing [A.2] ll.90–92 and Listing [A.3] ll.102–105, and the component level descriptor is presented in Listing [A.6]). For BioASQ 3B Batch 3, we use a simple multiplication method to combine the type coercion score and the occurrence count.

In BioASQ 3B Batches 4 and 5, we define a feature space containing 11 groups of features, as shown in Table [7.5], which extend the approach used by Weissenborn et al. [212], and use the logistic regression classifier to learn the scoring function. We only use the questions with non-zero recall for training, where we assign “1” to each candidate answer variant if it is also contained in
Table 7.5: Answer scorers

<table>
<thead>
<tr>
<th>Line</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Type coercion. For each Candidate Answer Occurrence (CAO), the percentage of semantic types that are also among the top-$k$ ($k = 1, 3, \text{ and } 5$) predicted answer types. To accumulate the scores from multiple CAOs, we use “average”, “maximum”, “minimum”, “non-zero ratio”, “one ratio”, and “boolean or”.</td>
</tr>
<tr>
<td>6</td>
<td>CAO count. We use the number of CAOs for each answer variant and we also count the total number of tokens in all occurrences.</td>
</tr>
<tr>
<td>7</td>
<td>Name count. The number of distinct candidate answer names, which differs from CAO count; if two CAOs have the same text string, only one will count.</td>
</tr>
<tr>
<td>8</td>
<td>Avg. covered token count. Averaged number of tokens in each CAO.</td>
</tr>
<tr>
<td>9</td>
<td>Stopword count. For each CAO, we calculate the stop word percentage. We use the same stoplist as described in Section 7.4.5. We accumulate the scores from multiple CAOs using “average”, “minimum”, “one ratio”, and “boolean or”.</td>
</tr>
<tr>
<td>10</td>
<td>Token overlap count. For each CAO, we calculate the percentage of tokens that overlap with the question. We accumulate the scores from multiple CAOs using “average”, “non-zero ratio”, and “boolean or”.</td>
</tr>
<tr>
<td>11</td>
<td>Concept overlap count. For each CAO, we calculate the percentage of covered concept mentions that overlap with the question. We accumulate the scores from multiple CAOs using “average”, “non-zero ratio”, and “boolean or”.</td>
</tr>
<tr>
<td>12</td>
<td>Token proximity. For each CAO, we calculate the averaged distance to the nearest occurrence of each question word in the relevant snippet. We set a window size of 10, and if any question word falls out of the window, we use a fixed distance of 20. We also transform the distance to its negation and inverse, and accumulate the scores from multiple CAOs using “average”, “maximum”, “minimum”, and “non-zero ratio”.</td>
</tr>
<tr>
<td>13</td>
<td>Concept proximity. Similar to token proximity, we calculate the distance from each CAO to each question concept mention in the relevant snippet.</td>
</tr>
<tr>
<td>14</td>
<td>LAT count. For each CAO, we calculate the percentage of tokens that overlap with a LAT token (i.e. the 8th feature in Table 7.4). We accumulate the scores from multiple CAOs using “average” and “non-zero ratio”.</td>
</tr>
<tr>
<td>15</td>
<td>Parse proximity. Similar to token proximity, we use the distance in the parse tree, which is important for list questions, as answer bearing sentences may be in the form of “includes A, B, C, …”</td>
</tr>
</tbody>
</table>
Table 7.6: Collective answer reranking features

<table>
<thead>
<tr>
<th>No.</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>the original score from the answer scoring prediction</td>
</tr>
<tr>
<td>2</td>
<td>min/max/avg token distance between each pair of candidate answer occurrences</td>
</tr>
<tr>
<td>3</td>
<td>min/max/avg Levenshtein edit distance between each pair of candidate answer variant names</td>
</tr>
<tr>
<td>4</td>
<td>min/max/avg number (and percentage) of semantic types that each pair of candidate answers have in common</td>
</tr>
<tr>
<td>5</td>
<td>min/max/avg edit distance between each pair of candidate answer variant names after transformed into their shape forms (i.e. upper-case letters are replaced with ‘A’, lower-case letters are replaced with ‘a’, digits are replaced with ‘0’, and all other characters are replaced with ‘-’.)</td>
</tr>
</tbody>
</table>

the gold standard answer set, and “0” otherwise. Since there are many more negative instances than positive instances, we assign to each negative instance a weight of

\[
\frac{\#\text{positive instances}}{\#\text{negative instances}}.
\]

For BioASQ 4B, we expand the feature space by adding nominal features (e.g. answer type name, concept type name, etc.), in addition to numeric features.

### 7.4.7 Answer Pruning (Used in BioASQ 3B Only)

For BioASQ 3B Batch 3, we use the factoid QA pipeline to produce answers for list questions without any pruning. For BioASQ 3B Batch 4, we use an absolute threshold to select only the answers that have a confidence score, predicted by the candidate answer scoring model, above threshold. Since BioASQ 3B Batch 5, instead of an absolute threshold for all questions, we have used a ratio threshold to filter the answers that have a confidence score above a percentage of the highest predicted score for the question (corresponding to Listing A.2 ll.94–96 and Listing A.3 ll.113–116). We tune the threshold on the development set.

### 7.4.8 Collective Answer Reranking (Developed for BioASQ 4B)

For BioASQ 4B, we employ a collective answer reranking method aiming to boost the low-ranked candidate answers which share the same semantic type with high-ranked candidate answers for list questions, before perform the answer pruning step (corresponding to Listing A.3 ll.107–111, and the component descriptor is presented in Listing A.7). The intuition is that list questions always ask for a list of concepts that have the same properties, which implies that the concepts usually have the same semantic types (e.g. all of them should be gene names). After the answer scoring step where a confidence score is assigned to each candidate answer individually, we can imagine the top candidate answers might have mixed types. For example, in a situation where the second
Table 7.7: Yes/No question answering features

<table>
<thead>
<tr>
<th>No.</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>“contradictory” concept count in the relevant snippets</td>
</tr>
<tr>
<td>2</td>
<td>overlapping token count in the relevant snippets</td>
</tr>
<tr>
<td>3</td>
<td>expected answer count in the relevant snippets</td>
</tr>
<tr>
<td>4</td>
<td>sentiment analysis via positive and negative word count of each relevant snippet</td>
</tr>
<tr>
<td>5</td>
<td>negation word count of each relevant snippet</td>
</tr>
<tr>
<td>6</td>
<td>question inversion</td>
</tr>
</tbody>
</table>

answer is a disease, but the rest of the top-5 answers are all gene names, we should expect that the second answer should be down-ranked.

We use the same labels used for training the candidate answer scoring model, but incorporate features that measure how similar each answer is to the other top-ranked answers, which are detailed in Table 7.6. The token distance counts the number of intermediate tokens between the candidate answer tokens in the snippet text, e.g.

Yamanaka factors, Oct3/4, Sox2, Klf4, and c-Myc, ...

The Levenshtein edit distance measures the morphological similarities between the answer texts, e.g.

fludarabine, cytarabine, amsacrine

and so does the shape edit distance, which measures the edit distance after have them transformed into the shape forms, e.g. the original answer texts are

Oct3/4, Sox2, Klf4, c-Myc

then the shape edit distance scorer uses the following text

Aaa0.0, Aaa0, Aaa0, a.Aaa

and the common semantic type count score “promotes” the candidate answers that have a large number of semantic types in common with the top ranked answers.

For each candidate answer, we calculate a feature value, according to Table 7.6, for each other candidate answer in the input candidate list, and then we calculate the max/min/avg value corresponding to the top-k candidate answers. We use 1, 3, 5, 10 for k, and use the logistic regression classifier to train a binary classifier by down-sampling the negative instances to balance the training set. In addition to list questions, we also apply the method to factoid questions.

7.4.9 Learning to Answer Yes/No Questions (Developed for BioASQ 4B)

We consider the yes/no question answering problem as a binary classification problem, which allows to prioritize, weight, and blend multiple pieces of evidence from various approaches using a supervised framework. We list the sources of evidence (features) integrated into the system. The component descriptor is presented in Listing A.8 and the complete yes/no QA pipeline is presented in Listing A.4.
“Contradictory” concept. First, we hypothesize that if a statement is wrong, then the relevant snippets should contain some statements that are contradictory to the original statement, with some mentions of “contradictory” concepts or “antonyms”. For example, the question is

Does HER2 under-expression lead to favorable response to trastuzumab?

and a relevant snippet that says

over-expression of HER2 is reported ...

is an indicator of “contradictory”. Another example is that the question

Are chromomethylases present in animal genomes?

has a relevant snippet

Dual binding of chromomethylase domains to H3K9me2-containing nucleosomes directs DNA methylation in plants.

To identify pairs of contradictory concepts or antonyms is difficult given the resources that we have. Instead, we try to identify all the different concepts in the snippets that have the same semantic type as each concept in the original statement. For a given concept type, the more the unique concepts are found in both question and relevant snippets, or the less the concepts in the questions are found in the snippets, the more likely the original statement is wrong.

Formally, for a concept type $t$, we calculate a “contradictory” score as follows:

$$\frac{\sum_{s \in S} \sum_{c \in s} \left[ \text{type}(c) = t \right]}{\sum_{c \in q} \left[ \text{type}(c) = t \right] + \sum_{s \in S} \sum_{c \in s} \left[ \text{type}(c) = t \right]}$$

where $S$ is the set of snippets, $q$ is the question, $c$ is a concept mention, and $\left[ \text{type}(c) = t \right]$ takes 1 if the concept $c$ is type $t$ and 0 otherwise. We derive the aggregated contradictory score from the concept type level scores using max/min/average statistics. We calculate a number of similar statistics to estimate how likely each snippet contradicts the original statement.

Overlapping token count. In case the concept identification modules fail to identify important concepts in either the original questions or relevant snippets, we also consider the difference of token mentions between the original question and the relevant snippets, instead of concepts.

Expected answer count. Not all concepts and tokens are equally important in the original questions. We find that many times the focus of a yes/no question is the last concept mention, which we denote as the expected answer. We count the frequency (and the percentage) that the expected answer is mentioned in the relevant snippets, as well as the frequency that concepts of the same type are mentioned.

Positive / negative / negation word count. Sometimes, an explicit sentiment is expressed in the relevant snippets to indicate how confident the author believes a statement is true or false. For example, the question

Can NXY-059 be used for treatment of acute ischemic stroke patients?
has two relevant snippets that both have negative words:

NXY-059, ..., has demonstrated neuroprotection in several animal models of acute ischemic stroke but failed to confirm clinical benefit in the second phase III trial (SAINT-II).

and

NXY-059 is no longer in development following a lack of efficacy found in a Phase III trial in patients with acute ischemic stroke.

We use a simple dictionary based method for sentiment analysis, and we count how many times each positive / negative word is mentioned in each snippet, then aggregate across the snippets using min/max/average. We also use a list of common English negation words for negation detection, for simplicity. Intuitively, a high negative or negation count indicates that the original statement tends to be incorrect.

Question inversion. The question inversion method answers a yes/no question by first converting it to a factoid question, then applies an existing factoid question answering pipeline to generate a list of alternate candidate answers, and finally evidences and ranks each candidate answer. If the expected answer in the original question is also ranked at the top among all candidates for the factoid question, then the statement is true.

In our system, we first assume the last concept mention corresponds to the expected answer. Therefore, its concept type(s) are also the answer type(s) of the factoid question, and all the synonyms of the concept are the answer variants. After the token(s) and concept mention(s) covered by the expected answer are removed from the original question and the question type is changed to FACTOID, we use the candidate answer generation and scoring pipeline for the factoid QA to generate and rank a list of candidate answers. Since annotating additional texts is computationally expensive, we do not retrieve any relevant snippets for the converted factoid questions, instead we only use the relevant snippets of the original yes/no questions (provided as part of the Phase B input). The rank and the score of the expected answer are used as question inversion features for yes/no question training.

We use a number of classifiers, e.g. Logistic Regression, Classification via Regression, Simple Logistic using LibLinear and Weka tools, after we down-sampled the positive (“yes”) instances. In Section 7.5, we report not only the performance of each method in terms of accuracy, but also accuracy on the “yes” question sub set and the “no” question sub set, since on an imbalanced data set, a simple “all-yes” method is also a “strong” baseline.

7.5 Preliminary Results & Analysis

We follow the conventional software development process to integrate the analysis components described in Section into a biomedical question answering system using the ECD description

http://www.cs.uic.edu/~liub/FBS/opinion-lexicon-English.rar
http://www.enchantedlearning.com/wordlist/negativewords.shtml
http://www.cs.waikato.ac.nz/ml/weka/
framework, which include manual comparison of options. Before we conduct a full configuration space exploration experiment in next section, we report the preliminary results and analysis from our initial development cycle and official participation in the competition. In Section 7.5.1 we report the official evaluation results in BioASQ 3B and 4B, which is followed by two error analysis studies for factoid QA and yes/no QA in Sections 7.5.2 and 7.5.3 respectively. Finally in Section 7.5.4 we report the comparison results of our systems that participated in BioASQ 3B and 4B challenges using the same subset from the development set.

### 7.5.1 Official Evaluation Results

We participated in the official evaluations of BioASQ 3B (in 2015) and 4B (in 2016) for a total of 6 test batches, focusing on the factoid, list QA, and yes/no QA tasks, but also submitted to relevant content retrieval tasks. Over the 70 days (between April 2 to Jun 10 2015) of intensive development for BioASQ 3B participation, our experiment database has recorded 717 experiments. Among the 669 successful experiments, there were 167 executing the training pipeline (177.5 topics per run on average), 422 executing the testing pipeline (24.1 topics per run on average) and 80 “dummy” runs used to cache service results (284.5 topics per run on average). We summarize the official evaluation results of relevant document and snippet retrieval in Batches 3, 4, and 5 of BioASQ 3B Phase A, factoid and list QA in Batches 3, 4, and 5 of BioASQ 3B and 4B Phase B, and yes/no QA in Batches 3, 4, and 5 of BioASQ 4B Phase B from the official evaluation portal in Table 7.8. The test descriptors used for Batches 5 in BioASQ 3B and 4B are shown in Listings A.1 to A.4.

Among all the systems that participated in BioASQ 3B Phase A evaluation, the performance of our document retrieval pipeline is scored at the bottom of the first tier. The absolute performance gaps between our pipeline and the system that is scored one place behind ours in Batches 3, 4, and 5 are measured as .0915, .0225, and .0869 respectively in terms of MAP, which are larger than those between our pipeline and the best-performing system (.0435, .0204, and .0466 respectively). Due to a relatively steep learning curve for the developers who have not had much experience with the system and the task, Phase A system used a different question analysis pipeline from the Phase B system, where the former integrated no concept retrieval module. Therefore, we believe document and snippet retrieval evaluated in Phase A can be further improved by considering synonyms expanded using UTS during query formulation. Moreover, the snippets extracted by the latter snippet retrieval stage can be fed back to the search engine as an expanded query to harvest more relevant information; reinforcement learning can thus be utilized in this scenario.

In the factoid and list QA tasks of BioASQ 3B Phase B, the official evaluation results indicated that our system achieved MRR scores of .1615, .5155, and .2727 for factoid questions, and F-measure score of .0969, .3168, and .1875 for list questions; five of these results are the highest scores reported among all participating systems. In BioASQ 4B Phase B, the official evaluation results again showed that our system achieved MRR scores of .2436, .3253, and .2854 for factoid questions, and F-measure score of .4828, .3115, and .2897 for list questions; all of these results are the highest scores among all participating systems. For the yes/no QA task, accuracy is used as the official evaluation metric, in fact an inaccurate measurement for imbalanced “yes”/“no” answers. We find our yes/no QA system achieved the highest performance among all nontrivial runs (those that do not use “all-yes” strategy). We notice that the performance in BioASQ 3B Batch 4 is higher.
Table 7.8: Partial official BioASQ 3B abd 4B evaluation results and ranks among systems (shown in the parentheses).

### Relevant document retrieval (BioASQ 3B Phase A)

<table>
<thead>
<tr>
<th>Batch</th>
<th>Precision</th>
<th>Recall</th>
<th>F-measure</th>
<th>MAP</th>
<th>GMAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>3rd</td>
<td>.2310 (15)</td>
<td>.3242 (15)</td>
<td>.2311 (15)</td>
<td>.1654 (15)</td>
<td>.0136 (15)</td>
</tr>
<tr>
<td>4th</td>
<td>.2144 (15)</td>
<td>.3320 (15)</td>
<td>.2263 (15)</td>
<td>.1524 (15)</td>
<td>.0081 (14)</td>
</tr>
<tr>
<td>5th</td>
<td>.2130 (15)</td>
<td>.4474 (15)</td>
<td>.2605 (15)</td>
<td>.1569 (15)</td>
<td>.0267 (8)</td>
</tr>
</tbody>
</table>

### Relevant snippet retrieval (BioASQ 3B Phase A)

<table>
<thead>
<tr>
<th>Batch</th>
<th>Precision</th>
<th>Recall</th>
<th>F-measure</th>
<th>MAP</th>
<th>GMAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>3rd</td>
<td>.1133 (3)</td>
<td>.1044 (5)</td>
<td>.0891 (3)</td>
<td>.0892 (1)</td>
<td>.0013 (5)</td>
</tr>
<tr>
<td>4th</td>
<td>.1418 (5)</td>
<td>.1264 (10)</td>
<td>.1153 (8)</td>
<td>.0957 (5)</td>
<td>.0027 (6)</td>
</tr>
<tr>
<td>5th</td>
<td>.1472 (9)</td>
<td>.1756 (9)</td>
<td>.1391 (9)</td>
<td>.1027 (9)</td>
<td>.0040 (5)</td>
</tr>
</tbody>
</table>

### Factoid and list QA (BioASQ 3B Phase B)

<table>
<thead>
<tr>
<th>Batch</th>
<th>Strict Acc.</th>
<th>Factoid Acc.</th>
<th>MRR</th>
<th>Precision</th>
<th>List Acc.</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>3rd</td>
<td>.1154 (1)</td>
<td>.2308 (1)</td>
<td>.1615 (1)</td>
<td>.5396 (1)</td>
<td>.6933 (1)</td>
<td>.0969 (7)</td>
</tr>
<tr>
<td>4th</td>
<td>.4483 (1)</td>
<td>.6207 (1)</td>
<td>.5155 (1)</td>
<td>.5494 (1)</td>
<td>.3480 (1)</td>
<td>.3168 (1)</td>
</tr>
<tr>
<td>5th</td>
<td>.2273 (1)</td>
<td>.3182 (1)</td>
<td>.2727 (1)</td>
<td>.2573 (5)</td>
<td>.1875 (1)</td>
<td></td>
</tr>
</tbody>
</table>

### Factoid and list QA (BioASQ 4B Phase B)

<table>
<thead>
<tr>
<th>Batch</th>
<th>Strict Acc.</th>
<th>Factoid Acc.</th>
<th>MRR</th>
<th>Precision</th>
<th>List Acc.</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>3rd</td>
<td>.2308 (1)</td>
<td>.2692 (1)</td>
<td>.2436 (1)</td>
<td>.5008 (6)</td>
<td>.4828 (1)</td>
<td></td>
</tr>
<tr>
<td>4th</td>
<td>.2903 (1)</td>
<td>.3871 (1)</td>
<td>.3253 (1)</td>
<td>.5494 (1)</td>
<td>.3115 (1)</td>
<td></td>
</tr>
<tr>
<td>5th</td>
<td>.2121 (1)</td>
<td>.3939 (1)</td>
<td>.2854 (1)</td>
<td>.4170 (1)</td>
<td>.2897 (1)</td>
<td></td>
</tr>
</tbody>
</table>

### Yes/no QA (BioASQ 4B Phase B)

<table>
<thead>
<tr>
<th>Batch</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>3rd</td>
<td>.5200 (6 or 1 among all nontrivial runs)</td>
</tr>
<tr>
<td>4th</td>
<td>.6667 (5 or 1 among all nontrivial runs)</td>
</tr>
<tr>
<td>5th</td>
<td>.7407 (8 or 1 among all nontrivial runs)</td>
</tr>
</tbody>
</table>
than in other batches, which we believe is because Batch 4 set contains more questions seeking for
the types of answers that have occurred more frequently in the training set, e.g. gene, disease, etc.

7.5.2 Error Analysis for Factoid QA

To further understand what causes the error and how we may improve the system, we manually
answer each factoid question in BioASQ 3B Batches 3, 4, and 5 using the gold standard snippets
provided for the input of Phase B, and compare with the output of our system to label the error
types (multiple types allowed) for each incorrectly answered question. We list the error categories
and give definition and examples to each category in Table 7.9 where we also show the occurrence
of each error category in each test batch. Based on the analysis, we believe a better concept
identification model and concept type prediction model will make the hugest impact to the overall
performance improvement.

Table 7.9: Error categories and occurrences for factoid questions in BioASQ 3B test batches 3, 4,
and 5.

<table>
<thead>
<tr>
<th>Error category</th>
<th>Batch</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3rd 4th 5th</td>
</tr>
<tr>
<td>Concept type identification/answer type prediction</td>
<td>9</td>
</tr>
<tr>
<td>The highest ranked answer has a different concept type from the answer type that question asks for, which may be caused by a wrongly predicted answer type, an incorrect score combination equation from the score prediction model, or the concept identification module.</td>
<td></td>
</tr>
<tr>
<td>Concept identification</td>
<td>4</td>
</tr>
<tr>
<td>Some answer variants are not identified as concepts or we can find little evidence from the relevant snippets for the concept. For example, for the question Neurostimulation of which nucleus is used for treatment of dystonia? none of the components is able to identify Bilateral globus pallidus internus (GPi) as a concept and further candidate answer variant.</td>
<td></td>
</tr>
<tr>
<td>Complex answer</td>
<td>2</td>
</tr>
</tbody>
</table>

Continued on next page
Table 7.9 – continued from previous page

<table>
<thead>
<tr>
<th>Error category</th>
<th>Batch</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3rd 4th 5th</td>
</tr>
</tbody>
</table>

The ideal answer is a complex phrase or sentence, rather than a single-entity concept, usually in response to the questions containing effect, role, function, etc. For example,

executors/mediators of apoptosis

should be extracted to answer the question

What is the function of caspases?

but we only see “apoptosis” in the candidate answer list.

