Towards Planning and Execution for Information Retrieval

Laurie S. Hiyakumoto and **Manuela M. Veloso**
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA, USA 15213
{hyaku, mmv}@cs.cmu.edu

Abstract

The problem of information gathering has received considerable attention from the planning community in recent years. However, research in this area has generally assumed a user's information goal is perfectly represented by the query, and typically adopts a relational database model for representing query operations and information sources. In this paper, we present a planning and execution framework intended to address the more general information retrieval (IR) problem in which queries are only approximate representations of the user's true information goals, and complete models of the content of information sources are not available. Our approach reformulates traditional IR techniques such as synonym expansion and relevance-feedback as domain operators, explicitly modeling the uncertainty of action outcomes and of satisfying the user's information needs, and taking into account resource constraints such as time. A forward-chaining planning and execution algorithm guides action selection, execution and replanning decisions using a user-defined utility function. We have implemented this approach in the INSPIRE (INtegrated System for Planning and Information REtrieval) architecture.

Introduction

Traditional ad-hoc approaches to information retrieval (IR) are becoming increasingly inadequate for today's large, dynamic, heterogeneous document collections, the most obvious example of which is the World Wide Web. Current IR systems place most of the burden on the users, relying on them to identify sources likely to contain relevant information, compose an appropriate query, and sift through retrieved documents to extract relevant information. Clearly, as document collections continue to grow, it will become impractical for users to perform these tasks for all but the simplest requests. Even today there are more sources than a person could possibly access in a reasonable amount of time, many of which contain redundant, irrelevant, outdated, or even erroneous information.

Ideally, a user should be able to state a high-level request to an IR system and it would take care of the rest. Although we are far from achieving this goal today, one promising approach is to use planning to automate more of the IR process. In addition to the obvious benefit of reducing user effort, a planning approach has two important advantages over traditional approaches. First, it provides a structured framework for learning which IR tools are best-suited for different types of retrieval tasks. Second, it frees system developers to try radically different IR techniques without worrying about their comprehensibility to novice users. However, before a planning and execution system can successfully apply to IR, it must address several challenges presented by the task:

- **Incompletely-specified goals:** A query is often a poor approximation of the user's true information goals, and different information goals may be expressed by identical queries.
- **Partial and incremental goal satisfaction:** For many information goals, retrieval is not an all-or-nothing proposition. Partial satisfaction of a user's request is common and may be achieved *incrementally* via multiple iterations.
- **Incomplete knowledge of information source contents:** It is virtually impossible to determine the full extent of information available from many large heterogeneous collections (e.g. the web pages indexed by Google).
- **Uncertain action outcomes:** The performance of IR techniques varies from one use to the next. For example, synonym expansion will produce better results on some queries than others, and retrieval times will vary with network and machine loads.
- **Resource constraints:** Most users are not willing to wait more than a few seconds for a response unless the information is extremely valuable to them.
- **User interaction:** Success is uncertain until we actually execute a plan and receive feedback from the user. An implementation capable of interleaving execution with planning is essential, as is support for interaction with, and possibly intervention by the user.

This paper presents an integrated planning and execution framework that addresses these issues. Our domain operators are drawn from traditional IR techniques such as synonym expansion and relevance feedback. We define an abstract representation of the user's goals and resource constraints using a set of real-valued *metrics* and a utility-function specifying their relative importance. A forward-chaining planning and execution algorithm searches through

Copyright © 2002, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

```
:name ( expand-query-with-synonyms )
:parameters ((?q query) system-time-spent est-query-quality )
:preconditions (
     :symbolic ( (not (applied-op-expand-with-synonyms ?q)) )
     :metric   ( (< est-query-quality 0.8) ) )
:effects (
     0.70 (:symbolic ( (del (not (applied-op-expand-with-synonyms ?q)))
                       (add (applied-op-expand-with-synonyms ?q)))
          :metric   ( (est-query-quality += 0.3)
                      (system-time-spent += 0.4)))
     0.28 (:symbolic ( (del (not (applied-op-expand-with-synonyms ?q)))
                       (add (applied-op-expand-with-synonyms ?q)))
          :metric   ( (est-query-quality -= 0.2)
                      (system-time-spent += 0.4)))
     0.02 (:symbolic ( (del (not (applied-op-expand-with-synonyms ?q)))
                       (add (applied-op-expand-with-synonyms ?q)))
          :metric   ( (system-time-spent += 0.5))) )
:execute ( synonym-expansion ?q )
```

Figure 1: Operator specification for `expand-query-with-synonyms`.

the space of *belief states*, sets of possible states and their associated likelihood, using expected utility and belief state uncertainty to guide action selection, execution, and replanning decisions.