**Mistakenly use question phrase as answer**

3 2 2

Although we design a scorer in the ranking module to identify whether each candidate answer co-occurs in the original question, which should lower the rank of those candidate answers, we still see some question phrase variants are chosen as the top answer. For example, the question

What is the effect of enamel matrix derivative on pulp regeneration?

mentions a concept

enamel matrix derivative

but the system ranks its acronym “EMD” at the top.

**Tokenization**

2 4 0

Tokenization module may fail if the concept contains punctuation marks, e.g. parentheses, colon, semicolon, etc, and/or numbers, as in the example

t(11;22)(q24:q12)

**Definition question**

2 0 1

The asker knows the terminology but asks for the definition, e.g.

What is Piebaldism?

or knows the properties and asks for terminology, e.g.

How are ultraconserved elements called when they form clusters?

We believe we need to introduce special question types and modules.

**Question type**

1 0 1

Continued on next page
Table 7.9 – continued from previous page

<table>
<thead>
<tr>
<th>Error category</th>
<th>Batch</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3rd</td>
</tr>
</tbody>
</table>

Identification of QUANTITY and CHOICE questions may fail in some cases. For example,

Alpha-spectrin and beta-spectrin subunits form parallel or antiparallel heterodimers?

does not use “Do” at the beginning. Another example is that “risk” is a QUANTITY indicator in the question

What is the risk of developing acute myelogenous leukemia in Fanconi anemia?

Snippets that have no information

Some snippets do not contain any answer variant. For example,

What is the main role of Ctf4 in dna replication?

has a gold standard snippet

Ctf4 remains a central player in DNA replication.

Relation concept identification

A relation concept refers to a verb or verbal adjective, e.g.

responsible or leading

that distinguishes the expected answer from other candidates that have the same concept type.

Syntactic function

The key to answer the question is embedded in the syntactic structures of the relevant snippets. For example, in the snippet

Medicarpin, the major phytoalexin in alfalfa, is synthesized via the isoflavonoid branch of phenylpropanoid metabolism.

no explicit relation word is used between “Medicarpin” and “the major phytoalexin”, but the syntactic structure clearly implies that the latter explains the former.

7.5.3 Error Analysis for Yes/No QA

From Section [7.5.1] we see that, despite integration of various sources of evidence, the current yes/no question answering system is still unreliable. We conducted a manual analysis of our yes/no question answering method using BioASQ 4B data set based on our own judgment of yes or no, which may not be consistent with the gold standard.

We found the BioASQ 4B data set is more imbalanced than the development data set, where we only identified six questions from all five test batches that have a “no” answer. We applied the proposed yes/no QA method to the six questions. Among these questions, three are correctly predicted (namely, “Is macitentan an ET agonist?”, “Does MVIIA and MVIIC bind to the same
calcium channel?” and “Is the abnormal dosage of ultraconserved elements disfavored in cancer cells?”), and the answers to the other three questions are wrong. We conduct an error analysis for the false positive predictions.

The first false positive question is

Are adenylyl cyclases always transmembrane proteins?

The key to this question is the recognition of the contradictory concept pair

transmembrane and soluble

or

transmembrane adenylyl cyclase (tmAC) and soluble AC.

This requires first correctly identifying both terms as biomedical concepts and then assigning correct semantic type labels to them, where the latter can only be achieved using MetaMap and TmTool. MetaMap correctly identified “transmembrane proteins” in the question and assigned a semantic label of “Amino Acid, Peptide, or Protein”, and identified “soluble adenylyl cyclase” in the snippet and assigned a semantic label of “Gene or Genome”. Due to the mismatch of semantic types “Amino Acid, Peptide, or Protein” and “Gene or Genome”, the system fails to recognize the contradiction.

In fact, we found that the same problem also happened during the answer type prediction and answer scoring steps, e.g. the question may be predicted to ask for a “Gene or Genome”, but the candidate answer is often labeled as “Amino Acid, Peptide, or Protein” by MetaMap/UTS. Because of the interchangeable use of “Amino Acid, Peptide, or Protein” and “Gene or Genome” terms, we might consider to treat them as one type. Moreover, the universal quantifier “always” also plays an important role, in contrast to a question with an existential quantifier such as “sometimes”, which the current system has not captured yet. However, this is not the main reason of the failure, since we assume the relevant snippets will rarely mention “soluble AC” if the question asks for whether “transmembrane” exists.

The second false positive question is

Can chronological age be predicted by measuring telomere length?

This should be an easy one, because we can find a negation cue “cannot” in the snippet

telomere length measurement by real-time quantitative PCR cannot be used to predict age of a person

The system integrates two types of negation cue related features: the negation cue count and the existence of a particular negation cue. We found the system correctly identified and counted the negation cue. Therefore, we suspect the classifier did not optimize the combination of features. Furthermore, we need to observe whether our hypothesis that the gold standard answer (yes or no) is strongly correlated with the negation word occurrence in the relevant snippets is true using the development set.

The third false positive question is

Does the 3D structure of the genome remain stable during cell differentiation?

195
The key to this question is the word “stable”, which requires biomedical, esp. genomics, knowledge to understand what “stable” means in the context of genome structure. The word “stable” is mentioned in one of the snippets:

the domains are stable across different cell types,

which however does not answer the question. Useful contradictory keywords that we find in the relevant snippets include “reorganization”, “alteration”, “remodelling”, etc. MetaMap/UTS identified “stable” as a concept of semantic type “Qualitative Concept”, whereas it labeled “reorganization” as a “Idea or Concept” and missed “alternation” and “remodelling”. It suggests that our contradictory concept based method works the best if the focus is factoid (entities), but the current knowledge base can hardly support identification of contradictory properties or behaviors.

The cases for false negative questions are more diverse, which makes it more difficult to find the causes of failures. One reason is that some snippets contain multiple sentences or clauses, and only one is crucial to answer the question, while others can negatively influence the results. For example, the snippet

OATP1B1 and OATP1B3-mediated transport of bilirubin was confirmed and inhibition was determined for atazanavir, rifampicin, indinavir, amprenavir, cyclosporine, rifamycin SV and saquinavir.

has two clauses, but the second one (“and inhibition...”), although is not relevant to the question, introduces other chemical names that confuse the classifier. Another problem is lack of understanding of specificity and generality between concepts, e.g. “encephalopathy” in the question is considered a different concept from “Wernicke encephalopathy” mentioned in the snippets, both belonging to the same disease category. The classifier incorrectly believed another disease name is mentioned to contradict the statement.

We found that yes/no questions are more difficult to answer than factoid and list questions, since there can be many different ways to support or oppose a statement. Although the problem can be simply viewed as a binary classification problem, due to the fact that a limited number of relevant snippets are provided, simple token or phrase level retrieval and statistics can hardly solve the problem. Instead, we believe that reliably answering yes/no questions requires deeper linguistic and semantic understanding of the questions and relevant snippets, which includes leveraging semantic networks of concepts to identify antonyms, hypernyms, and hyponyms, and utilizing dependency relations between the concepts, as well as sentiment analysis of the facts.

### 7.5.4 Manual Comparison of Configurations

We made a few improvements for BioASQ 4B, which include a standardized relevant content reranking module, additional nominal features for answer scoring, collective reranking, and yes/no QA pipeline. We manually compared these methods using the same BioASQ 3B Batch 5 data set to decide which method to use while we prepared our system for BioASQ 4B participation. In this section, we report the manual comparison results.
We first compare the retrieval results (Phase A) in Table 7.10. We can see that the proposed retrieval result reranking method via logistic regression improves the performance of concept and document retrieval, but not snippet retrieval, which may be due to the fact that the input candidate snippets have been reranked using a similar set of features at the document reranking step, and no further information is provided during the subsequent snippet reranking step.

Factoid and list QA results (Phase B) are reported in Table 7.11. We see that the best configuration for factoid question answering in terms of MRR is keeping the original feature set with no collective reranking. However, if additional features are used, then the collective reranking method can improve the performance, and achieve the highest lenient accuracy score.

To answer list questions, we tune the thresholds (absolute threshold or ratio threshold) and report the results from the thresholds that maximize the F1 score. Although the best F1 score is achieved by incorporating additional features without collective reranking and using a ratio-based pruning method, all other configurations without collective reranking have the lowest performance. In addition, we can see that additional features improve the performance in general, and after carefully tuning of the threshold and the ratio in the pruning step, we can achieve the same level of performance. We hypothesize that the collective reranking method combined with ratio-based thresholding can re-normalize the answer scores and is thus more robust than the baseline system that uses no collective reranking combined with an absolute threshold, in the sense that the performance of the former approach is less sensitive to the predefined threshold, although the latter can sometimes outperform the former when the threshold is carefully tuned. We submitted two runs in BioASQ 4B Batch 5 evaluation: oqa-3b-5 and oqa-3b-5-e for the proposed and baseline methods respectively (using the same thresholds), and initial evaluation result confirms our hypothesis.

Due to the imbalance between “yes” and “no” questions, we report the mean negative and positive accuracy scores in addition to the overall accuracy for yes/no question answering. We can see the performance is very sensitive to the choice of the classifier. Using the same set of features, ClassificationViaRegression achieves the highest performance, with both negative and
### Table 7.11: Manual comparison of factoid, list and yes/no QA methods

#### Factoid QA (BioASQ 3B Batch 5)

<table>
<thead>
<tr>
<th>Method</th>
<th>Lenient Acc.</th>
<th>MRR</th>
<th>Strict Acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original features</td>
<td>.5000</td>
<td>.3843</td>
<td>.3182</td>
</tr>
<tr>
<td>Original features + Collective reranking</td>
<td>.4545</td>
<td>.3791</td>
<td>.3182</td>
</tr>
<tr>
<td>Additional features + Collective reranking</td>
<td>.5455</td>
<td>.3732</td>
<td>.2727</td>
</tr>
<tr>
<td>Additional features</td>
<td>.5000</td>
<td>.3689</td>
<td>.2727</td>
</tr>
</tbody>
</table>

#### List QA (BioASQ 3B Batch 5)

<table>
<thead>
<tr>
<th>Method</th>
<th>F1</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Additional features + Ratio pruning</td>
<td>.4291</td>
<td>.4449</td>
<td>.4593</td>
</tr>
<tr>
<td>Additional features + Collective reranking + Ratio pruning</td>
<td>.4246</td>
<td>.4045</td>
<td>.4864</td>
</tr>
<tr>
<td>Additional features + Collective reranking + Absolute pruning</td>
<td>.3969</td>
<td>.4100</td>
<td>.4267</td>
</tr>
<tr>
<td>Original features + Collective reranking + Absolute pruning</td>
<td>.3704</td>
<td>.4231</td>
<td>.3645</td>
</tr>
<tr>
<td>Original features + Collective reranking + Ratio pruning</td>
<td>.3629</td>
<td>.3654</td>
<td>.3874</td>
</tr>
<tr>
<td>Additional features + Absolute pruning</td>
<td>.3463</td>
<td>.3840</td>
<td>.3677</td>
</tr>
<tr>
<td>Original features + Ratio pruning</td>
<td>.3460</td>
<td>.3188</td>
<td>.4431</td>
</tr>
<tr>
<td>Original features + Absolute pruning</td>
<td>.1461</td>
<td>.2639</td>
<td>.1183</td>
</tr>
</tbody>
</table>

#### Yes/no QA (BioASQ 3B Batch 5)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ClassificationViaRegression</td>
<td>.7143</td>
<td>.7778</td>
<td>.6842</td>
</tr>
<tr>
<td>SimpleLogistic</td>
<td>.7143</td>
<td>.4444</td>
<td>.8421</td>
</tr>
<tr>
<td>All-Yes</td>
<td>.6786</td>
<td>.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>Logistic regression</td>
<td>.5357</td>
<td>.2222</td>
<td>.6842</td>
</tr>
</tbody>
</table>

### 7.6 Configuration Space Exploration

In previous sections, we select models, parameter values, algorithms based on a developer’s “intuition” as well as mostly manual comparison. In this section, we use the proposed budgeted policy optimization strategies to explore the configuration spaces defined and constructed for the biomedical question answering tasks. In particular, we focus on two specific tasks – relevant snippet retrieval and factoid question answering.

When we conduct the simulated experiments in Sections 5.5 and 5.7, we construct simulated analysis components whose performance follows a predefined beta distribution, allowing us to compare the best policy discovered by the (budgeted) policy optimization strategy with the known positive accuracy scores greatly above 0.5 (random). All other methods tend to predict “yes”, which results in a high positive accuracy but a low (below 0.5) negative accuracy score.
optimal policy. However, in real world applications, such as the biomedical question answering task, we have no “gold standard” optimal policy. Even if we can evaluate all the possible combinations given a fixed number of inputs, we are still unable to claim the best policy is optimal for unseen inputs. To overcome this issue, we simulate a development process for BioASQ 4B following the analytics meta learning methodology. In particular, we develop and train the analysis components using all the questions and answers except the last batch (BioASQ 3B Batch 5) in the development set prepared for BioASQ 4B, which was made available in December 2015, and used the BioASQ 3B Batch 5 subset for policy optimization. Then, we pretend that we are participating in the BioASQ 4B Batch 1 official evaluation, and submit the optimal system(s) returned by the policy optimization strategies. We report the evaluation results using the development set prepared for BioASQ 5B, which contains the gold standard answers for the BioASQ 4B Batch 1 subset, and was available in December 2016. This allows us to compare the strategies and understand the learning progress of each strategy, although it does not necessarily indicate the true “optimality” of the policy.

7.6.1 Phase A: Relevant Snippet Retrieval

In this subsection, we first describe the configuration space for the relevant snippet retrieval task and report the configuration space exploration results.

Configuration Space

First, we integrate two pretrained models for the same ClearNLP parser, including bioinformatics and medical. Then, in Phases 2 to 8, we integrate all the concept identification components, including the Apache OpenNLP based NP and NPPPNP taggers, which were removed from the official evaluation in BioASQ 4B, and we choose to use or not to use (noop) for each component, which makes it effectively a power set of all the different components. We mainly vary the abstract query construction phase (Phase 10) and retrieval phases (Phases 12 and 15). The options include the stopword list, whether we should explicitly specify the customized term weight, whether we should use the identified concepts and/or tokens for query construction, whether we should select the tokens in certain POS tagsets, which query string constructor we should use, which sentence model we should use for sentence splitting, the hit size, etc. We list the components and configurations that are used in our experiment in Table 7.12. The resulting configuration space contains 93,772,800 combinations.

We use Psg2MAP (passage2 average precision) [79], an evaluation metric proposed for TREC Genomics task that ignores redundant characters when measuring the task performance, which is considered fairer than the normal traditional PsgMAP metric. We define the runtime performance as the time duration that the component spends during each execution. Given the size of the configuration space, we run each of the 40 budgeted policy optimization strategies that are proposed in Section 5.7 for up to 1 hour, and each is repeated 5 times. We deploy each exploration process on a separate AWS EC2 instance of r4.large type (2 Intel Xeon E5-2686 v4 processors and 15.25 GB of DDR4 memory). In our preliminary experiment, we find that the document retrieval phase (Phase 12) takes the most time. In fact, the Lucene retrieval executor may take up to 1 minute
### Table 7.12: Description of configuration space for relevant snippet retrieval

<table>
<thead>
<tr>
<th>Phase #</th>
<th>Phase name / options</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>question-parse</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>model: [clearnlp-bioinformatics, clearnlp-medical]</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>question-concept-metamap</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>question-concept-tmttool</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>question-concept-lingpipe-genia</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>question-concept-lingpipe-genetag</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>question-concept-opennlp-np</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>question-concept-opennlp-nppnp</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>concept-search-uts</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>concept-merge</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>use-name: [false, true] + noop</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>abstract-query-primary</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>abstract-query-secondary</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>concept: {use-weight: [false, true] × required: [false, true]} + noop</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>document-retrieval</td>
<td>11</td>
</tr>
<tr>
<td>13</td>
<td>document-rerank</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>liblinear, logreg, noop</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>passage-retrieval</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>passage-filter</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>noop + lucene-sentence</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>passage-rerank</td>
<td>2</td>
</tr>
</tbody>
</table>

for a complex structured query, and the other two components even require accessing remote Web services. Therefore, we also set a time-out threshold of 20 seconds for this phase. We report the mean (and standard error in plots) and the significant level over the comparand from t-test.

#### Comparison of learning performance of (budgeted) learning strategies

We report the learning performance of budgeted policy optimization strategies using an independent component selection in Table 7.13 and Figures 7.4, 7.5, and 7.6.

We first see that the evaluation results are much lower than those reported for the simulated experiments, which indicate either the relevant snippet retrieval task is harder, i.e. the input mean is lower, and/or the components integrated into the configuration space have lower quality, i.e.
Table 7.13: Results of budgeted learning strategies with independent component selection for BioASQ Phase A (relevant snippet retrieval task)

<table>
<thead>
<tr>
<th>Strategy</th>
<th>1m</th>
<th>10m</th>
<th>20m</th>
<th>30m</th>
<th>40m</th>
<th>50m</th>
<th>60m</th>
<th>Comparand</th>
</tr>
</thead>
<tbody>
<tr>
<td>TQ-ǫG/R</td>
<td>.0232</td>
<td>.0422</td>
<td>.0732</td>
<td>.0747</td>
<td>.0761</td>
<td>.0694</td>
<td>.0543</td>
<td>-</td>
</tr>
<tr>
<td>TQ-ǫG/D</td>
<td>.0176</td>
<td>.0173</td>
<td>.0253</td>
<td>.0492</td>
<td>.0461</td>
<td>.0490</td>
<td>.0554</td>
<td>TQ-ǫG/R</td>
</tr>
<tr>
<td>TQ-OFF/R</td>
<td>.0078</td>
<td>.0122</td>
<td>.0142</td>
<td>.0280</td>
<td>.0291</td>
<td>.0455</td>
<td>.0588</td>
<td>TQ-ǫG/R</td>
</tr>
<tr>
<td>TQ-OFF/D</td>
<td>.0082</td>
<td>.0135</td>
<td>.0162</td>
<td>.0141</td>
<td>.0258</td>
<td>.0267</td>
<td>.0236</td>
<td>TQ-OFF/R</td>
</tr>
<tr>
<td>TQ-UCBE/R</td>
<td>.0285</td>
<td>.0272</td>
<td>.0191</td>
<td>.0342</td>
<td>.0234</td>
<td>.0472</td>
<td>.0727</td>
<td>TQ-ǫG/R</td>
</tr>
<tr>
<td>TQ-UCBE/D</td>
<td>.0079</td>
<td>.0321</td>
<td>.0353</td>
<td>.0479</td>
<td>.0386</td>
<td>.0418</td>
<td>.0563</td>
<td>TQ-UCBE/R</td>
</tr>
<tr>
<td>TQ-J</td>
<td>.0345</td>
<td>.0302</td>
<td>.0249</td>
<td>.0178</td>
<td>.0176</td>
<td>.0433</td>
<td>.0453</td>
<td>TQ-ǫG/R</td>
</tr>
<tr>
<td>FA-ǫG/R</td>
<td>.0058</td>
<td>.0272</td>
<td>.0287</td>
<td>.0309</td>
<td>.0282</td>
<td>.0396</td>
<td>.0548</td>
<td>TQ-ǫG/R</td>
</tr>
<tr>
<td>FA-ǫG/D</td>
<td>.0098</td>
<td>.0171</td>
<td>.0466</td>
<td>.0649</td>
<td>.0523</td>
<td>.0485</td>
<td>.0302</td>
<td>FA-ǫG/R</td>
</tr>
<tr>
<td>FA-OFF/R</td>
<td>.0088</td>
<td>.0228</td>
<td>.0356</td>
<td>.0163</td>
<td>.0392</td>
<td>.0399</td>
<td>.0385</td>
<td>FA-ǫG/R</td>
</tr>
<tr>
<td>FA-OFF/D</td>
<td>.0136</td>
<td>.0286</td>
<td>.0233</td>
<td>.0393</td>
<td>.0304</td>
<td>.0222</td>
<td>.0238</td>
<td>FA-OFF/R</td>
</tr>
<tr>
<td>FA-UCBE/R</td>
<td>.0128</td>
<td>.0357</td>
<td>.0365</td>
<td>.0327</td>
<td>.0348</td>
<td>.0335</td>
<td>.0557</td>
<td>FA-ǫG/R</td>
</tr>
<tr>
<td>FA-UCBE/D</td>
<td>.0220</td>
<td>.0255</td>
<td>.0113</td>
<td>.0268</td>
<td>.0151</td>
<td>.0216</td>
<td>.0206</td>
<td>FA-UCBE/R</td>
</tr>
<tr>
<td>FA-J</td>
<td>.0213</td>
<td>.0295</td>
<td>.0194</td>
<td>.0389</td>
<td>.0399</td>
<td>.0492</td>
<td>.0465</td>
<td>FA-ǫG/R</td>
</tr>
<tr>
<td>M+ ǫG/R</td>
<td>.0112</td>
<td>.0237</td>
<td>.0213</td>
<td>.0380</td>
<td>.0393</td>
<td>.0399</td>
<td>.0419</td>
<td>TQ-ǫG/R</td>
</tr>
<tr>
<td>M+ ǫG/D</td>
<td>.0100</td>
<td>.0152</td>
<td>.0166</td>
<td>.0131</td>
<td>.0228</td>
<td>.0228</td>
<td>.0266</td>
<td>M+ ǫG/R</td>
</tr>
<tr>
<td>M+UCBE/R</td>
<td>.0110</td>
<td>.0134</td>
<td>.0328</td>
<td>.0400</td>
<td>.0485</td>
<td>.0452</td>
<td>.0452</td>
<td>M+UCBE/R</td>
</tr>
<tr>
<td>M+UCBE/D</td>
<td>.0283</td>
<td>.0318</td>
<td>.0195</td>
<td>.0226</td>
<td>.0313</td>
<td>.0452</td>
<td>.0452</td>
<td>M+UCBE/R</td>
</tr>
<tr>
<td>M+J</td>
<td>.0088</td>
<td>.0149</td>
<td>.0302</td>
<td>.0182</td>
<td>.0210</td>
<td>.0214</td>
<td>.0219</td>
<td>M+ ǫG/R</td>
</tr>
</tbody>
</table>

the component’s task performance is lower. Also, we see from the figures that all the curves have much wider surrounding strips, suggesting the performance has higher variance and the exploration processes terminate before they converge, which can be attributed to either the input variance or the component’s task performance variance. Nevertheless, we can still see from Table 7.13 that most strategies keep achieving higher results as the experiment proceeds.

From the progress curves in Figures 7.4, 7.5, and 7.6, we see that most of the strategies using a TQ or FA value estimator can collect 300–1000 episodes, but those using a M+ model-based method can only collect 200–250 episodes. In fact, we find that the error condition is more complex for this task than the simulated environments. There are two common error types: ParseException and IllegalArgumentException. As we discussed in Section 5.4.4, a ParseException does not necessarily indicate the component that throws it is not functioning, instead it often occurs when certain combination(s) of analysis components in upstream pipeline that are incompatible with each other are used. The IllegalArgumentException is trickier. Even the same combination of analysis components is used, a IllegalArgumentException may occur for some inputs (more complex or concept-rich) but not for the others. As a result, a model-based method tends to predict that almost all combinations of components can produce an error. Therefore, training an error predictor is challenging for this task, especially when using an aggressive exploration method. When we
compare the progress curves of the strategies using a TQ or FA value estimator and an independent component selection method, we find that those using the density ordered heuristic collect more episodes than those using the reward ordered heuristic. Those using the Successive Rejects algorithm (J) collect the fewest episodes in the first 40 minutes, then accelerate in the last 20 minutes, and finally collect more episodes than many other strategies. In fact, the Successive Rejects algorithm explores randomly (more aggressively than $\epsilon$-greedy) and does not eliminate any option until after 30+ minutes. Therefore, while most strategies start avoiding the low-quality and high-cost components soon after they collect enough evidence, those using the Successive Rejects algorithm delay the action of eliminating these components.

Next, we compare the best policies returned from the strategies at different checkpoints. We
find the policy optimization strategies that use the reward ordered heuristic tend to outperform their counterparts that use the density ordered heuristic. We hypothesize that it is because the components in different phases of this configuration space are dependent, i.e. the component that maximizes the overall task performance depends on the components selected in other phases, which is a different scenario from the simulated experiment. For example, whether or not we should use the concept names in the abstract query depends on which concept identification components we have used upstream, and which document retrieval component we should use in turn depends on the complexity and syntax of the query string created by an earlier abstract query constructor. Therefore, the components that have the highest density in each greedy phase and are thus selected during the exploration process do not necessarily have the highest reward in that phase, which sets up a suboptimal “baseline” system and misleads the selection of components in the remaining exploratory phases.

Among the strategies that use the reward ordered heuristic, we find the TQ-\(\epsilon\)G/R strategy performs consistently the best in the first 40 minutes, but starts to drop afterward, which might be attributed to a large learning rate, as we have noticed from Figure 5.6 in Section 5.5, where we claim that this phenomenon is an indicator of non-convergence, due to a large learning rate (\(\alpha\)) and/or large exploration rate (\(\epsilon\)), the two parameters of the exploration strategy. Real world problems, such as the biomedical QA task, tend to have much higher variance than the simulated tasks, i.e. we may get far different results even if we apply the same combination of components. Usually, we recommend to use small learning rate for a problem with large variance but then expect a low convergence rate to secure convergence. We can also consider the same suggestions as for the inferior optimal performance shown in Figure 5.6, including an decayed or adaptive learning rate and/or exploration rate.

When we compare the TQ-\(\epsilon\)G/R strategy with the FA-\(\epsilon\)G/R strategy, both using the same learning rate (0.1), we see the performance of the latter grows more slowly but steadily, which might suggest the necessity of using a universal baseline estimator (i.e. the intercept in the linear regres-
Table 7.14: Results of budgeted learning strategies with random phase selection for BioASQ Phase A (relevant snippet retrieval task)

<table>
<thead>
<tr>
<th>Strategy</th>
<th>1m</th>
<th>10m</th>
<th>20m</th>
<th>30m</th>
<th>40m</th>
<th>50m</th>
<th>60m</th>
<th>Comparand</th>
</tr>
</thead>
<tbody>
<tr>
<td>TQ-RR/R</td>
<td>.108 ns</td>
<td>.098 ns</td>
<td>.0289*</td>
<td>.0316*</td>
<td>.0315*</td>
<td>.0465 ns</td>
<td>.0451 ns</td>
<td>TQ-εG/R</td>
</tr>
<tr>
<td>TQ-RR/D</td>
<td>.0273 ns</td>
<td><strong>.0472</strong></td>
<td>.0468 ns</td>
<td>.0395 ns</td>
<td>.0552 ns</td>
<td>.0507 ns</td>
<td>.0636 ns</td>
<td>TQ-εG/D</td>
</tr>
<tr>
<td>TQ-RU1/R</td>
<td>.0075 ns</td>
<td>.0314 ns</td>
<td>.0331 ns</td>
<td>.0367 ns</td>
<td>.0549†</td>
<td>.0513 ns</td>
<td><strong>.0710</strong></td>
<td>TQ-UCBE/R</td>
</tr>
<tr>
<td>TQ-RU1/D</td>
<td>.0090 ns</td>
<td>.0208 ns</td>
<td>.0244 ns</td>
<td>.0309 ns</td>
<td>.0303 ns</td>
<td>.0312 ns</td>
<td>.0285 ns</td>
<td>TQ-UCBE/D</td>
</tr>
<tr>
<td>TQ-RUE/R</td>
<td><strong>.0286</strong></td>
<td>.0382 ns</td>
<td>.0532†</td>
<td><strong>.0639</strong></td>
<td><strong>.0648</strong></td>
<td><strong>.0748</strong></td>
<td>.0637 ns</td>
<td>TQ-UCBE/R</td>
</tr>
<tr>
<td>TQ-RUE/D</td>
<td>.0110 ns</td>
<td>.0241 ns</td>
<td>.0400 ns</td>
<td>.0517 ns</td>
<td>.0478 ns</td>
<td>.0403 ns</td>
<td>.0417 ns</td>
<td>TQ-UCBE/D</td>
</tr>
<tr>
<td>TQ-RJ</td>
<td>.0096 ns</td>
<td>.0264 ns</td>
<td>.0261 ns</td>
<td>.0280 ns</td>
<td>.0623†</td>
<td>.0607 ns</td>
<td>.0633 ns</td>
<td>TQ-J</td>
</tr>
<tr>
<td>FA-RU1/R</td>
<td>.0151†</td>
<td>.0342 ns</td>
<td>.0530 ns</td>
<td>.0741*</td>
<td>.0765**</td>
<td>.0773†</td>
<td>.0792 ns</td>
<td>FA-εG/R</td>
</tr>
<tr>
<td>FA-RU1/D</td>
<td>.0151 ns</td>
<td>.0238 ns</td>
<td>.0392 ns</td>
<td>.0401 ns</td>
<td>.0435 ns</td>
<td>.0451 ns</td>
<td>.0532 ns</td>
<td>FA-εG/D</td>
</tr>
<tr>
<td>FA-RU1/R</td>
<td>.0195 ns</td>
<td>.0367 ns</td>
<td>.0391 ns</td>
<td>.0489 ns</td>
<td>.0372 ns</td>
<td>.0538 ns</td>
<td>.0725 ns</td>
<td>FA-UCBE/R</td>
</tr>
<tr>
<td>FA-RU1/D</td>
<td>.0079 ns</td>
<td>.0243 ns</td>
<td>.0440 ns</td>
<td>.0409 ns</td>
<td>.0468 ns</td>
<td>.0499†</td>
<td>.0473 ns</td>
<td>.0481 ns</td>
</tr>
<tr>
<td>FA-RUE/R</td>
<td><strong>.0489</strong></td>
<td>.0250 ns</td>
<td>.0267 ns</td>
<td>.0668 ns</td>
<td>.0335 ns</td>
<td>.0447 ns</td>
<td>.0352 ns</td>
<td>FA-UCBE/R</td>
</tr>
<tr>
<td>FA-RUE/D</td>
<td>.0186 ns</td>
<td>.0228 ns</td>
<td>.0129 ns</td>
<td>.0166 ns</td>
<td>.0227 ns</td>
<td>.0281 ns</td>
<td>.0313 ns</td>
<td>FA-UCBE/D</td>
</tr>
<tr>
<td>FA-RJ</td>
<td>.0061 ns</td>
<td>.0087 ns</td>
<td>.0136 ns</td>
<td>.0271 ns</td>
<td>.0531 ns</td>
<td>.0510 ns</td>
<td>.0594 ns</td>
<td>FA-J</td>
</tr>
<tr>
<td>M+-RR/R</td>
<td>.0089 ns</td>
<td>.0213 ns</td>
<td>.0187 ns</td>
<td>.0393 ns</td>
<td>.0400 ns</td>
<td>.0284 ns</td>
<td>.0334 ns</td>
<td>M+-εG/R</td>
</tr>
<tr>
<td>M+-RR/D</td>
<td>.0022 ns</td>
<td>.0445 ns</td>
<td><strong>.0519†</strong></td>
<td>.0516†</td>
<td>.0296 ns</td>
<td>.0240 ns</td>
<td>.0430 ns</td>
<td>M+-εG/D</td>
</tr>
<tr>
<td>M+-RU1/R</td>
<td><strong>.0211</strong></td>
<td>.0496†</td>
<td>.0417 ns</td>
<td>.0545 ns</td>
<td><strong>.0724</strong> ns</td>
<td>.0394 ns</td>
<td>.0411 ns</td>
<td>M+-UCBE/R</td>
</tr>
<tr>
<td>M+-RU1/D</td>
<td>.0025 ns</td>
<td><strong>.0565</strong></td>
<td>.0477 ns</td>
<td>.0612 ns</td>
<td>.0666 ns</td>
<td><strong>.0461</strong> ns</td>
<td><strong>.0585</strong> ns</td>
<td>M+-UCBE/D</td>
</tr>
<tr>
<td>M+-RUE/R</td>
<td>.0119 ns</td>
<td>.0255 ns</td>
<td>.0330 ns</td>
<td>.0321 ns</td>
<td>.0376 ns</td>
<td>.0355 ns</td>
<td>.0382 ns</td>
<td>M+-UCBE/R</td>
</tr>
<tr>
<td>M+-RUE/D</td>
<td>.0094 ns</td>
<td>.0500 ns</td>
<td>.0308 ns</td>
<td>.0353 ns</td>
<td>.0308 ns</td>
<td>.0141 ns</td>
<td>.0224 ns</td>
<td>M+-UCBE/D</td>
</tr>
<tr>
<td>M+-RJ</td>
<td>.0071 ns</td>
<td>.0205 ns</td>
<td>.0312 ns</td>
<td>.0305 ns</td>
<td>.0466†</td>
<td>.0444 ns</td>
<td>.0349 ns</td>
<td>M+-J</td>
</tr>
</tbody>
</table>

In general, those that use UCBE/R for option selection exhibit the most consistently good performance among others, and the model-based policy optimization strategies (M+) do not perform well.

Next, we report the comparison results of the strategies that use a two-step option selection method in Table 7.14 and Figures 7.7, 7.8, and 7.9. When we compare the progress curves, we see the difference between those that use the density ordered heuristic and those that use the reward ordered heuristic is less noticeable, due to more conservative exploration schedules. However, we see that those use the Successive Rejects algorithm (J) can collect much more episodes than others in the final one third of the exploration process. We suspect that the configuration space contains a few components that cost much more than the others. Even if they are not the greedy options, they are still selected with a small probability in exploratory phases, and once they are selected, the efficiency is still hugely affected. The only strategy to improve the efficiency in this case is to eliminate such components completely from the configuration space.

If the budget only allows a 20-minute exploration, the best budgeted policy optimization strategies include the TQ-RR/D, M+-RU1/D, and M+-RUE/D strategies, all using the density ordered
heuristic. When the budget is relatively sufficient (> 20 minutes), we find that those using the reward ordered heuristic outperform their counterparts using the density ordered heuristic. Among all the strategies, the FA-RR/R strategy performs most consistently the best after 20 minutes, which also surprisingly has a relatively small variance. First, we can learn from the fact that the FA-RR/R strategy outperforms the TQ-RR/R strategy, using a universal baseline estimator (i.e. the intercept in the linear regression) for value estimation would be necessary. Second, since it also outperforms FA-RR/D and the other strategies that use the density ordered heuristic, it suggests that we should use a behavior policy that is close to (not too off) the estimation policy, and then apply the off-policy method for value estimation. Third, since it outperforms the FA-RU1/R and FA-RUE/R strategies, we believe a more conservative but steady exploration strategy, rather than a decayed
Figure 7.9: Learning curves of M+ strategies w/ 2-step option selection for Phase A

Figure 7.10: Estimated Q value and cost of components in Phases 10, 12, 15 of Phase A. The blue circles indicate integrated components and the red star indicates the baseline component.

...exploration strategy (e.g. UCB), would help accurately estimate the values of components in the configuration space with intra-phase component dependencies. We also find that when using the two-step option selection method for the model-based strategies (M+), the performance can be greatly improved, especially given a tight budget (10 minutes), which is because the relatively conservative two-step option selection method can collect more training instances but the dimensionality of feature space grows sublinearly, making it easier and accurately to predict the benefit and error likelihood.

Learned policy

We observe the best policy returned by the TQ-RUE/R strategy (a randomly selected run on the BioASQ 3B Batch 5 subset), due to its high performance and simplicity of model interpretation, and compare with the default baseline system used for the official evaluation and open source re-
lease. We plot the estimated Q value and the estimated cost (both using the tabular value estimation technique) for the components in Phases 10 (primary abstract query construction), 12 (document retrieval), and 15 (passage filter) of Phase A in Figure 7.10. We see that the components that the component developers and the integration testers believe the best and thus are specified as the baseline components tend to have low cost as expected, since they have undergone a series of performance optimization procedures during the refactoring process. However, they are not necessarily the most effective ones.

In Figure 7.10(a), the baseline component is ranked at 7/10 in terms of its estimated Q value, which uses the option

\[
\text{TokenConceptAbstractQueryGenerator}[\text{inherit:
example.bioasq.abstract_query.token-concept-pos-stoplist#pos-
tags-path:/dictionaries/pos-tags.txt#noun-tags-path:/dictionaries/noun-
tags.txt#stoplist-path:/dictionaries/stoplist.txt#use-weight:
false}],
\]

whereas the option with the highest estimated Q value uses

\[
\text{ConceptAbstractQueryGenerator}[\text{inherit:
example.bioasq.abstract_query.concept#use-weight:
true#required:
true}],
\]

which ignores all the tokens and only considers the concept names.

In Figure 7.10(b), the baseline component is indeed the component that is estimated the highest Q value, which is defined as

\[
\text{LuceneDocumentRetrievalExecutor}[\text{inherit:
example.bioasq.document.retrieval.lucene-medline#index:
../index/medline16n-lucene#id-field:pmid#title-field:
aricleTitle#text-field:abstractText#uri-prefix:
http://www.ncbi.nlm.nih.gov/pubmed/#query-string-construk-
tor:edu.cmu.lti.oaqa.baseqa.providers.query.LuceneQueryStringC-
StandardAnalyzer#hits:100#fields:articleTitle#abstractText}
].
\]

Besides that this component has much better runtime and task performance than the two components that require remote Web services (the two blue circles on the left-top corner of the figure), it also outperforms its sibling options, which include the Lucene document retrieval modules that use the bag-of-phrase or boolean-bag-of-phrase query constructor, and/or that return the top 50 or 150 (hits) documents.

In Figure 7.10(c), the baseline component, a simple noop option (pass without doing anything), costs the least, but it is not the most effective one again. The option that gives the highest Q value is

207
The best option uses an unsupervised method to filter the sentences, and returns only the top 100 sentences for the downstream supervised method to predict relevance, whereas the \texttt{noop} component passes all candidate sentences onto the passage reranking component for classification. The reason that we originally decided to deprecate the passage filtering phase is that we intend to totally count on the supervised passage reranking component to decide whether each candidate sentence is relevant, which requires the recall of the input sentences should be as high as possible. The best way to maintain a high level of recall is not to filter any passage at all. However, the configuration space exploration result suggests that our assumption is wrong. The reality is that, as we increases the \texttt{hits} from 100 (optimal) to 200 or 300 (suboptimal), the false positive number goes up more quickly than the true negative number, meaning that the classifier can more correctly distinguish the irrelevant passages from the relevant ones if the irrelevant passages are similar with the relevant ones (within the top 100 sentences in terms of the Lucene score).

### 7.6.2 Phase B: Factoid Question Answering

In this subsection, we first describe the configuration space for the factoid question answering task and report the configuration space exploration results.

**Configuration Space**

First, all the NLP parser options and concept identification component options that are integrated into the configuration space for the relevant snippet retrieval task are again specified for both the question and evidence (relevant snippets) parsing and concept identification phases in the configuration space for the factoid QA task. In this experiment, we mainly vary the answer type prediction phase (Phase 19), the answer generation phase (Phase 20), the answer scoring and ranking phase (Phase 22), and the answer pruning phase (Phase 24). For answer type prediction phase, we specify two pretrained models, one allowing the “null” type and the other not allowing, and for each model, we also decide how many top results we use for the downstream pipeline. For answer generation phase, we use different combinations (power set) of answer generators. For answer ranking, we also integrate the Weka CVR classifier and the deprecated simple scorer that is developed for BioASQ 3B, in addition to the logistic regression wrapper. For answer pruning phase, we include the absolute thresholding method alone, the ratio thresholding method alone, and also both methods together, each with different threshold values. We list the components and configurations that are used in this experiment in Table 7.15. We see that, the resulting configuration space contains 5,945,425,920 combinations, much more than that for the relevant snippet retrieval task, although we specify fewer options for each phase, due to a longer pipeline.
<table>
<thead>
<tr>
<th>Phase #</th>
<th>Phase name / options</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>question-parse</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>model: [clearnlp-bioinformatics, clearnlp-medical]</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>question-concept-metamap</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>question-concept-tmtool</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>question-concept-lingpipe-genia</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>question-concept-lingpipe-genetag</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>question-concept-opennlp-np</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>question-concept-opennlp-nppnnp</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>question-focus</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>passage-to-view</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>evidence-parse</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>model: [clearnlp-bioinformatics, clearnlp-medical]</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>evidence-concept-metamap</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>evidence-concept-tmtool</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>evidence-concept-lingpipe-genia</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td>evidence-concept-frequent-phrase</td>
<td>2</td>
</tr>
<tr>
<td>15</td>
<td>evidence-concept-opennlp-np</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>evidence-concept-opennlp-nppnnp</td>
<td>2</td>
</tr>
<tr>
<td>17</td>
<td>concept-search-uts</td>
<td>2</td>
</tr>
<tr>
<td>18</td>
<td>concept-merge</td>
<td>2</td>
</tr>
<tr>
<td>19</td>
<td>answer-type</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>liblinear-null-predict: {limit: [1, 3, 5, 7, 9]} + liblinear-predict: {limit: [1, 3, 5, 7, 9]}</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>answer-generate</td>
<td>12</td>
</tr>
<tr>
<td>21</td>
<td>answer-modify</td>
<td>2</td>
</tr>
<tr>
<td>22</td>
<td>answer-rank</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>liblinear-predict + weka-cvr-predict + simple-scorer: {smoothing: [2.0, 1.0, 5.0, 10.0, 20.0]}</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>answer-collective-index</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>liblinear-predict + weka-cvr-predict + noop</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>answer-prune</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>noop + absolute: {threshold: [.5, .7]} + ratio: {threshold: [.72, .88]} + absolute: {threshold: [.5, .7]} \times ratio: {threshold: [.72, .88]}</td>
<td></td>
</tr>
</tbody>
</table>
We use MRR (reciprocal rank), a classic metric for the factoid QA task, to measure the task performance. Again, we define the runtime performance as the time duration that the component spends during each execution. Considering the average execution time of the integrated components, we run each of the 40 budgeted policy optimization strategies that are used in Section 5.7, for up to 10 minutes, and each is repeated 5 times. We execute the five random runs using the same strategy in parallel on a server equipped with the Intel Xeon E5-1410 8-core processor and a 16-GB memory. In fact, compared with the configuration space for the relevant snippet retrieval task, there is no component like the ones in the document retrieval phase that are I/O-heavy or depend on remote Web servers. We report the mean (and standard error in plots) and the significant level over the comparand from t-test.

### Comparison of learning performance of (budgeted) learning strategies

We first compare the strategies that use an independent option selection method. The results are reported in Table 7.16 and Figures 7.11, 7.12, and 7.13. Although we define much less budget to explore the configuration space for Phase B, we can still collect more episodes than Phase A, which means the average cost of executing an episode or component is less. We see that the strate-
gies using the density ordered heuristic clearly collect more episodes than those using the reward ordered heuristic, more noticeable from those using a TQ or FA value estimator. The Successive Rejects algorithm does not help collect more episodes, indicating that the runtime performance and the task performance of the components is either uncorrelated or positively correlated, which is different from the configuration space for Phase A (relevant snippet retrieval task). We see that the numbers reported in Table 7.16 are higher than those reported in Table 7.13.

When we compare the best policies returned from the strategies at different checkpoints, we find that many strategies exhibit an upward trend as more episodes are collected. The official evaluation has reported our factoid QA system that participated and won the BioASQ 4B challenge achieved the performance of 0.2308 in terms of MRR (in Table 7.8). When we compare the results
between our baseline system and the best systems returned by the learning strategies after they use up the 10-minute budget, we find that the TQ-UCBE/R, FA-εG/D, M+-εG/R strategies out of a total of 19 strategies identify a better system. Among the strategies, the model-based strategies (M+) tend to have the best and the most consistent performance, especially when used with the reward ordered heuristic. The components that are integrated into the configuration space for Phase B less likely throw exceptions than those for Phase A, and therefore it is easier for a model-based strategy to learn the benefit of each component.

When we switch the option selection method to a more conservative two-step option, the results are reported in Table 7.17 and Figures 7.14, 7.15, 7.16. We find that the performance of the most budgeted policy optimization strategies, especially those using a TQ or FA value estimator and...
Table 7.17: Results of budgeted learning strategies with random phase selection for BioASQ Phase B (factoid QA task)

<table>
<thead>
<tr>
<th>Strategy</th>
<th>30s</th>
<th>2m</th>
<th>4m</th>
<th>6m</th>
<th>8m</th>
<th>10m</th>
<th>Comparand</th>
</tr>
</thead>
<tbody>
<tr>
<td>TQ-RR/R</td>
<td>.0835ns</td>
<td>.1247ns</td>
<td>.1778ns</td>
<td>.1685ns</td>
<td>.2101†</td>
<td>.2044ns</td>
<td>TQ-cG/R</td>
</tr>
<tr>
<td>TQ-RR/D</td>
<td>.1382ns</td>
<td>.1563ns</td>
<td>.1913ns</td>
<td>.2283**</td>
<td>.2173ns</td>
<td>.2285ns</td>
<td>TQ-cG/D</td>
</tr>
<tr>
<td>TQ-RU1/R</td>
<td>.0830†</td>
<td>.1963ns</td>
<td>.2263ns</td>
<td>.2045ns</td>
<td>.2657ns</td>
<td>.2498ns</td>
<td>TQ-UCBE/R</td>
</tr>
<tr>
<td>TQ-RU1/D</td>
<td>.1015ns</td>
<td>.1430ns</td>
<td>.2105ns</td>
<td>.2083†</td>
<td>.2520†</td>
<td>.2299*</td>
<td>TQ-UCBE/D</td>
</tr>
<tr>
<td>TQ-RUE/R</td>
<td>.1206ns</td>
<td>.2516ns</td>
<td>.2212ns</td>
<td>.2172ns</td>
<td>.2419ns</td>
<td>.2224ns</td>
<td>TQ-UCBE/R</td>
</tr>
<tr>
<td>TQ-RUE/D</td>
<td>.1382ns</td>
<td>.1418ns</td>
<td>.1613ns</td>
<td>.1804†</td>
<td>.2293ns</td>
<td>.2389*</td>
<td>TQ-UCBE/D</td>
</tr>
<tr>
<td>TQ-RJ</td>
<td>.0727†</td>
<td>.1436ns</td>
<td>.1913ns</td>
<td>.1772ns</td>
<td>.2027†</td>
<td>.1944ns</td>
<td>TQ-J</td>
</tr>
<tr>
<td>FA-RR/R</td>
<td>.0400ns</td>
<td>.0799ns</td>
<td>.1062ns</td>
<td>.1060ns</td>
<td>.1325ns</td>
<td>.1371*</td>
<td>FA-cG/R</td>
</tr>
<tr>
<td>FA-RR/D</td>
<td>.0735ns</td>
<td>.1784ns</td>
<td>.1906ns</td>
<td>.1898ns</td>
<td>.1907ns</td>
<td>.1816ns</td>
<td>FA-cG/D</td>
</tr>
<tr>
<td>FA-RU1/R</td>
<td>.1148ns</td>
<td>.1739ns</td>
<td>.2356ns</td>
<td>.2443†</td>
<td>.2325ns</td>
<td>.2396†</td>
<td>FA-UCBE/R</td>
</tr>
<tr>
<td>FA-RU1/D</td>
<td>.0147ns</td>
<td>.1724ns</td>
<td>.2150*</td>
<td>.2388**</td>
<td>.2414**</td>
<td>.2380**</td>
<td>FA-UCBE/D</td>
</tr>
<tr>
<td>FA-RUE/R</td>
<td>.1144ns</td>
<td>.1897ns</td>
<td>.2387†</td>
<td>.2293ns</td>
<td>.2422†</td>
<td>.2485†</td>
<td>FA-UCBE/D</td>
</tr>
<tr>
<td>FA-RUE/D</td>
<td>.0723ns</td>
<td>.2011†</td>
<td>.2367*</td>
<td>.2498*</td>
<td>.2393*</td>
<td>.2744**</td>
<td>FA-UCBE/D</td>
</tr>
<tr>
<td>FA-RJ</td>
<td>.0241ns</td>
<td>.1247ns</td>
<td>.1325ns</td>
<td>.1642ns</td>
<td>.2027†</td>
<td>.1944ns</td>
<td>FA-J</td>
</tr>
<tr>
<td>M+-RR/R</td>
<td>.0299ns</td>
<td>.2436ns</td>
<td>.1957†</td>
<td>.2155ns</td>
<td>.2473ns</td>
<td>.2699ns</td>
<td>M+-cG/R</td>
</tr>
<tr>
<td>M+-RR/D</td>
<td>.0160ns</td>
<td>.1072ns</td>
<td>.1413ns</td>
<td>.1630ns</td>
<td>.1352ns</td>
<td>.2177ns</td>
<td>M+-cG/D</td>
</tr>
<tr>
<td>M+-RU1/R</td>
<td>.0000ns</td>
<td>.1613ns</td>
<td>.2343ns</td>
<td>.1928ns</td>
<td>.2218ns</td>
<td>.2070ns</td>
<td>M+-UCBE/R</td>
</tr>
<tr>
<td>M+-RU1/D</td>
<td>.0160ns</td>
<td>.1543ns</td>
<td>.1823ns</td>
<td>.1974ns</td>
<td>.2123ns</td>
<td>.2126ns</td>
<td>M+-UCBE/D</td>
</tr>
<tr>
<td>M+-RUE/R</td>
<td>.0000ns</td>
<td>.1841ns</td>
<td>.1873ns</td>
<td>.2257ns</td>
<td>.2336ns</td>
<td>.2353ns</td>
<td>M+-UCBE/R</td>
</tr>
<tr>
<td>M+-RUE/D</td>
<td>.0116ns</td>
<td>.2113ns</td>
<td>.2066ns</td>
<td>.2172ns</td>
<td>.2401ns</td>
<td>.2613ns</td>
<td>M+-UCBE/D</td>
</tr>
<tr>
<td>M+-RJ</td>
<td>.0144ns</td>
<td>.2044ns</td>
<td>.2265*</td>
<td>.2276ns</td>
<td>.2439ns</td>
<td>.2307ns</td>
<td>M+-J</td>
</tr>
</tbody>
</table>

the density ordered heuristic, gets improved. For example, the two-step option selection method significantly (with a confidence level of at least 95%) improves five out of seven strategies that use a FA value estimator in terms of the MRR evaluation metric for the policy returned at 10 minutes, which also makes it the most effective strategy group for optimizing the Phase B (factoid QA) task. Using the two-step option selection method, nine (out of 21) strategies are able to discover a system that performs better than the baseline system, including the TQ-RU1/R, TQ-RUE/R, FA-RU1/R, FA-RU1/D, FA-RUE/R, FA-RUE/D, M+-RR/R, M+-RUE/R, M+-RUE/D strategies. All other strategies are also able to find a system whose performance is measured at least 0.17, compared to the strategies using the aggressive independent option selection method, more than half (10/19) of which fail to identify a system with the performance above 0.17 given the same budget.

We further compare between the strategies that use a two-step option selection method, and we observe different phenomena from the experimental results for Phase A. First, we find that those using the density ordered heuristic perform slightly better than or comparable to their counterparts using the reward ordered heuristic. Also, the best policy is often discovered by the strategies that
use either the random/UCB1 (RU1) method or the random/UCB-E method for option selection, rather than the random/random (RR) method as in Phase A.

**Learned policy**

Again, we use the best policy returned by the TQ-RUE/R strategy, and compare with the default baseline system. We plot the estimated Q value and the estimated cost (both using the tabular value estimation technique) for the components in Phases 19 (answer type prediction), 20 (answer generation), and 24 (answer pruning) of Phase B in Figure 7.17. We see that the figure suggests there exist other components that may have better performance than those currently integrated in
Figure 7.17: Estimated Q value and cost of components in Phases 19, 20, 24 of Phase B. The blue circles indicate integrated components and the red star indicates the baseline component.

The baseline system.

Figure 7.17(a) shows the learned cost and Q value for the components in the answer type prediction phase. Among them, the baseline component is estimated to have the lowest Q value, which uses

\[
\text{AnswerTypeClassifierPredictor[inherit:example.bioasq.answer_type.liblinear-null-predict#classifier:inherit:example.bioasq.answer_type.liblinear-null#feature-constructor:inherit:baseqa.answer_type.feature-constructor#limit:5]},
\]

and the component with the highest Q value uses

\[
\text{AnswerTypeClassifierPredictor[inherit:example.bioasq.answer_type.liblinear-null-predict#classifier:inherit:example.bioasq.answer_type.liblinear-null#feature-constructor:inherit:baseqa.answer_type.feature-constructor#limit:1]},
\]

which uses the same configuration except that it only returns the top-1 result, rather than top-5. In fact, we decided to use the top 5 predicted answer types in the subsequent processing steps when we first prepared for BioASQ 3B evaluation, because we found the top-1 result is not reliable back then. However, as we have detailed in Section 7.4.4, we have made many changes since built them, e.g., we included the additional concept types after we also used TmTool to annotate the gold standard answer types, we introduced the “null” type if neither TmTool or MetaMap could identify, we relabeled the gold-standard answer types for the gold-standard answer texts, using both services after we separated the texts inside parentheses, also we retrained the model using a development set almost doubled in size, etc. This observation reminds us that it’s time to reevaluate if the original assumption still holds.

Figure 7.17(b) also suggests that there may exist components that is more efficient and has higher task performance in the answer generation phase. The baseline component is
and the top two components for this phase are

\[
\]

We find that both of the top components do not use the cav-covering-concept generator. In other words, this generator introduces more noise than useful answer texts. Among the top components, the exploration strategy prefers not to use the choice generator, because the data set that we use for this exploration experiment does not contain any choice question.

Figure 7.17(c) compares the performance of answer pruning components. The baseline system uses the noop component, but the strategy selects the component

\[
\text{AnswerModificationManager[inherit:example.bioasq.answer.modify.pruner#handlers:-inherit:baseqa.answer.modify.modifiers.list-50-inherit:baseqa.answer.modify.modifiers.ratio-72]}. 
\]

In fact, since we are only allowed to return the top 10 answers, a component that has low thresholding behaves as if it is not executed. The difference between these components is the estimation error (due to the randomness).

### 7.6.3 Summary

Finally, we summarize the consideration factors when we need to choose the exploration strategy, based on the two exploration experiments for real world problems reported in this section.

- If you are on a very tight budget, then you may consider an aggressive exploration strategy, i.e. those using independent option selection and the density ordered heuristic, e.g. UCBE/D or \( \epsilon \)-G/D. However, the more aggressive the exploration strategy you use is, the more unpredictable the exploration outcome might be.
- When you are on a not-so-tight budget, you may consider to use a two-step option selection method combined with any value estimator (value-based method or model-based method).
• If you know there exists *intra-phase component dependency*, it would also be much safer to use the *reward ordered heuristic* with a *conservative but steady exploration strategy* such as RR to avoid decaying the exploration rate as the exploration proceeds.

• If you believe the configuration space only contains *independent components*, i.e. there is no or little intra-phase component dependency, then you may use a *density ordered heuristic* and a more *aggressive exploration strategy* such as RUE, which still ensures the behavior policy is not too-off the estimation policy.

• If there exist component(s) that are not the highest quality but cost much more than the others, you should consider an *elimination algorithm*, such as RJ.

### 7.7 Conclusion

In this chapter, we demonstrate how to leverage the analytics meta learning methodology to accomplish a real world analysis task – biomedical question answering task. We describe the analysis component construction details in Section [7.4] and report the configuration space exploration results in Section [7.6]. We focus on comparing the exploration strategies proposed in Chapter [5] and summarize the strategy selection principles based on our observation. We also compare the information systems discovered by the exploration strategies with the baseline system which participated in the official evaluation (whose results are also reported in Section [7.5]), and discover that almost half of the strategies use a two-step option selection method have identified a better system.

In the future, we plan to use a larger development set and test set to reduce the learning and testing variance. We also consider to further create analysis components for the achieved functionalities to improve their performance and develop components for the missing functionalities, such as summary generation, to complement the current system.
Chapter 8

Case Study: Pharmaceutical Decision Task

Developing a new drug is a complex and costly process that proceeds from the identification of a potential therapeutic candidate to marketing a drug product, usually taking more than a decade and costing in excess of 1 billion US dollars [87, 160]. This includes identifying and properly validating drug targets, defining the utility of employing probes in the early discovery phase, medicinal chemistry, lead optimization, preclinical proof of concept strategies, and drug delivery needs through preclinical proof of concept [88]. There exist various approaches in drug development: New Chemical Entity (NCE) discovery, drug repurposing (beneficial activity currently marketed drugs possess against novel targets), drug delivery improvement, etc. [88].

Along each development path, decision makers need to consider a series of predetermined criteria at each phase, in order to prioritize drug candidates to reduce risk to human subjects and to increase the chance of picking a winning therapeutic molecule in NCE [160]. For example, in the earliest stage of NCE, target identification and target validation (or target assessment) aim to select and prioritize a number of disease targets (agents with a particular biological action that are anticipated to have therapeutic utility), and estimate the “druggability” of each target influenced by a complex balance of scientific, medical and strategic considerations, including efficacy, safety, commercial profits, etc. [87, 102]. A huge amount of structured or unstructured data have become available and can facilitate the stakeholders to make better prediction and decision. For example, one can review the literature and public/proprietary data to collect a list of molecule modulating targets, know the history of success of each type of target, and obtain genetic confirmation. He/she can also investigate the intellectual property and marketability of the target from patent database and market analysis portals [88].

Discovering and evaluating all the relevant, but possibly inconsistent or even contradictory, conclusions from over 20 million biomedical research publications as well as various manually curated databases can be very challenging. We attempt to leverage analytics meta learning to help with such complex decision support tasks. In this case study, we focus on the scientific factors rather than the commercial aspects of the problem. In Section 8.1, we formally define the use case of the problem by showing inputs and expected outputs. In Section 8.2, we first present the manually created analytics procedure for target validation, which includes a decision process and a simple generic execution process for each decision factor. In Section 8.3, we describe a machine learning solution synthesis algorithm and the analysis components that we adapt from
the biomedical question answering system developed in Chapter 7. In Section 8.4 we apply a full analytics meta learning approach to optimize target validation decision support system, where we compare the performance between the 28 value-based budgeted exploration strategies. The optimal system can achieve an accuracy of above 0.75. In this experiment, the system is able to update the meta learning model simultaneously while it trains the lower-level individual machine learning components such as the solution synthesis module. We conclude in Section 8.5.

Many decision support systems have been developed in support of various decision scenarios during the past few decades [33, 121, 218]. Although most decision support systems are built for predetermined tasks, which requires domain knowledge obtained from humans and stored in a structured data store with a predefined schema, a few decision support systems have explored how to understand decision needs from natural language input. For example, the Structured Evidential Argumentation System (SEAS) [121] provides decision references for national security crisis warnings, and Intelligent Decision System (IDS) [218] analyzes business innovation self-assessments written in free-text. However, neither approach is general or extensible enough to cover the cases we implemented with our proposed solution and open-source implementation. IBM’s Chef Watson [23] is another interesting example, which is able to suggest original cooking recipes after read 9,000 existing manually curated recipes from Bon Appetit, a special type of procedural description on a special procedural knowledge base.

8.1 Task Description

Similar to biomedical information seeking needs described in Section 7.2, pharmaceutical decision tasks have the same input type, e.g. a factoid/list question

Which genes are directly involved in the breast cancer and can be a suitable target?

or a yes/no question

Is FGFR1 directly involved in the breast cancer and can be a suitable target?

and output requirement

AKT1, BRCA1, BRCA2, BRIP1, ERBB2, FGFR1, KDR1, mTOR, NBN, FGFR1, KDR1, mTOR, etc.

or

yes

Since the users are often interested in understanding how the decision is made, a detailed decision report is also needed, which includes the decision logic and the collected evidence, either formatted or in natural language, e.g.
• If there exists any in vivo experiment showing that modulating the activity of the gene affects biochemical function or phenotype of breast cancer, then it is a positive sign that the gene can be a target.

• We have found the literature that there exists in vivo experiment showing that modulating the activity of FGFR affects biochemical function or phenotype of breast cancer, including

  - PMID: 25400686
  - PMID: 15863030
  - ...

8.1.1 Evaluation Metrics

Although the outcome in the decision support tasks is sometimes subjective and difficult to measure until later phases are conducted in the drug development process, this case study, as the first attempt to a series of pharmaceutical decision scenarios, demonstrates how to make prediction purely and automatically from the literature by considering various decision factors involved in the decision. In the case of a yes/no decision support task, we may use the mean accuracy, i.e. the average of binary (1 for correct and 0 for wrong) returns, as the evaluation metric.

8.1.2 Data Set

To judge the “drugability” for target validation, we use the DisGeNET 2.0 data set [15], which contains 6,029 diseases and 9,313 genes integrated from a number of gene-disease association databases, including OMIM, UniProt, PharmGKB and CTD. We focus on the manually-created items and exclude the LHGDN subset, which contains gene-disease relations automatically extracted by text mining algorithms. We split the manually created data set into 5,605 subsets, each of which corresponds to one of the diseases. Then, we create a “positive” example task for each disease and a gene whose relation with the disease is known, and create a “negative” example task for each disease and one of the remaining (unrelated) genes from the 6158 candidates. We use all the positive examples, and sample the negative examples to create a balanced data set containing a total of 60,000 example tasks.

1 Online Mendelian Inheritance in Man Database, http://omim.org
3 Pharmacogenomics Knowledge Base, http://www.pharmgkb.org
4 Comparative Toxicogenomics Database, http://ctdbase.org
8.2 Analytics Procedure Definition: Analytics Procedure for Target Validation

We introduce a manually created decision process for target validation, which is then expanded into a full analytics procedure.

We create a simple target validation decision process to test our hypothesis, based on literature review as well as personal knowledge and experience from a group of professional biochemists and bioinformaticians working at a pharmaceutical company. We first define GENE and DISEASE as the variables in the target validation decision process template (DPT), which could then be instantiated with actual GENE and DISEASE names from the relations in the DisGeNET data set. We expand the template containing questions in the form of Jeopardy! clues in a top-down fashion, which, depending on the use case, can be treated as a yes/no question if both GENE and DISEASE are instantiated and a list question if only DISEASE is instantiated. Specifically, we start with defining the decision goal, which summarizes the target identification task from a high level:

The GENE is directly involved in DISEASE and can be a suitable target.

As one can easily see that necessary detailed criteria for a “druggable” target cannot be thoroughly depicted in the decision goal, the template was then expanded to a second level, which contains six important decision factors such as gene expression, gene mutation, pathway, clinical trials, etc., that is further decomposed as necessary. Finally, the decision process contains a total of 16 factors in three levels, which is shown in Figure 8.1. We can see from Figure 8.1 that some strong indicators can hardly be satisfied by a single gene, whereas some other weak factors only contribute to the decision if some gene satisfies all the criteria. We also pay special attention to the criteria 1.6 and 1.6.1, which intuitively are negative indicators. We do not specify any preferences to the decision processes as input, i.e. a uniform prior is used by default, and we count on the decision logic to decide what weights should be assigned when combining the factors.

We further expand the decision process to a full analytics procedure by decomposing each decision factor with an execution process similar to the biomedical question answering task procedure (as described in Section 7.3), which commonly consists of parsing, interpretation, retrieval, answer extraction, evidencing and ranking phases. Although we design to use the same general execution procedure to deal with all the sub analytics tasks at each decision factor, and we further utilize the same set of components to instantiate the execution process, the resulting analytics engines are still different and specific to each factor, due to the difference of configurations. We will detail this in the next section.

8.3 Analysis Component Construction

This section focuses on instantiating the analytics procedure with analysis components. We first describe the solution synthesis method, and then we describe the analysis components in the yes/no

\(^5\)Conventionally, a gene can refer to either a gene or a gene product such as protein or mRNA.
1 The GENE is directly involved in the DISEASE and can be a suitable target.

1.1 Any experiment showing that modulating the activity of the GENE with a chemical compound or genetic modification causes the DISEASE.

1.1.1 Any human in vivo experiment showing that modulating the activity of the GENE affects biochemical function or phenotype of the DISEASE.

1.1.2 Any in vitro experiment showing that modulating the activity of the GENE affects biochemical function or phenotype of the DISEASE.

1.1.3 Any animal model study showing that modulating the activity of the GENE in animals causes the DISEASE.

1.2 The GENE is expressed in the human tissue related to the DISEASE.

1.2.1 The GENE is expressed in normal human tissue related the DISEASE.

1.2.2 The GENE expression is altered in human DISEASE tissue or human DISEASE cell.

1.2.3 The alteration of the GENE expression is correlated with the DISEASE severity.

1.3 Any mutation is associated with the DISEASE.

1.3.1 Any mutation of the GENE has significantly associated with the DISEASE.

1.3.2 Any mutations in other genes linked to the GENE associated with DISEASE.

1.4 Any pathway involving the GENE supports that the GENE causes the DISEASE.

1.5 Any clinical trials show that targeting the GENE can prevent or slow the progress of the DISEASE.

1.6 Any evidence suggests that targeting the GENE will have side effects.

1.6.1 Targeting the GENE will cause liver, heart, and kidney damage.

Figure 8.1: Decision subtasks in the three-level target validation task
question answering pipeline, modified from the general biomedical question answering system (Section 7.4).

8.3.1 Solution Synthesis Method

A solution synthesis process summarizes the solutions from the factors into a single decision. For the yes/no decision support task such as the target validation task we study in this section, the solution synthesis module takes a binary (yes/no) answer from each solution, and outputs a binary value. We model it as a binary classification task, where we create the feature vector from the state representation of each decision factor, which can be different from the state representation used for value estimation and analytics space exploration.

In this section, we consider two meta information types – the current decision factor (CF), partial sub analytics processing trace (PT) – and an intermediate object, the factor-level answer output, to construct an indicator vector representing the last state of each decision factor, which is used as the input of the solution synthesis module. For example, a valid feature vector is

\[
\langle 1, \ 57eb7e04e1ef96c2bfa9f1a8d2e9ed04, \text{no} \rangle = 1,
\langle 1.1, \ 32d8585b3b9dc69e902d1c2f169fddef, \text{no} \rangle = 1,
\langle 1.1.1, b09fa92f1d33799f050b4f80df2e1c3e, \text{yes} \rangle = 1,
\]

... 

The features corresponding to the states that are not experienced in this episode have value of zero.

Similar to our discussion on state representation for analytics space exploration MDP in Section 5.8, we note that it is necessary to include the factor-level subtrace in the feature representation for solution synthesis, since the factor-level output depends not only on the input and the decision factor that is used to analyze the input, but also the detailed sub analytics engine (configuration) that is used to execute the input. In fact, one can use only the decision factor name and the factor output to represent the feature in a reinforcement learning environment, which slowly updates its optimal policy and also behaves as the optimal policy (i.e. on-policy), as long as an online (nonstationary) learning method is used. In our problem, since the exploration strategies tend to be more aggressive than the reinforcement learning environment, although introducing the factor-level subtrace into the feature representation may increase the dimensionality, we have observed improved learning performance in preliminary experiments over a denser feature representation without factor-level subtrace.

Here, we treat it as a black-box solution synthesis module, whose model and update rules are predefined in the analytics space construction phase. The analytics meta learning framework informs the synthesis solution components the reward it receives but does not further interfere its internal learning process. We also note that the solution synthesis module learning can be combined with the analytics meta learning task into a single framework.

For this case study, we use the logistic regression classifier with \( L_1 \) regularizer to learn the solution synthesis module.
Table 8.1: Yes/no QA analysis components

<table>
<thead>
<tr>
<th>Category</th>
<th>Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parsing</td>
<td>ClearNLP tokenizer</td>
</tr>
<tr>
<td>Interpretation</td>
<td>UMLS for disease syn/acronym expansion</td>
</tr>
<tr>
<td></td>
<td>EntrezGene for gene syn/acronym expansion</td>
</tr>
<tr>
<td></td>
<td>OMIM for disease syn/acronym expansion</td>
</tr>
<tr>
<td>Document retrieval / passage reranking</td>
<td>Lucene search engine</td>
</tr>
<tr>
<td>Evidencing and ranking</td>
<td>Document frequency (DF)</td>
</tr>
<tr>
<td></td>
<td>Retrieval/reranking score</td>
</tr>
</tbody>
</table>

8.3.2 Supporting Evidence based Yes/No QA Analysis Components

We deal with the analytics tasks represented as yes/no questions, i.e. either yes or no will be assigned to each decision factor. We list the key components of the QA pipeline in Table 8.1. We largely reuse the components from the general biomedical question answering system described in Chapter 7, including the tokenizer, concept identification / synonym expansion components, abstract query construction modules, and retrieval components. Document frequencies and the raw BM25 retrieval scores obtained from the document retrieval phase are used to determine the factor-level answer. Different from Yang et al. [221], we use a simple unsupervised learning method – thresholding to determine the yes/no answer. However, we only specify the possible values of the thresholds, and transfer the responsibility of determining the thresholds to the analytics meta learning framework, which can be considered as a way to ensemble the weak learners.

We use the identical sub configuration space for all the factors. However, since the modules may have difference performance for each factor, e.g. a concept identification algorithm may work the best for some factor description but not for others, or some condition is only considered as being met if we can find more evidence (higher document frequency) than other conditions, we allow the analytics meta learning to discover the best configuration (component and its parameter setting) customized to each factor, which is made possible by the state representation as a pair of factor name and configured component.

8.4 Analytics Space Exploration

In this section, we use the proposed analytics space exploration strategies to explore the analytics space for the target validation task, which involves of optimizing the configurations for each analysis components at each phase of the system as well as simultaneously learning the solution synthesis module to combine the decision factors.

225
Table 8.2: Description of configuration space for target validation task

<table>
<thead>
<tr>
<th>Phase #</th>
<th>Phase name / options</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>question-parse-concept</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>abstract-query</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>token-concept + concept: [required: [false, true]] + token</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>document-retrieval</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>passage-retrieval</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>passage-filter</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>noop + lucene-sentence</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>answer</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>df-threshold: [1, 10, 50] × score-threshold: [1, 10, 50]</td>
<td></td>
</tr>
</tbody>
</table>

8.4.1 Configuration Space

Since we are uncertain about the quality of concept identifier (and synonym expansion module), the effectiveness of the Lucene-based sentence reranking component, and most uncertain about the thresholds of document frequency and retrieval score to determine the answer for each decision factor, in this experiment, we vary the abstract query construction phase, passage filter phase, and most importantly the threshold-based answer generation phase. We list the components and configurations that were used in our experiment in Table 8.2. The resulting factor-level sub configuration space contains 72 combinations. Since we have defined 16 decision factors, each may have a different configuration, the total number of different combinations for this target validation task is

521, 578, 814, 501, 447, 328, 359, 509, 917, 696 \( (5.2 \times 10^{29}) \).

We use accuracy, i.e. 1 for a correct prediction and 0 otherwise, to measure the task performance, and define the runtime performance as the time duration that the component spends during each execution. To optimize the decision support system for target validation task, we apply each of the 40 budgeted policy optimization strategies proposed and compared in Section 5.7 for up to 4 hours, and repeat each setting for 5 times. We use current factor (CF) and current phase (CP) for state representation, instead of current phase (CP) alone in Sections 5.7 and 7.6. We deploy each strategy (5 exploration processes simultaneously) on an AWS EC2 instance of r4.2xlarge type (8 Intel Xeon E5-2686 v4 processors and 61 GB of DDR4 memory). Different from the case study on biomedical question answering task, which uses a separate test set to conduct offline evaluation, we measure the online performance of each strategy. In particular, for each checkpoint \( t \), we calculate the mean accuracy of the behavior policy using the average of actual returns within a time window of 2000 seconds before each checkpoint, i.e. \( [t - 2000, t] \). If the checkpoint \( t < 2000 \), we use all the actual returns from the first episode, i.e. \( [0, t] \). We also report the significant level over the comparand by t-test. We plot the learning curves measured by mean reward (and standard error), where we also use a sliding window of size 2000 seconds.
Table 8.3: Results of budgeted learning strategies with independent component selection for target validation

<table>
<thead>
<tr>
<th>Strategy</th>
<th>3m</th>
<th>10m</th>
<th>30m</th>
<th>1h</th>
<th>2h</th>
<th>3h</th>
<th>4h</th>
<th>Comparand</th>
</tr>
</thead>
<tbody>
<tr>
<td>TQ-εG/R</td>
<td>.3680</td>
<td>.4440</td>
<td>.5734</td>
<td>.6324</td>
<td>.6394</td>
<td>.6384</td>
<td>.7007</td>
<td>-</td>
</tr>
<tr>
<td>TQ-εG/D</td>
<td>.4927ns</td>
<td>.5173ns</td>
<td>.5441ns</td>
<td>.6096ns</td>
<td>.6215ns</td>
<td>.6471ns</td>
<td>.6878ns</td>
<td>TQ-εG/R</td>
</tr>
<tr>
<td>TQ-OFF/R</td>
<td>.4333ns</td>
<td>.5003ns</td>
<td>.5456ns</td>
<td>.5675ns</td>
<td>.6459ns</td>
<td>.6718ns</td>
<td>.5912ns</td>
<td>TQ-εG/R</td>
</tr>
<tr>
<td>TQ-OFF/D</td>
<td>.3907ns</td>
<td>.4167ns</td>
<td>.5329ns</td>
<td>.6306*</td>
<td>.6081ns</td>
<td>.6475ns</td>
<td>.6566ns</td>
<td>TQ-OFF/R</td>
</tr>
<tr>
<td>TQ-UCBE/R</td>
<td>.4200ns</td>
<td>.5640ns</td>
<td>.6509ns</td>
<td>.7057ns</td>
<td>.7646*</td>
<td>.7215ns</td>
<td>.7349ns</td>
<td>TQ-εG/R</td>
</tr>
<tr>
<td>TQ-UCBE/D</td>
<td>.5533ns</td>
<td>.5290ns</td>
<td>.5403*</td>
<td>.5725*</td>
<td>.6681**</td>
<td>.6516†</td>
<td>.6867ns</td>
<td>TQ-UCBE/R</td>
</tr>
<tr>
<td>TQ-J</td>
<td>.3273ns</td>
<td>.4420ns</td>
<td>.4651ns</td>
<td>.4922*</td>
<td>.5215*</td>
<td>.5029*</td>
<td>.5634†</td>
<td>TQ-εG/R</td>
</tr>
<tr>
<td>FA-εG/R</td>
<td>.4687ns</td>
<td>.5223ns</td>
<td>.4880ns</td>
<td>.4982*</td>
<td>.5358*</td>
<td>.5332*</td>
<td>.5570†</td>
<td>TQ-εG/R</td>
</tr>
<tr>
<td>FA-εG/D</td>
<td>.4513ns</td>
<td>.4353ns</td>
<td>.4681ns</td>
<td>.5162ns</td>
<td>.5179ns</td>
<td>.4940ns</td>
<td>.5765ns</td>
<td>FA-εG/R</td>
</tr>
<tr>
<td>FA-OFF/R</td>
<td>.3847ns</td>
<td>.5020ns</td>
<td>.5183ns</td>
<td>.5646ns</td>
<td>.5473ns</td>
<td>.5546ns</td>
<td>.5549ns</td>
<td>FA-εG/R</td>
</tr>
<tr>
<td>FA-OFF/D</td>
<td>.4847ns</td>
<td>.5100ns</td>
<td>.5489ns</td>
<td>.5789ns</td>
<td>.6130ns</td>
<td>.6328*</td>
<td>.6014ns</td>
<td>FA-OFF/R</td>
</tr>
<tr>
<td>FA-UCBE/R</td>
<td>.5007ns</td>
<td>.5313ns</td>
<td>.5173ns</td>
<td>.5911†</td>
<td>.6021ns</td>
<td>.5861ns</td>
<td>.5769ns</td>
<td>FA-OFF/R</td>
</tr>
<tr>
<td>FA-UCBE/D</td>
<td>.4947ns</td>
<td>.4987ns</td>
<td>.4918ns</td>
<td>.5383ns</td>
<td>.5819ns</td>
<td>.6062ns</td>
<td>.6438†</td>
<td>FA-UCBE/R</td>
</tr>
<tr>
<td>FA-J</td>
<td>.4600ns</td>
<td>.4857ns</td>
<td>.4836†</td>
<td>.4589†</td>
<td>.5193ns</td>
<td>.4981ns</td>
<td>.5313ns</td>
<td>FA-εG/R</td>
</tr>
</tbody>
</table>

8.4.2 Comparison of Analytics Space Exploration Strategies

At each budget checkpoint (3 minutes, 10 minutes, 30 minutes, 1 hour, 2 hours, 3 hours, and 4 hours), we report the mean accuracy in Tables 8.3 and 8.4 for those using independent and two-step option selection methods respectively, and show the moving-averaged mean reward curves in Figures 8.2, 8.3, 8.4, and 8.5.

First, we can see that the best performance obtained by any strategy is close to or slightly above 0.75 in terms of mean accuracy, which we believe is the global optimum performance of the analytics space. In other words, the developers need to design alternative analytics procedures and/or improve the quality of analysis components in the analytics space in order to further improve the performance. We can also see that the variance is much higher than that of the biomedical question answering task, due to the much larger configuration space and additional uncertainty introduced by the solution synthesis component, which trains its own learning model simultaneously on the fly. However, we can still clearly compare the performance trends of the strategies by their performance means.

When an independent option selection method is used, as shown in Figures 8.2 and 8.3, we can see that the performance of most strategies is improved steadily during the entire 4 hours of exploration process. We compare between the strategies using the tabular MC control (TQ) method and the linear MC control (FA) method. We see that the TQ strategies can achieve higher performance than the FA strategies, where most of the former ones have a terminal performance of above 0.65 and the latter ones can hardly find a system that has a mean accuracy of 0.65 or above. Actually, although both the TQ and FA strategies use the same state representation, the pair of current factor (CF) and current phase (CP), the TQ strategies treat each pair as a single key in the
value table, whereas the FA strategies see them as two independent feature dimensions in a vector space. In our case that the analytics space contains 16 factors and specifies a 5-phase pipeline for each factor, the TQ strategies define a discrete state space that has $16 \times 5 = 80$ different states, whereas the FA strategies define a $16 + 5 = 21$-dimensional vector space. From the results, we can see that the discrete state space used by the TQ strategies can better distinguish the integrated components of different performance.

The best performance among those using an independent option selection method is achieved by the TQ-UCBE/R and TQ-UCBE/D strategies, both being able to diversity the selected components, especially at the beginning of the exploration process, compared with other strategies, and thus have relatively high performance at the initial checkpoint. Between the two best strategies, the

Figure 8.2: Learning curves of TQ strategies w/ independent option selection for target validation

Figure 8.3: Learning curves of FA strategies w/ independent option selection for target validation
Table 8.4: Results of budgeted learning strategies with random phase selection for target validation

<table>
<thead>
<tr>
<th>Strategy</th>
<th>3m ns</th>
<th>10m ns</th>
<th>30m ns</th>
<th>1h ns</th>
<th>2h ns</th>
<th>3h ns</th>
<th>4h ns</th>
<th>Comparand</th>
</tr>
</thead>
<tbody>
<tr>
<td>TQ-RR/R</td>
<td>.4067</td>
<td>.4130</td>
<td>.5259</td>
<td>.6479</td>
<td>.6732</td>
<td>.6363</td>
<td>.6800</td>
<td>TQ-εG/R</td>
</tr>
<tr>
<td>TQ-RR/D</td>
<td>.3247</td>
<td>.5270</td>
<td>.5627</td>
<td>.6377</td>
<td>.6876</td>
<td>.6896</td>
<td>.6984</td>
<td>TQ-εG/D</td>
</tr>
<tr>
<td>TQ-RU1/R</td>
<td>.5167</td>
<td>.5123</td>
<td>.5921</td>
<td>.6359</td>
<td>.6780</td>
<td>.6927</td>
<td>.7314</td>
<td>TQ-UCBE/R</td>
</tr>
<tr>
<td>TQ-RU1/D</td>
<td>.3513</td>
<td>.4403</td>
<td>.6317</td>
<td>.7244</td>
<td>.7392</td>
<td>.7336</td>
<td>.7430</td>
<td>TQ-UCBE/D</td>
</tr>
<tr>
<td>TQ-RUE/R</td>
<td>.3993</td>
<td>.5403</td>
<td>.6484</td>
<td>.6834</td>
<td>.7332</td>
<td>.7246</td>
<td>.7521</td>
<td>TQ-UCBE/R</td>
</tr>
<tr>
<td>TQ-RUE/D</td>
<td>.3880</td>
<td>.4923</td>
<td>.6274</td>
<td>.7280</td>
<td>.6847</td>
<td>.7065</td>
<td>.7356</td>
<td>TQ-UCBE/D</td>
</tr>
<tr>
<td>TQ-RJ</td>
<td>.4740</td>
<td>.5327</td>
<td>.6118</td>
<td>.6202</td>
<td>.6551</td>
<td>.6667</td>
<td>.6204</td>
<td>TQ-J</td>
</tr>
<tr>
<td>FA-RR/R</td>
<td>.4553</td>
<td>.5037</td>
<td>.5292</td>
<td>.5856</td>
<td>.6036</td>
<td>.5839</td>
<td>.5909</td>
<td>FA-εG/R</td>
</tr>
<tr>
<td>FA-RR/D</td>
<td>.4620</td>
<td>.5090</td>
<td>.5000</td>
<td>.5826</td>
<td>.6526</td>
<td>.6411</td>
<td>.6731</td>
<td>FA-εG/D</td>
</tr>
<tr>
<td>FA-RU1/R</td>
<td>.4207</td>
<td>.5057</td>
<td>.5878</td>
<td>.6687</td>
<td>.6620</td>
<td>.6965</td>
<td>.7348</td>
<td>FA-UCBE/R</td>
</tr>
<tr>
<td>FA-RU1/D</td>
<td>.4467</td>
<td>.4553</td>
<td>.5070</td>
<td>.5580</td>
<td>.5880</td>
<td>.6053</td>
<td>.6362</td>
<td>FA-UCBE/D</td>
</tr>
<tr>
<td>FA-RUE/R</td>
<td>.4553</td>
<td>.4727</td>
<td>.5187</td>
<td>.6005</td>
<td>.5942</td>
<td>.6347</td>
<td>.6236</td>
<td>FA-UCBE/R</td>
</tr>
<tr>
<td>FA-RUE/D</td>
<td>.4820</td>
<td>.4940</td>
<td>.5276</td>
<td>.5859</td>
<td>.6843</td>
<td>.7113</td>
<td>.6998</td>
<td>FA-UCBE/D</td>
</tr>
<tr>
<td>FA-RJ</td>
<td>.4093</td>
<td>.4630</td>
<td>.5418</td>
<td>.6043</td>
<td>.6392</td>
<td>.6680</td>
<td>.6314</td>
<td>FA-J</td>
</tr>
</tbody>
</table>

TQ-UCBE/R strategy exhibits better performance than the TQ-UCBE/D strategy, its variant using the density ordered heuristic. However, we note that the accuracy scores and the reward curves measure the online performance of the behavior policy, i.e. the actual rewards collected during the exploration process, which can be different from the expected optimal policy curve, which measures the performance of the estimation policy. Since the difference between the estimation policy and the behavior policy is larger when using the TQ-UCBE/D strategy (the highest-density option can be off-policy) than the TQ-UCBE/R (the highest-reward option is on-policy), the actual expected optimal performance of the estimation policy could be higher than the behavior performance as shown in Figure 8.3, especially when using the TQ-UCBD/D strategy. The TQ-J method arbitrarily selects an option at each step, which, although is sufficient to determine the worst-performing component for the purpose of option elimination, makes the learning of the solution synthesis model difficult due to sparsity issue.

When we use two-step option methods, we can see from Figures 8.4 and 8.5 that the performance of all the TQ and FA strategies is improved, including the TQ/J strategy, which uses the most aggressive exploration strategy. The TQ strategies again outperform the FA strategies, and have much better convergence property. In fact, we can see that most FA strategies are able to converge to the optimum performance after only 30 minutes. We conclude that the full analytics meta learning task that combines individual analysis component learning is much more challenging than the pure configuration space exploration task, since we have to find the optimal aggressiveness of exploration that should allow (1) the value estimation method to estimate the value of each component correctly, (2) the entire analytics space to be explored efficiently, and (3) the individual analysis components to update their own models effectively. We see that a two-step option selection method best suits this scenario.

229
As discussed in Section 5.8, one difference between the full analytics space exploration MDP and the configuration space exploration MDP is the inclusion of factor name in the state representation, which effectively distinguishes the same analysis component being used in two different factors that require difference configurations. We compare the learning curves of TQ strategies with a two-step option selection method that use both the current factor (CF) and the current phase (CP) for state representation (Figure 8.4) with those that use only the current phase (CP) for state representation (Figure 8.6). We see that the latter ones exhibit less stability even though they can normally collect more episodes given the same budget, where some strategies are stuck in a local optimum while others cannot converge.

Finally, we note the online performance metric, such as the mean accuracy and/or moving-
averaged reward, can accurately reflect the actual performance of the system and does not suffer from the overfitting problem, although we use the same development set for both training and testing. In fact, a reinforcement learning setting assumes that there are infinitely number of possible episodes when the agent experiences a real world environment and any agent can hardly experience and remember all of the scenarios. In our case, we have 60,000 different example tasks in the problem set and $60,000 \times 5.2 \times 10^{29} = 3.1 \times 10^{34}$ number of different combinations, which makes it very unlikely to sample exactly the same input and the configuration during an exploration process within no more than 2,000 episodes, even if we perform exactly greedily. Moreover, the individual analysis component first makes a prediction to complete its execution task, before it later receives the feedback from the terminal state to update its model, similar to an interactive learning scenario.

8.5 Conclusion

In this chapter, we introduce a decision support task in drug discovery – target validation. We formally define the use case of the problem by showing inputs and expected outputs, and present the manually created analytics procedure for target validation, which includes a decision process and a simple generic execution process for each decision factor. Then, we describe the analysis components in the analytics space, which includes a machine learning solution synthesis algorithm as well as the components that we adapt from the biomedical question answering system developed in Chapter 7. We apply a full analytics meta learning approach to optimize target validation decision support system. In this experiment, the system is able to update the meta learning model simultaneously while it trains the lower-level individual machine learning components such as the solution synthesis module. We find the strategies that use the two-step option selection method and tabular MC control for policy optimization achieve the best performance. The resulting system can achieve a performance of 0.75 in terms of mean accuracy.
An interesting extension to this chapter is to consider a jointly learning architecture that combines analytics meta learning and the solution synthesis learning, which uses a unified state representation and value estimation method. In fact, our proposed logistic regression based solution synthesis method uses a triple of factor name, partial trace, and the content in the intermediate content (the answer) to represent the final state of each factor, and employs a specialized model update method.
Chapter 9

Case Study: Product Recommendation Task

This chapter continues the study of analytics meta learning applications, and focuses on another use case that involves complex problem solving procedure – product recommendation task. Unlike the target validation problem, algorithmic methods for the product recommendation problem (as well as the related rating problem) have been extensively investigated [127]. In this section, rather than propose any new algorithm for this relatively well-studied problem, we focus on utilizing analytics meta learning methodology to construct an information system from automatically discovered decision processes and existing information processing components, in order to test its flexibility and generalizability.

In Section 9.1, we first give a formal definition to the product recommendation task, and in Section 9.2 we describe to leverage purchase guide documents to extract human decision processes for phone recommendation task. In Section 9.3, we describe the analysis components that we have developed for the phone recommendation task. In Section 9.4, we propose to apply a full analytics meta learning approach to construct and optimize a general-purpose product recommendation system using review texts, which involves exploring the configuration space at the factor level, learning individual analysis components, and selecting the decision processes that work the best for the phone recommendation task. We compare the performance between the 28 value-based budgeted exploration strategies. The optimal system can achieve an accuracy of above 0.93. We conclude in Section 9.5.

9.1 Task Description

Similar to the pharmaceutical decision task introduced in Chapter 8 and formally defined in Section 8.1, the product recommendation task is an analytics task that real world users have to deal with every day online and offline. For example, one can tell the system the decision needs in natural language, e.g.

Is buying an iPhone 5 recommended?

or
Which PHONE is recommended to purchase?

and the answer can be

yes

or

iPhone 6s, Samsung Galaxy s6, etc.

Users also expect to find out the supporting evidence from the review texts (e.g. user experience, quantity, etc.) and/or specification database (e.g. weight, color, operating system, etc.).

9.1.1 Evaluation Metrics

Since this task is more subjective than target validation, we can hardly rely on a manually-created gold-standard product recommendation list. Therefore, we use the average rating of each product on Amazon\(^1\) which is then converted from a 5-point rating scale to a binary recommendation. Specifically, ratings equal to or greater than 3 are assigned yes, otherwise no. In this way, we can determine whether a particular product is recommended by the known buyers, which is compared with the system prediction. Thus, we can utilize accuracy, the same evaluation metric described in Chapter 8, to evaluate the pipeline’s task performance.

9.1.2 Data Set

We employ the Amazon product review corpus \(^2\). Various product reviews give insights about the same product from different perspectives, in the same way that different publications in PubMed discuss different perspectives of the same gene/disease. Similar to the preparation procedure for PubMed abstract corpus, we here focus on the review/text field in each review and tag with productId. To conduct the experiment for phone recommendation, we use the Cell Phones & Accessories subset. We further eliminate the reviews for phone accessories from the data set by selecting the products whose title ends with any of

phone (at&t), phone (t-mobile), phone (verizon wireless), phone (sprint), phone (unlocked),

where case is ignored. Although this method may also eliminate some products that are also phones, it guarantees the quality of remaining subset.

We might also consider other fields such as userId or time, which provide supplementary information for the recommendation task. However, incorporating structured data with a different schema often requires extra effort to adapt existing QA pipelines.

\(^1\)http://www.amazon.com
\(^2\)http://snap.stanford.edu/data/web-Amazon-links.html
9.2 Analytics Procedure Definition

In this section, we propose to take advantage of purchase guide articles to extract human decision processes for phone recommendation.

Community how-to guide websites discourage users to create duplicate how-to manuals, instead they encourage them to create multiple Methods under the same article. We identified three relevant wikiHow articles for the search query “how to choose a phone”:

- **How to Choose a Cell Phone** ([http://www.wikihow.com/Choose-a-Cell-Phone](http://www.wikihow.com/Choose-a-Cell-Phone)),
- **How to Choose a Smartphone** ([http://www.wikihow.com/Choose-a-Smartphone](http://www.wikihow.com/Choose-a-Smartphone)),
- **How to Choose Between Iphone and Android Smartphones** ([http://www.wikihow.com/Choose-Between-Iphone-and-Android-Smartphones](http://www.wikihow.com/Choose-Between-Iphone-and-Android-Smartphones)).

We extract 38 factors from the first article, 32 factors from the second article, and 15 factors from the third article. If we keep only the top three levels of decision hierarchy, we obtain 18 factors from the first article and 16 factors from the second article, and if we further allow the top two levels, we obtain 4 factors for the first article, 3 factors for the second article, and 8 factors for the third article. Totally, we obtain 8 different decision processes with a total of 144 factors. The same decision factor in different decision processes are treated as different factors, since each may play a different role in the decision process, e.g. a decision process that lacks detailed atomic or low-level factors has to rely on high-level (abstract) factors to make the decision, and it puts less weight on them once the detailed factors are defined. Different from the target validation evaluation set, the phone recommendation data set is slightly imbalanced, where the positive to negative ratio is around 2:1.

9.3 Analysis Component Construction

We adapt the same set of analysis components from the target validation task in Chapter 8 to construct the analytics space for the product recommendation task. We made a few necessary modifications. First, we use the pretrained general English model to replace the bioinformatics model to parse natural language task descriptions. Then, we apply the keyphrase extraction model proposed in Section 3.4 to extract the actionable phrases from action description texts in a wikiHow article and treated them as key “concepts”. We also create a review index for the phones using Lucene to replace the Medline index.

When we design the analytics procedure for the target validation task in Chapter 8 each decision factor has a last analysis step that generates a factor-level yes/no answer based on the thresholds on the document frequency and the retrieval score, and a separate solution synthesis method learns to summarize these factor-level scores to produce the final yes/no decision. In order to determine whether a user is satisfied with each aspect of a phone, we further leverage a sentiment analysis module, e.g. a sentiment word dictionary to detect the sentiment of each passage, and introduce additional thresholds, e.g. the number of negative passages and the percentage

[http://www.cs.uic.edu/~liub/FBS/opinion-lexicon-English.rar](http://www.cs.uic.edu/~liub/FBS/opinion-lexicon-English.rar)
of negative passages, to determine the answer at each factor level. However, this would put much
pressure on the analytics meta learning framework in determining the optimal real-valued thresh-
OLds from user-specified discrete threshold options. To overcome this problem, we use a solution
synthesis module that directly learns to combine the raw retrieval and sentiment analysis scores
from each factor to produce a binary output, which replaces the factor-level answer generation
phase and the solution synthesis module.

To train the solution synthesis model, we define the features (i.e. state representations) from
the meta information and/or the intermediate content. Similar to the answer synthesis model used
in Section 8.3, we may use the meta information, e.g. the partial trace (PT) alone, the current
factor (CF) alone, the current phase and factor combined or the concatenated, where the former
(“combined”) defines two dimensions in the feature vector for linear MC control and the latter
(“concatenated”) defines one dimension in the feature vector for linear MC control, as well as
the retrieval and sentiment analysis results. We define seven features to characterize the retrieval
and sentiment analysis results, including negative sentiment count, nonnegative sentiment count,
maximum retrieval score of negative passage, maximum retrieval score of nonnegative passage,
total document frequency, negative sentiment ratio, nonnegative sentiment ratio. For example, if
we use the current factor, partial trace, and the seven intermediate data features, we may obtain a
feature vector as follows:

\[
\langle 1, 57eb7e04e1ef96c2bfa9f1a8d2e9ed04, \text{negative sentiment ratio} \rangle = 0.6,
\langle 1, 57eb7e04e1ef96c2bfa9f1a8d2e9ed04, \text{nonnegative sentiment ratio} \rangle = 0.4,
\ldots
\langle 1.1, 32d8585b3b9dc69e902d1c2f169fddef, \text{negative sentiment ratio} \rangle = 0.2,
\langle 1.1, 32d8585b3b9dc69e902d1c2f169fddef, \text{nonnegative sentiment ratio} \rangle = 0.8,
\ldots
\]

We may use any classification algorithm, e.g. logistic regression, decision tree, etc., to learn
from the features and feedback. In Section 9.4, we use the current factor, partial trace, the seven
intermediate data features, and logistic regression classifier to train a solution synthesis model for
analytics meta learning, and compare the results of different strategies. We also compare the so-
lution synthesis models by incorporating different feature representation methods and classifiers.
Furthermore, we specify these solution synthesis models as options and let the analytics meta learn-
ing framework choose the best solution synthesis model for the task. We use the 3rd-party Java
reimplementation of the LibLinear toolki4 for logistic regression, and use the decision tree imple-
mentation from the SMILE projecti5 where we use the default configuration for both classifiers,
e.g. maximum of 100 nodes in the decision tree.

i4 https://github.com/bwaldvogel/liblinear-java
i5 https://halfenengl.github.io/smile/
Table 9.1: Description of analytics space for phone recommendation task

<table>
<thead>
<tr>
<th>Phase #</th>
<th>Phase name / options</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>product-reduction</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>question-parse-concept</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>abstract-query</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>token-concept + concept: {required: [false, true]} + token</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>document-retrieval</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>passage-retrieval</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>passage-filter</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>noop + lucene-sentence</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>solution-synthesis</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>logistic-regression: {features: [C, FT, FTC]}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>decision-tree: {features: [C, FT, FTC]}</td>
<td></td>
</tr>
</tbody>
</table>

9.4 Analytics Space Exploration

In this section, we explore the analytics space defined for the product recommendation task using the proposed exploration strategies. We also compare the solution synthesis models manually, by incorporating different feature representation methods and classifiers, and automatically, by specifying these solution synthesis models as options and let the analytics meta learning framework choose the best solution synthesis model for the task.

9.4.1 Analytics Space

We list the components and configurations that are used in our experiment in Table 9.1. The resulting factor-level sub configuration space contains 8 combinations from phases 2–6. We also have eight options specified for the product reduction phase, each corresponding to one of the eight decision processes. We define six solution synthesis methods in Phase 7, where we use C to represent the combined current factor and partial trace (two separate features), FT to represent the concatenated current factor and partial trace (a single feature), and FTC to represent the combined current factor, partial trace, and their concatenation (three features in total). When we combine with two classifiers, logistic regression and decision tree, we obtain six solution synthesis methods. In the first part of this section, we focus on a fixed solution synthesis method – the logistic regression method with combined current factor and partial trace representation. Since we have 38, 32, 15, 18, 16, 4, 3, and 8 factors in each of the 8 decision processes, the total number of different combinations for this phone recommendation task is

$$20,769,266,662,301,824,796,790,636,735,900,065 \times 10^{31}.$$ 

In the second part of this section, we further specify all the solution synthesis methods as options and let the analytics meta learning framework learn and select the optimal method, which defines $1.7 \times 10^{35}$ combinations.
### Table 9.2: Results of budgeted learning strategies with independent component selection for product recommendation

<table>
<thead>
<tr>
<th>Strategy</th>
<th>3m</th>
<th>10m</th>
<th>30m</th>
<th>1h</th>
<th>2h</th>
<th>3h</th>
<th>4h</th>
<th>Comparand</th>
</tr>
</thead>
<tbody>
<tr>
<td>TQ-(\epsilon) G/R</td>
<td>.0920</td>
<td>.1177</td>
<td>.1549</td>
<td>.1798</td>
<td>.2047</td>
<td>.7205</td>
<td>.7489</td>
<td>-</td>
</tr>
<tr>
<td>TQ-(\epsilon) G/D</td>
<td>.5300**</td>
<td>.5560*</td>
<td>.5998*</td>
<td>.7170*</td>
<td>.7919*</td>
<td>.7486ns</td>
<td>.7147ns</td>
<td>TQ-(\epsilon) G/R</td>
</tr>
<tr>
<td>TQ-OFF/R</td>
<td>.4227†</td>
<td>.4770†</td>
<td>.6228*</td>
<td>.7655*</td>
<td>.8575*</td>
<td>.8846*</td>
<td>.8763ns</td>
<td>TQ-OFF/R</td>
</tr>
<tr>
<td>TQ-OFF/D</td>
<td>.3500ns</td>
<td>.3747ns</td>
<td>.4650ns</td>
<td>.6650ns</td>
<td>.7656ns</td>
<td>.8517ns</td>
<td>.8645ns</td>
<td>TQ-OFF/R</td>
</tr>
<tr>
<td>TQ-UCBE/R</td>
<td>.4927</td>
<td>.5310*</td>
<td>.5481†</td>
<td>.6392†</td>
<td>.7291*</td>
<td>.7373ns</td>
<td>.7259ns</td>
<td>TQ-OFF/R</td>
</tr>
<tr>
<td>TQ-UCBE/D</td>
<td>.5420ns</td>
<td>.5337ns</td>
<td>.5370ns</td>
<td>.6464ns</td>
<td>.7330ns</td>
<td>.7368ns</td>
<td>.7370ns</td>
<td>TQ-UCBE/R</td>
</tr>
<tr>
<td>TQ-J</td>
<td>.4093*</td>
<td>.4727*</td>
<td>.4849ns</td>
<td>.5118ns</td>
<td>.5488†</td>
<td>.5716*</td>
<td>.6001†</td>
<td>TQ-(\epsilon) G/R</td>
</tr>
<tr>
<td>F A-(\epsilon) G/R</td>
<td>.3773*</td>
<td>.4090†</td>
<td>.4799ns</td>
<td>.5771†</td>
<td>.6662*</td>
<td>.7307ns</td>
<td>.6973ns</td>
<td>TQ-(\epsilon) G/R</td>
</tr>
<tr>
<td>F A-(\epsilon) G/D</td>
<td>.1053†</td>
<td>.1150†</td>
<td>.0948*</td>
<td>.1420*</td>
<td>.4666†</td>
<td>.6875ns</td>
<td>.7327ns</td>
<td>F A-(\epsilon) G/R</td>
</tr>
<tr>
<td>F A-OFF/R</td>
<td>.4413ns</td>
<td>.4853†</td>
<td>.5831ns</td>
<td>.7077†</td>
<td>.8188†</td>
<td>.8629ns</td>
<td>.8732*</td>
<td>F A-(\epsilon) G/R</td>
</tr>
<tr>
<td>F A-OFF/D</td>
<td>.1173*</td>
<td>.1153*</td>
<td>.1122**</td>
<td>.1500**</td>
<td>.6556†</td>
<td>.7291ns</td>
<td>.8729ns</td>
<td>F A-OFF/R</td>
</tr>
<tr>
<td>F A-UCBE/R</td>
<td>.4787ns</td>
<td>.5147*</td>
<td>.5380ns</td>
<td>.6354†</td>
<td>.6474ns</td>
<td>.6789ns</td>
<td>.6701ns</td>
<td>F A-(\epsilon) G/R</td>
</tr>
<tr>
<td>F A-UCBE/D</td>
<td>.4287ns</td>
<td>.4737ns</td>
<td>.5121ns</td>
<td>.5550*</td>
<td>.5898ns</td>
<td>.5930*</td>
<td>.6086ns</td>
<td>F A-UCBE/R</td>
</tr>
<tr>
<td>F A-J</td>
<td>.5073ns</td>
<td>.4987ns</td>
<td>.5094ns</td>
<td>.4955**</td>
<td>.5225*</td>
<td>.5759*</td>
<td>.6143ns</td>
<td>F A-(\epsilon) G/R</td>
</tr>
</tbody>
</table>

We use the same computing environment to conduct the exploration experiments for the phone recommendation task as for the target validation task in Section 8.4.

### 9.4.2 Comparison of Analytics Space Exploration Strategies

We report the results in Tables 9.2 and 9.3 and Figures 9.1, 9.2, 9.3, and 9.4. From the progress curves, we see that all the strategies are able to collect much more episodes given the same budget (4 hours), which is mainly because the Lucene index for the phone related reviews is much smaller than the index for the entire PubMed abstract corpus, and thus it takes less time to process a retrieval task. From the reward curves, we see that this task can achieve much higher accuracy than the target validation task, as most of the TQ strategies using a two-step option selection method produce a mean accuracy above 0.91 at the final checkpoint. Moreover, we see most strategies gradually improve the behavior performance throughout the entire exploration process until converge to the maximum. The variance of behavior performance is also smaller than that in the target validation task, especially when it converges (e.g. when it is close to 4 hours in Figure 9.3). We can expect that the actual expected optimal performance may have a much lower variance than the behavior performance since it does not explore the non-greedy options at all.

When we compare the exploration strategies that use an independent option selection method, at first glance, we see that the TQ-\(\epsilon\) G/R, F A-\(\epsilon\) G/D, and F A-OFF/D strategies show extremely poor performance (< 0.2) at the checkpoints before 1 hour. However, if we compare the progress curves and the learning curves, we see that they actually collect the fewest episodes among all the strategies. For example, if we compare the performance of the FA strategies when they all collect the
same number of episodes, say 1000, they actually exhibit similar performance. Therefore, we see that cost is an important factor in budgeted analytics meta learning problem. We also have a similar finding as in Section 8.4 that using a density ordered heuristic may hurt the performance since it generates episodes that contain mostly non-greedy off-policy options and does not help the solution synthesis model learning. However, we still encourage an off-policy learning paradigm and recommend the TQ-OFF/R or FA-OFF/R strategies. They significantly outperform the comparands when we specify a moderate budget (1–2 hours) before all the strategies converge. We also discourage aggressive exploration such as random option selection method, e.g. Successive Rejects (J), since it generates a very sparse feature space with insufficient training instances (with respect to the dimensionality of the feature space), and therefore, we see that both the TQ-J and FA-J strategies can only achieve a mean accuracy of above 0.6 when they exhaust the entire 4-hour budget, but they still make steady improvement up until almost the last minute.

When we use a two-step option selection method for either a TQ strategy or a FA strategy, we see from Table 9.3 that they have exhibited significantly better performance than their counterparts using an independent option selection method, at all the checkpoints after the first hour when they start to converge. Again, we see some strategies using the density ordered heuristic, such as the TQ-RR/D and FA-RU1/D strategies have much lower performance before all the strategies converge. We also find that the TQ strategies tend to converge to the expected optimal performance while the FA strategies converge to a relatively larger region. The best final performance is achieved by the TQ-RU1/D and FA-RU1/D strategies, although the FA-RU1/D strategy has the lowest performance at earlier checkpoints.
Figure 9.2: Learning curves of FA strategies w/ independent option selection for phone recommendation

Figure 9.3: Learning curves of TQ strategies w/ 2-step option selection for phone recommendation

9.4.3 Comparison of Solution Synthesis Methods

We compare the solution synthesis methods using the same exploration strategy TQ-RUE/R, and plot the results in Figure 9.5. We see that the logistic regression classifier has better performance than the decision tree classifier regardless of the feature representation. When the learning method (logistic regression or decision tree) is fixed, feature representation affects the convergence rate but does hardly change the optimum. For example, when we define a higher dimensional feature space (such as DT/FTC and LR/FTC), the training process requires more episodes and takes longer time. When we compare the fixed solution synthesis methods with the optimal solution synthesis
Table 9.3: Results of budgeted learning strategies with random phase selection for phone recommendation

<table>
<thead>
<tr>
<th>Strategy</th>
<th>3m</th>
<th>10m</th>
<th>30m</th>
<th>1h</th>
<th>2h</th>
<th>3h</th>
<th>4h</th>
<th>Comparand</th>
</tr>
</thead>
<tbody>
<tr>
<td>TQ-RR/R</td>
<td>.4280*</td>
<td>.4867*</td>
<td>.6323*</td>
<td>.8203*</td>
<td>.8705*</td>
<td>.8789*</td>
<td>.8802†</td>
<td>TQ-eG/R</td>
</tr>
<tr>
<td>TQ-RR/D</td>
<td>.3560ns</td>
<td>.3720ns</td>
<td>.413ns</td>
<td>.6220ns</td>
<td>.6989ns</td>
<td>.6953ns</td>
<td>.8718*</td>
<td>TQ-eG/D</td>
</tr>
<tr>
<td>TQ-RU1/R</td>
<td>.4127ns</td>
<td>.4527ns</td>
<td>.5830ns</td>
<td>.7230ns</td>
<td>.7508ns</td>
<td>.9102***</td>
<td>.9175***</td>
<td>TQ-UCBE/R</td>
</tr>
<tr>
<td>TQ-RU1/D</td>
<td>.5087***</td>
<td>.5497***</td>
<td>.6676*</td>
<td>.8478***</td>
<td>.9126**</td>
<td>.9218**</td>
<td>.9211***</td>
<td>TQ-UCBE/D</td>
</tr>
<tr>
<td>TQ-RU2/R</td>
<td>.3927ns</td>
<td>.4690ns</td>
<td>.6866†</td>
<td>.8944***</td>
<td>.9141***</td>
<td>.9120***</td>
<td>TQ-UCBE/R</td>
<td></td>
</tr>
<tr>
<td>TQ-RU2/D</td>
<td>.4613ns</td>
<td>.5007ns</td>
<td>.6332*</td>
<td>.8619***</td>
<td>.9092**</td>
<td>.9135**</td>
<td>.9172***</td>
<td>TQ-UCBE/D</td>
</tr>
<tr>
<td>TQ-RJ</td>
<td>.4467*</td>
<td>.4620*</td>
<td>.6003*</td>
<td>.8278*</td>
<td>.8705*</td>
<td>.8870*</td>
<td>.9151*</td>
<td>TQ-J</td>
</tr>
</tbody>
</table>

automatically and dynamically selected by the analytics meta learning framework, we see that the latter has the lowest initial performance. It is because each solution synthesis can collect training instances only when it is selected, and thus each specified option tends to have fewer instances than the fixed solution synthesis model given the same budget. However, as it continues to learn the performance of each solution synthesis method, it can effectively avoid the low-performing methods. In fact, we can see that it outperforms all the decision tree based solution synthesis methods within less than an hour, and its performance is still improved before the 4-hour budget is used up.

Since we experiment with these two solution synthesis methods (logistic regression classifier and decision tree classifier) only in this case study, it would be difficult to tell whether the phenomenon that the logistic regression method outperforms the decision tree in solution synthesis is universal or task specific. In fact, there might be other factors that have involved in the difference of performance. For example, a different implementation and/or parameter setting may exhibit different performance. It would be interesting to conduct similar experiments for more analytics tasks. Moreover, the objective of single-analytics meta learning is to treat each option as a black box and estimate the cost and benefit of each option, rather than to understand why it works or doesn’t from the first principle. A similar problem, namely multi-analytics meta learning or transferring analytics meta learning, which we do not study in this thesis, attempts to understand the relation between the option performance and the task, and make prediction whether an option has a universally high performance or it only works with certain types of tasks. We leave multi-analytics

6In the future, we can also create a solution synthesis phase that broadcasts the actual gold-standard label at the end of the episode to all the solution synthesis options.
meta learning for future work.

We further analyze how the optimal solution synthesis method is found by the analytics meta learning framework. In Figure 9.6, we show the actual solution synthesis method selected by the behavior policy and the expected optimal solution synthesis method given by the estimation policy at each episode within the first hour of the exploration process, where the x-axis shows the elapsed episode and the six color represents the six solution synthesis methods. We see that it selects a logistic regression method as the optimal option until the 23rd episode when it changes to select a decision tree method. Then, the analytics meta learning focuses on estimating the value of this decision tree method. At the 146th episode, it determines that the value of this method is lower than the previously estimated optimal method and thus switched back to the logistic regression method.
Figure 9.6: The actual selected and the expected optimal solution synthesis method at each episode following the TQ-RUE/R strategy with different solution synthesis methods specified as options.

method. From then on, the framework continues to increase the estimated value of this logistic regression method, except that it prefers an exploratory option DT/FTC twice for 10 episodes and 2 episodes each due to randomness.

9.4.4 Summary

When we compare the performance of budgeted exploration strategies in these two case studies that involve full analytics space exploration (target validation task and the phone recommendation task) with the biomedical question answering task (configuration space exploration), we suggest that

• one should avoid using aggressive exploration strategies such as Successive Rejects (J),

• one should also avoid the behavior policies that are “too off” the optimal policy, such as those using the density ordered heuristic,

• a value-based value estimation method, such as tabular MC control or linear MC control, accompanied with a two-step option selection is a safe choice.

9.5 Conclusion

In this chapter, we introduce the product recommendation task. We first give a formal definition to the product recommendation task, and then perform the proposed automatic procedure construction algorithm to define analytics procedures for phone recommendation. Then, we collect product recommendation related data sources and reuse or develop analysis components to each specific analysis processing task. Finally, we explore the analytics space that contains over $10^{15}$ information systems, which involves exploring the configuration space at the factor level, learning individual analysis components, and selecting the decision processes that work the best for the
phone recommendation task. Despite the simplicity of the analysis components being integrated
into the analytics space, the analytics space exploration strategies can still find the optimal system
whose expected performance can reach 0.93 in terms of mean accuracy.
Chapter 10

Conclusion & Future Directions

In this chapter, we first summarize the main results of this thesis and our contributions to the research of analytics meta learning, including procedural knowledge discovery, analytics space exploration, and applications in building complex information systems such as information retrieval system, question answering system and decision support systems. Then, we discuss the future directions for each aspect.

10.1 Contributions

This thesis studies the problem of analytics meta learning, which aims to assist, if not replace, human in the design, planning, and evaluation during the development of intelligent information systems for analytics tasks, to guarantee continuous improvement towards to the task objective. We give formal definitions to the subjects we study in the thesis, which include analytics task, analytics procedure, information system (analytics engine), etc. Then, we formally define the analytics meta learning problem and present the solution framework, which consists of three major steps: analytics procedure definition, analysis component construction, and analytics space exploration.

10.1.1 Procedural Knowledge Discovery

As a subproblem of analytics meta learning, we introduce the problem of procedural knowledge discovery, which aims to automatically discover procedural knowledge – the knowledge exercised in the performance of some task, i.e. what actions should be performed (execution processes) and what factors should be considered to achieve some goal (decision processes) – and accordingly automatically construct analytics procedures for arbitrary analytics tasks. In this thesis, we explore a few human-generated data sources, such as on-line community how-to guides, search logs, etc., and study the canonicalization of representation to facilitate analysis component construction and execution. The experimental results show that the proposed methods taking the advantage of a semi-structured procedural knowledge base outperform the commercial search engines in actionable phrase suggestion task, and they also provide useful supplementary materials to existing procedural knowledge bases to aid automatic procedural knowledge base construction.
10.1.2  Analytics Space Exploration

We study the optimization problem in analytics meta learning – analytics space exploration, which operates directly on any arbitrary predefined and constructed analytics space. We model the evaluation and exploration processes as Markov Decision Processes (MDPs), and model a (budgeted) analytics space exploration problem as a (budgeted) policy optimization problem. We incorporate the state-of-the-art policy optimization algorithms proposed for reinforcement learning problems, including tabular value estimation, function approximation, policy gradient, etc., and also propose off-policy and model-based extensions, which provide the strategy candidate set containing 62 non-budgeted exploration strategies and 40 budgeted exploration strategies.

Then, we focus on exemplifying the proposed methodology to tackle analytics tasks and reporting the learning performance of the proposed analytics meta learning method as well as the performance of its resulting products, i.e. information systems, in terms of task-specific metrics and measurements. We apply these strategies to a simulated problem (of size $10^{10}$ or varied) and a number of real world problems, including a relevant content retrieval task (of size $9.3 \times 10^7$), a factoid question answering (of size $5.9 \times 10^9$), a pharmaceutical decision task (of size $5.2 \times 10^{29}$), and a product recommendation task (of size $1.7 \times 10^{35}$). We give suggestions on selecting an exploration strategy for a new analytics problem, depending on a number of decision factors, e.g. whether we are on a tight budget, whether the analytics/configuration space has inter-phase component dependency, whether there exists a phase that has extremely costly components, whether low-level analysis component learning is involved, etc.

10.1.3  Framework Implementation & Applications

We first present the principles of design and implementation of analytics meta learning software framework, and accordingly develop an open source toolkit that supports defining and specifying analysis components and analytics procedures (UIMA-ECD), and further describe a framework extension (AML) that implements various policy optimization algorithms to conduct analytics space exploration.

We also develop an open source biomedical question answering system, OAQA BioASQ, using the analytics meta learning methodology, which won the official BioASQ 2015 and 2016 challenges two year in a row, and is nominated as an official baseline system in the upcoming challenges. Nevertheless, when we leverage the proposed exploration strategies to boost the system performance using the same development set, the results show that more than half of the linear MC control strategies using a two-step option selection method have identified an optimal system with even better performance than the winning system. Next, we consider two complex analytics task scenarios that require human decision making – pharmaceutical decision task and product recommendation task, where we demonstrate construction and optimization of both analytics procedure and analysis components. These tasks expect intelligent information systems that are able to not only quickly find existing known answers but also precisely discover the unknown facts. The optimal systems can achieve an accuracy of above 0.75 and 0.93 respectively, which show that the systems are able to assist scientific research and everyday life activities.
10.2 Future Directions

There are many future directions of this thesis. In this section, we exemplify interesting research topics along each direction.

10.2.1 Procedural Knowledge Discovery

Procedural (or actionable) knowledge discovery is an important research topic in both information retrieval and natural language processing, and has been gaining much attention recently. For example, the Text REtrieval Conference (TREC) has started a new track named Tasks Track\[224\] since 2015, whose goal is to evaluate system’s understanding of tasks users aim to achieve and evaluate relevance of retrieved documents with respect to underlying tasks in query, where a subtask of the Tasks Track is subtask suggestion. Another similar attempt is the NTCIR Actionable Knowledge Graph (AKG) Task, a pilot task starting in 2017, whose goal is to generate knowledge graphs that are optimized for facilitating users’ actions (buying, booking, downloading, comparing, creating, etc.), where it defines two subtasks: Action Mining (AM) subtask and Actionable Knowledge Graph Generation (AKGG) subtask. In order to accomplish these tasks, in addition to the existing sources, we may consider to explore a wider range of human-generated data sources, such as on-line community question answering sites (e.g. Yahoo Answers\[2\]), bibliographic databases (e.g. MEDLINE), etc., and study the canonicalization of representation to facilitate generalization of procedural knowledge. There, we can leverage both the structured information (e.g. section information) and the textual information (e.g. discourse relation), similar to the proposed algorithm for extracting from semi-structured procedural knowledge bases.

10.2.2 Analytics/Configuration Space Exploration

Analytics space exploration is similar to the multi-armed bandit problem and reinforcement learning problem, but poses additional challenges that do not exist in these traditional problems, as compared in Table 5.1. Interesting research focuses include MDP modeling methods and policy optimization algorithms. We list a number of example topics.

- An extension of the CSE-MDP model used in this thesis is to consider continuous MDP, which can incorporate real-valued parameters directly without discretization and leverage a liner MC control or other strategies that can take a feature vector as state and state-action representation.
- Furthermore, one can also study how to represent the input and the intermediate object as feature vectors, and eventually automatically learn the representation for these objects.
- One may also be interested in studying the theoretical properties of the two-step option selection methods used in the proposed (budgeted) exploration strategies.

[http://answers.yahoo.com](http://answers.yahoo.com)
• We can also modify the reward function that can also generates negative signals for policy-based strategies.
• We may study a variant of the two-step option selection method that always selects the highest-reward component in the greedy phases.
• In this thesis, we focus mostly on constant learning rate and constant exploration rate (or constant exploration decay rate). As a next step, we may consider to apply an adaptive learning rate, such as AdaGrad [52], Adam [98], etc., and further study an adaptive exploration rate following a similar idea.
• A future work may also involve comparing the performance of fixing the option selection method with that of switching between methods, e.g. $\epsilon$-first [196].
• We have investigated other possible cost measurements but didn’t use any cost measurement other than time. For example, the cost of using Amazon Web Services. However, even the advanced CloudWatch API cannot report cost/charge in real time. To incorporate this metric, the analytics meta learning framework should support asynchronous model update, i.e. when it finishes executing an episode/component at time $t$, it immediately starts executing the next episode/component rather than waits for the performance evaluation, and the episode/component completed at time $t$ is used to update the models only when the corresponding metric is reported at a later time $t'$.

We may also consider advanced topics, such as multi-analytics meta learning, human-in-the-loop system optimization, which also require knowledge from other research areas, e.g. transfer learning or multi-task learning, human computer interaction.

Multi-analytics meta learning

The objective of (single) analytics meta learning is to treat each option as a black box and estimate the cost and benefit of each option, rather than understand why it works or doesn’t from the first principle. A similar problem, namely multi-analytics meta learning or transfer learning across analytics tasks, which we do not study in this thesis, attempts to understand the relation between the option performance and the task, and make prediction whether an option has a universally high performance or it only works with certain types of tasks. We leave multi-analytics meta learning for future work.

Nontrivial weight initialization that reflects the performance of each option is the key step in transfer learning across tasks. The current framework can be extended to enable transfer learning across tasks across users, with a simple introduction of an additional persistence provider – component/option performance persistence provider (PPP). If the PPP is shared or centralized, then it can easily provide a transparent transfer learning experience for other users who have access to the same PPP and explicitly specify to do so. It consists a client (user) side specification and a server side setup.

The simplest PPP server/database should provide a mapping from a STATE to a mapping from each compatible component OPTION to its PERFORMANCE, which can be implemented by a high-

[https://aws.amazon.com/cloudwatch/](https://aws.amazon.com/cloudwatch/)
Table 10.1: Performance persistence database

<table>
<thead>
<tr>
<th>STATE</th>
<th>OPTION</th>
<th>PERFORMANCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>question-parse</td>
<td>QuestionParser[inherit:bioqa.question.parse.clearnlp-bioinformatics#parser-providers.parser.clearnlp-bioinformatics]</td>
<td>0.0006</td>
</tr>
<tr>
<td>question-parse</td>
<td>QuestionParser[inherit:bioqa.question.parse.clearnlp-medical#parser-provider:inherit:bioqa.providers.parser.clearnlp-medical]</td>
<td>-0.0061</td>
</tr>
<tr>
<td>question-conc</td>
<td>QuestionConceptRecognizer[inherit:example.bioasq.question.concept.metamap-cached#concept-provider:inherit:example.bioasq.providers.kb.metamap-cached]</td>
<td>0.0084</td>
</tr>
<tr>
<td>ept-metamap</td>
<td>NoOpAnnotator[inherit:base.noop]</td>
<td>-0.0092</td>
</tr>
<tr>
<td>question-conc</td>
<td>QuestionConceptRecognizer[inherit:example.bioasq.question.concept.tmtool-cached#concept-provider:inherit:example.bioasq.providers.kb.tmtool-cached]</td>
<td>0.0280</td>
</tr>
<tr>
<td>ept-tmtool</td>
<td>NoOpAnnotator[inherit:base.noop]</td>
<td>-0.0198</td>
</tr>
</tbody>
</table>

performance distributed key-value store, such as Redis\footnote{https://redis.io/} or a relational database. At the end of each configuration space exploration process, the execution driver (e.g. UIMA-ECD driver in our implementation) can automatically insert the learned performance data as a new entry into the database. When the next (possibly remote) user defines another configuration space and then specifies the same PPP, the framework can pull the previously learned performance automatically from the PPP to initialize the policies as long as it encounters the same state and option. After the new exploration, the framework inserts the learned performance results back to the PPP and merges with the previous entries.

For example, if we use a policy-gradient method to optimize the biomedical QA system, and use the simplest current phase (CP) for state representation, we may see the data entries in the database as shown in Table 10.1 where the performance is the weight of the option in the softmax policy. As we have discussed in Section 5.4.3 policy-based methods allow easy transferring learned knowledge across tasks.

The client side configuration should be specified at each phase, mainly including whether the user would like to push and/or pull the performance results to/from the PPP. For example, the user may only want to pull the results from a Redis PPP but refuse to share the performance results with other users, so he/she may only define the pull-broker as a broker URL to the Redis URL but keeps the push-broker undefined.

\footnote{https://redis.io/}
# partial configuration space (ECD) definition
- inherit: aml.phase
  name: example-phase
  options: |
  ...
  performance-persistence-provider: |
  inherit: example-ppp

# example-ppp.yaml
inherit: default-ppp
pull-broker: redis://localhost:6379/aml-ppp
push-broker: null

A more comprehensive performance persistence database should also define columns (or keys) that represent experiment meta information (as part of the experiment persistence provider as introduced in Section 6.2), such as USER_ID, DATE_TIME, TASK_NAME, DATASET, etc.

- USER_ID can be used to better control credential and privacy.
- DATE_TIME can be used to filter outdated performance results or the most recent (possibly unstable) results.
- TASK_NAME and DATASET can be specified so that only the performance results belonging to the same task and/or data set are pulled for transferring knowledge, in case the state/option has different performance on different tasks or data sets.

### Human-in-the-loop system optimization

The problem formulation described in Chapter 2 and the exploration strategies presented in Chapter 5 make no assumption that the analytics space should only include system components and no human components.

In fact, when each analysis step in an analytics procedure can be implemented by either an automatic component, an interactive component, or a manual component, and an information system can thus consist of all different types of components. As a result, an analytics space for a certain analytics task can also be a mixture of automatic, interactive, and manual components, even within the same process step. We take answer generation step (step \( i \)) as an example. We may instantiate five components for this step, each of which considers a different component type. The first component \( c_{i,1} \) is an automatic algorithm, the second component \( c_{i,2} \) can be a very experienced human annotator, the third component \( c_{i,3} \) is a novice annotator, the fourth component \( c_{i,4} \) is a random crowd worker annotator, e.g. Amazon Mechanic Turk, and the fifth component \( c_{i,5} \) can be a do-it-yourself annotator, i.e. it delegates the processing task to the end-user (customer) rather than any person in the development team as \( c_{i,2}, c_{i,3}, c_{i,4} \). All human annotators \( (c_{i,2}, \ldots, c_{i,5}) \) may have different interfaces and thus require different implementations. In the following, we show a simple command line interface using the UIMA framework, which uses the synchronous interaction model.

```java
public class ExampleHumanAnnotator extends JCasAnnotator_ImplBase {
```
@Override
public void process(JCas jcas)
throws AnalysisEngineProcessException {
    String text = jcas.getDocumentText();
    System.out.printf("Identify the answer from \"%s\"\n", text);
    Scanner scanner = new Scanner(System.in);
    String line;
    while ((line = scanner.nextLine()) != null) {
        TypeFactory.createAnswer(jcas, line).addToIndexes();
    }
}

An analytics space exploration strategy is unaware of what is done underneath each option, whether it is automatic, interactive, or manual, as long as they satisfy the same input/output capabilities, and treats each option \( c_{ij} \) as a black box. We note that the “value” in the context of reinforcement learning is always a single scalar number that consider all relevant factors for decision making. In our case studies, we simply use the evaluation metrics, such as MAP, MRR, accuracy, etc., to represent the return (the actual value collected in an episode). In the human-in-the-loop case, component \( c_{i,1} \) may induce a licensing fee for commercial use and have a moderate task performance, whereas component \( c_{i,2} \) may have the highest task performance but a much higher salary cost, and component \( c_{i,4} \) may have a moderate task performance either but possible lowest cost. However, management accounting can conduct cost-benefit analysis to estimate the net profit from all these factors, which can be used as the return for our purpose. Once the single value is estimated for each single task input, the objective of the analytics meta learning framework is to make estimation of each option by experiencing different combinations of these components. For example, it uses component \( c_{i,1} \) in the first episode, then explores component \( c_{i,4} \) in the second episode, and so on.

The simulated configuration space used in the simulated experiments described in Sections 5.5 and 5.7 can be flexibly interpreted as a pool of automatic components, human components, or mixture, since the task performance of each component is a randomly drawn from a parameterized beta distribution, which can represent the overall gain/loss accounting all the relevant factors. In fact, to leverage human intelligent to achieve certain analytics task, especially (realtime) crowd-powered systems, is an emerging research area in human-computer interaction (HCI). An interesting HCI-focused next step would be to conduct a case study that involves all different types of human components and compare its performance with that of a completely automatic system.

Automatically change the procedure

In the Watson’s multi-phase answer ranking, the earlier phases use faster and more universal feature extractors on a larger number of candidates, where the intuition is that one only wants to run the more expensive feature extractors when he/she has to. We can imagine a system that can tell us how many such phases he/she should specify and what feature extractors should be plugged into each phase, by starting with a single answer ranking phase and then growing the pipeline.
To further support this, we need to extend the system in three different ways.

**Automatic procedure construction** First, we want to avoid manually creating a descriptor for each analytics procedure that has exactly $N = 1, \ldots, \text{MAX\_VALUE}$ answer ranking phases. Also, we can hardly find any human-generated source that describes an analytics procedure for each $N$. Therefore, we should leverage an automatic analytics procedure construction process, similar to *automatic service composition*, which can utilize the input/output (syntactic and/or semantic) capabilities to construct a functioning information workflow. For example, an answer (re)ranking phase/component should take a ranked list of answers and return another ranked list of answers, which allows an analytics procedure to include zero to infinite number of answer reranking phases.

**Feedback loop from exploration to definition** In this thesis, we define the analytics procedure definition and analytics space exploration as two separate steps, where in the first step we define either manually or automatically a finite number of static analytics procedures, and then in the second step we apply an exploration strategy to select the best policy without changing or extending predefined procedures. In the case studies, we demonstrate a single development iteration using analytics meta learning. We may further extend the work to have a feedback loop from the analytics space exploration stage back to the analytics procedure definition stage, to request a change or variation in the procedure definition, as shown by the dashed lines show in Figure 2.1.

**Awareness of numeric values** The current system can only deal with discrete values. However, even nonconvex numeric values can still help the exploration algorithm to identify the next potential high-value option. For example, if the system has tried a pipeline with 3, 4, and 5 answer ranking phases, and discovers that the best performance is obtained when there are 5 answer ranking phases, and the second best performance is obtained when there are 4 answer ranking phases, then intuitively, the next procedure to explore should have 6 answer ranking phases.

### 10.2.3 Question Answering & Decision Support

Question answering and decision support systems are both examples of complex intelligent information systems, which have become more popular than ever since the success of IBM’s Watson system [58]. Our hypothesis is that the best performance can only be achieved through careful design of a flexible and extensible architecture, coupled with continuous, incremental experimentation and optimization over various combinations of existing state-of-the-art components, rather than relying on a single “magic” component or a single end-to-end model. In the case studies, we put most of our effort in identifying and integrating the existing off-the-shelf NLP, retrieval and machine learning tools and algorithms (in the analysis component construction phase), and we see that the experimental results prove our hypothesis and show that the proposed analytics meta learning methodology and framework can help achieve this goal. In the future, we may first alter the analytics procedure definition. For example, we may adopt a Question Answering over Linked Data (QALD) style architecture for biomedical question answering task, which centers around an
existing structured knowledge base (e.g. LinkedLifeData[^5]) and also integrates the components that converts the question into a SPARQL query. We may also consider a vector representation (i.e. embedding) for the the AbstractQuery, and retrieval and rank the relevant contents using this vector representation. Even though we deal with the same analytics tasks, such new architecture and data structure designs require different analytics procedure definitions, where we also need to construct new analytics spaces for analytics space exploration. There are also analytics tasks in the BioASQ challenge that remain to be solved or improved, such as summarization task or yes/no question answer task.

[^5]: http://linkedlifedata.com/
### Appendix A

**SQLite Schemata & YAML Descriptors**

#### A.1 SQLite Schema for Experiment & Evaluation Persistence Providers

Table A.1: SQLite schema for experiment & evaluation persistence providers

<table>
<thead>
<tr>
<th>Table</th>
<th>Columns</th>
<th>Type</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>experiment_meta</td>
<td>experiment</td>
<td>CHAR(36)</td>
<td>NOT NULL</td>
</tr>
<tr>
<td></td>
<td>phase</td>
<td>INTEGER</td>
<td>NOT NULL</td>
</tr>
<tr>
<td></td>
<td>stage</td>
<td>INTEGER</td>
<td>NOT NULL</td>
</tr>
<tr>
<td></td>
<td>size</td>
<td>INTEGER</td>
<td>NOT NULL</td>
</tr>
<tr>
<td>experiment_log</td>
<td>id</td>
<td>BIGINT(20)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>experiment</td>
<td>CHAR(36)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>traceId</td>
<td>CHAR(64)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>type</td>
<td>VARCHAR(45)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>message</td>
<td>TEXT</td>
<td></td>
</tr>
<tr>
<td>experiment</td>
<td>id</td>
<td>INT</td>
<td>PRIMARY KEY</td>
</tr>
<tr>
<td></td>
<td>dataset</td>
<td>VARCHAR(255)</td>
<td>DEFAULT ’default’</td>
</tr>
<tr>
<td></td>
<td>date</td>
<td>TIMESTAMP</td>
<td>DEFAULT CURRENT_TIMESTAMP</td>
</tr>
<tr>
<td></td>
<td>sequenceIds</td>
<td>VARCHAR(2047)</td>
<td>DEFAULT ’0’</td>
</tr>
<tr>
<td></td>
<td>sequenceStart</td>
<td>INT</td>
<td>DEFAULT 0</td>
</tr>
<tr>
<td></td>
<td>sequenceEnd</td>
<td>INT</td>
<td>DEFAULT 0</td>
</tr>
<tr>
<td></td>
<td>name</td>
<td>VARCHAR(100)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>author</td>
<td>VARCHAR(45)</td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Table A.1 – continued from previous page</th>
</tr>
</thead>
<tbody>
<tr>
<td>resource</td>
</tr>
<tr>
<td>configuration</td>
</tr>
<tr>
<td>size</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cas_str</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
</tr>
<tr>
<td>experiment</td>
</tr>
<tr>
<td>dataset</td>
</tr>
<tr>
<td>date</td>
</tr>
<tr>
<td>hostname</td>
</tr>
<tr>
<td>optionId</td>
</tr>
<tr>
<td>phase</td>
</tr>
<tr>
<td>phaseUid</td>
</tr>
<tr>
<td>predecessorId</td>
</tr>
<tr>
<td>question</td>
</tr>
<tr>
<td>sequenceId</td>
</tr>
<tr>
<td>startTime</td>
</tr>
<tr>
<td>trace</td>
</tr>
<tr>
<td>answer</td>
</tr>
<tr>
<td>answerPattern</td>
</tr>
<tr>
<td>endTime</td>
</tr>
<tr>
<td>isCorrect</td>
</tr>
<tr>
<td>status</td>
</tr>
<tr>
<td>xcas</td>
</tr>
<tr>
<td>traceHash</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>partial_measurements</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
</tr>
<tr>
<td>experimentId</td>
</tr>
<tr>
<td>traceId</td>
</tr>
<tr>
<td>calculator</td>
</tr>
<tr>
<td>evaluatee</td>
</tr>
<tr>
<td>measure</td>
</tr>
<tr>
<td>value</td>
</tr>
<tr>
<td>sequenceId</td>
</tr>
<tr>
<td>stage</td>
</tr>
<tr>
<td>traceHash</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>accumulated_measurements</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
</tr>
<tr>
<td>experimentId</td>
</tr>
</tbody>
</table>

Continued on next page
### Table A.1 – continued from previous page

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>traceId</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>calculator</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>evaluatee</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>measure</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>value</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>stage</td>
<td>INT</td>
</tr>
<tr>
<td>traceHash</td>
<td>CHAR</td>
</tr>
</tbody>
</table>

#### A.2 YAML Descriptors Used in the Biomedical Question Answering Case Study

Listing A.1: ECD main descriptor of relevant content retrieval pipeline used for Batch 5 of BioASQ 4B Phase B

```yaml
# execute
#   mvn exec:exec -Dconfig=bioasq.phase-a-test
# to test the pipeline

configuration:
  name: phase-a-test
  author: ziy

persistence-provider: | inherit: baseqa.persistence.local-sqlite-persistence-provider
collection-reader:  | inherit: baseqa.collection.json.json-collection-reader
dataset: BIOASQ-QA
file:    | - input/4b-5-a.json
persistence-provider: | inherit: baseqa.persistence.local-sqlite-persistence-provider
pipeline: |
  - inherit: ecd.phase
    name: question-parse
    options: |
      - inherit: bioqa.question.parse.clearnlp-bioinformatics
  - inherit: ecd.phase
    name: question-concept-metamap
    options: |
      - inherit: bioqa.question.concept.metamap-cached
```
- inherit: ecd.phase
  name: question-concept-tmtool
  options: |
    - inherit: bioqa.question.concept.tmtool-cached

- inherit: ecd.phase
  name: question-concept-lingpipe-genia
  options: |
    - inherit: bioqa.question.concept.lingpipe-genia

- inherit: ecd.phase
  name: concept-search-uts
  options: |
    - inherit: bioqa.evidence.concept.search-uts-cached

- inherit: ecd.phase
  name: concept-merge
  options: |
    - inherit: baseqa.evidence.concept.merge

- inherit: ecd.phase
  name: abstract-query-primary
  options: |
    - inherit: baseqa.abstract_query.token-concept

# concept
- inherit: ecd.phase
  name: concept-retrieval
  options: |
    - inherit: bioqa.concept.retrieval.lucene-bioconcept

- inherit: ecd.phase
  name: concept-rerank
  options: |
    - inherit: bioqa.concept.rerank.predict-liblinear

# document
- inherit: ecd.phase
  name: document-retrieval
  options: |
    - inherit: bioqa.document.retrieval.lucene-medline

- inherit: ecd.phase
  name: document-rerank
options: |
    - inherit: bioqa.document.rerank.predict-liblinear

# snippet
- inherit: ecd.phase
  name: passage-retrieval
  options: |
    - inherit: bioasq.passage.retrieval.document-to-passage

- inherit: ecd.phase
  name: passage-rerank
  options: |
    - inherit: bioqa.passage.rerank.predict-liblinear
    - inherit: base.noop

post-process:
  # submission
  - inherit: bioasq.collection.json.json-cas-consumer

Listing A.2: ECD main descriptor of factoid and list QA pipeline used for Batch 5 of BioASQ 3B Phase B

# execute
# mvn exec:exec -Dconfig=bioasq.phase-b-test-factoid-list
# to test the pipeline

configuration:
  name: phase-b-test-factoid-list-4b
  author: ziy

persistence-provider:
  inherit: baseqa.persistence.local-sqlite-persistence-provider

collection-reader:
  inherit: baseqa.collection.json.json-collection-reader
dataset: BIOASQ-QA
  file:
    - /input/3b-5-b.json
type: [factoid, list, yesno, summary]
persistence-provider: |
  inherit: baseqa.persistence.local-sqlite-persistence-provider

pipeline:
  - inherit: ecd.phase
    options: |
      - inherit: bioqa.quesanal.parse-clearnlp-bioinformatics
- inherit: ecd.phase
  options: |
    - inherit: bioqa.quesanal.concept-metamap

- inherit: ecd.phase
  options: |
    - inherit: bioqa.quesanal.concept-lingpipe-genia

- inherit: ecd.phase
  options: |
    - inherit: baseqa.quesanal.concept-opennlp-np

- inherit: ecd.phase
  options: |
    - inherit: baseqa.quesanal.concept-opennlp-nppnpn

- inherit: ecd.phase
  options: |
    - inherit: baseqa.quesanal.lexical-answer-type

- inherit: ecd.phase
  options: |
    - inherit: bioqa.quesanal.at.predict-liblinear

- inherit: ecd.phase
  options: |
    - inherit: baseqa.retrieval.passage-to-view

- inherit: ecd.phase
  options: |
    - inherit: bioqa.retrieval.passage-parse-clearnlp-bioinformatics

- inherit: ecd.phase
  options: |
    - inherit: bioqa.retrieval.passage-concept-metamap

- inherit: ecd.phase
  options: |
    - inherit: bioqa.retrieval.passage-concept-lingpipe-genia

- inherit: ecd.phase
  options: |
    - inherit: baseqa.retrieval.passage-concept-opennlp-np
- inherit: ecd.phase
  options: |
    - inherit: baseqa.retrieval.passage-concept-opennlp-nppnp

- inherit: ecd.phase
  options: |
    - inherit: bioqa.retrieval.concept-search-uts

- inherit: ecd.phase
  options: |
    - inherit: baseqa.retrieval.concept-merge

- inherit: ecd.phase
  options: |
    - inherit: bioqa.answer.generate

- inherit: ecd.phase
  options: |
    - inherit: baseqa.answer.modify

- inherit: ecd.phase
  options: |
    - inherit: bioqa.answer.score-predict-liblinear

- inherit: ecd.phase
  options: |
    - inherit: baseqa.answer.pruner

post-process:
  # submission
  - inherit: bioasq.collection.json.json-cas-consumer

Listing A.3: ECD main descriptor of factoid and list QA pipeline used for Batch 5 of BioASQ 4B Phase B

# execute
# mvn exec:exec -Dconfig=bioasq.phase-b-test-factoid-list
# to test the pipeline

configuration:
  name: phase-b-test-factoid-list-4b
  author: ziy

persistence-provider:
  inherit: baseqa.persistence.local-sqlite-persistence-provider
collection-reader:
  inherit: baseqa.collection.json.json-collection-reader
dataset: BIOASQ-QA
file:
  - input/4b-5-b.json
type: [factoid, list]
persistence-provider:
  inherit: baseqa.persistence.local-sqlite-persistence-provider

pipeline:
  - inherit: ecd.phase
    name: question-parse
    options: |
      - inherit: bioqa.question.parse.clearnlp-bioinformatics

  - inherit: ecd.phase
    name: question-concept-metamap
    options: |
      - inherit: bioqa.question.concept.metamap-cached

  - inherit: ecd.phase
    name: question-concept-tmtool
    options: |
      - inherit: bioqa.question.concept.tmtool-cached

  - inherit: ecd.phase
    name: question-concept-lingpipe-genia
    options: |
      - inherit: bioqa.question.concept.lingpipe-genia

  - inherit: ecd.phase
    name: question-lexical-answer-type
    options: |
      - inherit: baseqa.question.lexical-answer-type

  - inherit: ecd.phase
    name: passage-to-view
    options: |
      - inherit: baseqa.evidence.passage-to-view

  - inherit: ecd.phase
    name: evidence-parse
    options: |
      - inherit: bioqa.evidence.parse.clearnlp-bioinformatics
- inherit: ecd.phase
  name: evidence-concept-metamap
  options: |
    - inherit: bioqa.evidence.concept.metamap-cached

- inherit: ecd.phase
  name: evidence-concept-tmtool
  options: |
    - inherit: bioqa.evidence.concept.tmtool-cached

- inherit: ecd.phase
  name: evidence-concept-lingpipe-genia
  options: |
    - inherit: bioqa.evidence.concept.lingpipe-genia

- inherit: ecd.phase
  name: evidence-concept-frequent-phrase
  options: |
    - inherit: baseqa.evidence.concept.frequent-phrase

- inherit: ecd.phase
  name: concept-search-uts
  options: |
    - inherit: bioqa.evidence.concept.search-uts-cached

- inherit: ecd.phase
  name: concept-merge
  options: |
    - inherit: baseqa.evidence.concept.merge

- inherit: ecd.phase
  name: answer-type
  options: |
    - inherit: bioqa.answer_type.predict-liblinear-null

- inherit: ecd.phase
  name: answer-generate
  options: |
    - inherit: bioqa.answer.generate.generate

- inherit: ecd.phase
  name: answer-modify
  options: |
    - inherit: baseqa.answer.modify.modify
- inherit: ecd.phase
  name: answer-score
  options: |
    - inherit: bioqa.answer.score.predict-liblinear

- inherit: ecd.phase
  name: answer-collective-score
  options: |
    - inherit: bioqa.answer.collective_score.predict-liblinear
    - inherit: base.noop

- inherit: ecd.phase
  name: answer-prune
  options: |
    - inherit: baseqa.answer.modify.pruner

post-process:
  # submission
  - inherit: bioasq.collection.json.json-cas-consumer

Listing A.4: ECD main descriptor of yes/no QA pipeline used for Batch 5 of BioASQ 4B Phase B

# execute
# mvn exec:exec -Dconfig=bioasq.phase-b-test-yesno
# to test the pipeline

configuration:
  name: phase-b-test-yesno
  author: ziy

persistence-provider:
  inherit: baseqa.persistence.local-sqlite-persistence-provider

collection-reader:
  inherit: baseqa.collection.json.json-collection-reader
dataset: BIOASQ-QA
  file:
    - input/4b-5-b.json
type: [yesno]
persistence-provider: |
  inherit: baseqa.persistence.local-sqlite-persistence-provider

pipeline:
  - inherit: ecd.phase
    name: question-parse
    options: |
- inherit: bioqa.question.parse.clearnlp-bioinformatics

- inherit: ecd.phase
  name: question-concept-metamap
  options: |
    - inherit: bioqa.question.concept.metamap-cached

- inherit: ecd.phase
  name: question-concept-tmtool
  options: |
    - inherit: bioqa.question.concept.tmtool-cached

- inherit: ecd.phase
  name: question-concept-lingpipe-genia
  options: |
    - inherit: bioqa.question.concept.lingpipe-genia

- inherit: ecd.phase
  name: passage-to-view
  options: |
    - inherit: baseqa.evidence.passage-to-view

- inherit: ecd.phase
  name: evidence-parse
  options: |
    - inherit: bioqa.evidence.parse.clearnlp-bioinformatics

- inherit: ecd.phase
  name: evidence-concept-metamap
  options: |
    - inherit: bioqa.evidence.concept.metamap-cached

- inherit: ecd.phase
  name: evidence-concept-tmtool
  options: |
    - inherit: bioqa.evidence.concept.tmtool-cached

- inherit: ecd.phase
  name: evidence-concept-lingpipe-genia
  options: |
    - inherit: bioqa.evidence.concept.lingpipe-genia

- inherit: ecd.phase
  name: evidence-concept-frequent-phrase
  options: |
- inherit: baseqa.evidence.concept.frequent-phrase

- inherit: ecd.phase
  name: concept-search-uts
  options: |
    - inherit: bioqa.evidence.concept.search-uts-cached

- inherit: ecd.phase
  name: concept-merge
  options: |
    - inherit: baseqa.evidence.concept.merge

- inherit: ecd.phase
  name: answer-yesno
  options: |
    - inherit: bioqa.answer.yesno.predict-weka-other
    - inherit: baseqa.answer.yesno.all-yes

post-process:
  # submission
  - inherit: bioasq.collection.json.json-cas-consumer

Listing A.5: ECD component descriptor of bioqa.answer.generate used for Batch 5 of BioASQ 3B Phase B

```
class: edu.cmu.lti.oaqa.baseqa.answer.CavGenerationManager

generators: |
  - inherit: baseqa.answer.generators.choice
  - inherit: baseqa.answer.generators.quantity
  - inherit: baseqa.answer.generators.concept
  - inherit: baseqa.answer.generators.cav-covering-concept
  - inherit: baseqa.answer.generators.yesno
```

Listing A.6: ECD component descriptor of bioqa.answer.score.predict-liblinear used for Batch 5 of BioASQ 3B Phase B

```
inherit: baseqa.answer.score-predict

classifier: 'inherit: bioqa.answer.score-classifier-liblinear'
scorers: |
  - inherit: baseqa.answer.scorers.type-coercion
  - inherit: baseqa.answer.scorers.cao-count
  - inherit: baseqa.answer.scorers.name-count
  - inherit: baseqa.answer.scorers.avg-covered-token-count
  - inherit: baseqa.answer.scorers.stopword-count
  - inherit: baseqa.answer.scorers.token-overlap-count
```
Listing A.7: ECD component descriptor of `bioqa.answer.collective_score.predict-liblinear` used for Batch 5 of BioASQ 4B Phase B

```yaml
- inherit: baseqa.learning_base.classifier-predict

candidate-provider: 'inherit: baseqa.answer.score.candidate-provider'
scorers: |
  - inherit: baseqa.answer.collective_score.scorers.original
  - inherit: baseqa.answer.collective_score.scorers.distance
  - inherit: baseqa.answer.collective_score.scorers.edit-distance
  - inherit: baseqa.answer.collective_score.scorers.type-coercion
  - inherit: baseqa.answer.collective_score.scorers.shape-distance

classifier: 'inherit: bioqa.answer.collective_score.liblinear-classifier'
feature-file: result/answer-collective-score-predict-liblinear.tsv
```

Listing A.8: ECD component descriptor of `bioqa.answer.yesno.predict` used for Batch 5 of BioASQ 4B Phase B

```yaml
- inherit: baseqa.answer.yesno.predict

scorers: |
  - inherit: baseqa.answer.yesno.scorers.concept-overlap
  - inherit: bioqa.answer.yesno.scorers.token-overlap
  - inherit: baseqa.answer.yesno.scorers.expected-answer-overlap
  - inherit: baseqa.answer.yesno.scorers.sentiment
  - inherit: baseqa.answer.yesno.scorers.negation
  - inherit: bioqa.answer.yesno.scorers.question-inversion
```
Bibliography


[27] Christopher Brewster, Kieron O’Hara, Steve Fuller, Yorick Wilks, Enrico Franconi, Mark A Musen, Jeremy Ellman, and Simon Buckingham Shum. Knowledge representation with


271


[50] Wenkui Ding, Tao Qin, Xu-Dong Zhang, and Tie-Yan Liu. Multi-armed bandit with budget constraint and variable costs. In AAAI, 2013. 5.2 5.1.1 5.6.1


[55] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblin-


[111] Xin Li and Dan Roth. Learning question classifiers. In *ACL’2002*, pages 1–7, 2002. 7.4.4


[169] Thomas L Saaty. What is the analytic hierarchy process? Springer, 1988. 2.1.2 2.2
[176] Luo Si and Jamie Callan. Modeling search engine effectiveness for federated search. In SIGIR’2005, pages 83–90, 2005. 7.4.2
[177] Luo Si, Jie Lu, and Jamie Callan. Combining multiple resources, evidences and criteria for genomic information retrieval. In Proceedings of TREC’06, 2006. 4.2 7.1
[185] Nicola Stokes, Yi Li, Lawrence Cavedon, and Justin Zobel. Exploring criteria for successful


[194] the Apache UIMA Development Community. Uima tutorial and developers’ guides, August 2015. 2.1


[196] Long Tran-Thanh, Archie Chapman, Jose Enrique Munoz De Cote Flores Luna, Alex Rogers, and Nicholas R Jennings. Epsilon–first policies for budget–limited multi-armed bandits. 2010. 5.1.1, 5.7.3, 10.2.2


[199] George Tsatsaronis, Georgios Balikas, Prodromos Malakasiotis, Ioannis Partalas, Matthias Zschunke, Michael R Alvers, Dirk Weissenborn, Anastasia Krithara, Sergios Petridis, Dim-

282


[217] Di Xu and Jamie Callan. Towards a simple and efficient web search framework, 2014. 7.4.2


[221] Zi Yang, Ying Li, James Cai, and Eric Nyberg. Quads: question answering for decision support. In *Proceedings of the 37th international ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR’2014)*, pages 375–384. ACM, 2014. 1.3 1.3 1.4 5.3 5.5.1 7 8.3.2


[225] Xiao Yu, Hao Ma, Bo-June (Paul) Hsu, and Jiawei Han. On building entity recommender systems using user click log and freebase knowledge. In *Proceedings of WSDM’2014*, pages 263–272, 2014. 3.3


284