This work contributes to planning research in two ways. First, it identifies planning challenges posed by the general IR task and presents a model that addresses them. Second, it presents an approach to handling incompletely-specified goals using an integrated planning and execution process.

The remainder of the paper is organized as follows. First, we describe the domain and problem representations we use. Secondly, we present our planning and execution algorithm. We then briefly describe the INSPIRE architecture that implements the approach, and discuss ongoing work in learning parameters to estimate the efficacy of domain operators. We further discuss the relationship between our work and previous research in planning and execution. We conclude with a discussion of current and future research directions.

The IR Planning Domain

An IR planning problem consists of a domain definition and a problem statement. The domain defines the problem-independent knowledge of the IR task which is common to different retrieval requests. The problem statement defines a problem-specific search context, the user's information goals, and resource constraints. In this section, we describe our representational choices for each.

Domain

An IR domain consists of:

- A set of types defining classes of objects (e.g., a `QUERY`)
- A set of metrics declaring resources, costs (e.g., `(ELAPSED-TIME 5)`) and *quality estimates* that will be monitored within states
- A set of operators defining the available atomic actions

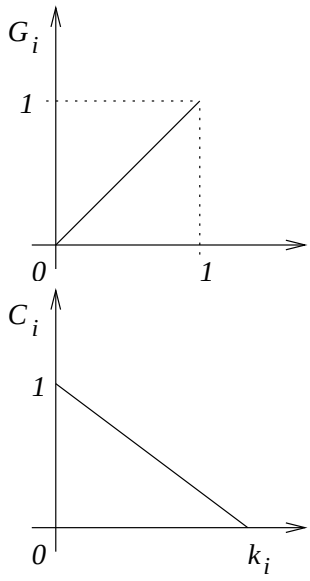
Metrics are declared as part of the domain description, and initialized within a problem statement. Like literals, they may be altered as either a side-effect or a primary effect of actions. In addition to representing consumable resources and execution costs, we also use metrics to define special *quality estimates* that are used to estimate how well the current state of knowledge will achieve a particular subgoal (e.g., `(est-query-quality 0.3)`). Representing knowledge goals in this way lets us compare the relative values of different states when the goals are only partially satisfied. (Discussion of how these quality estimates can be generated is deferred to a later section. For now, we will just assume the estimates are available to us.)

An operator is defined by its preconditions and effects, plus an *execution function* that declares the procedure that will actually be called during execution, along with the operator bindings to pass as arguments.

Preconditions are conjunctions of literal and metric expressions. Literal preconditions may contain typed variables, negation, and inequality constraints on numeric features. Metric preconditions specify inequality constraints on the metric values.

Effects consist of literals to be added to or deleted from the state, and update functions to apply to metrics. We can express uncertainty in these effects by declaring multiple effect sets that enumerate all possible sets of outcomes and the joint probability distribution for each.

A complete specification for a simple `expand-query-with-synonyms` operator is shown in Figure 1. This operator is used to revise a query by adding synonyms of individual query terms. For example, if applied to the single-word query *'car'*, we obtain a new query containing *'car, auto, automobile, machine, motorcar'*. It takes one variable of type `QUERY`, modifies two metrics, and has three possible outcomes: a state in which the query quality has improved, one in which it has remained unchanged (because no synonyms were available), and one in which the quality degrades. A metric precondition on `estimated-query-`




```
:objects ((Q1 query) (LYCOS search-engine) (GOOGLE
search-engine))

:util ((system-time-spent C1 (user-time-spent C2)
      (est-results-quality G1) (est-query-quality G2))

:init (:symbolic ((available-search-engine GOOGLE)
                  (available-search-engine LYCOS)
                  (query-confidence Q1 0.2)
                  (query-terms Q1 2))
      :metric ((user-time-spent 0)
             (system-time-spent 0)
             (est-query-quality 0.2)
             (est-results-quality 0)) )

:goal (:util-criteria (threshold 0.4)
      :succ-criteria (threshold 0.8)
      :utilfn ((1 C1) (10 C2) (100 G1) (20 G2)) )
```

Figure 2: Sample problem statement specifying the initial state, goal criteria and utility function to use.

`quality` limits applicability to queries whose quality is below the given threshold. We also specify a literal precondition and effect that restrict it to one-time application, as this technique is generally only effective the first time it is used on a particular query.

Problem Statement

A specific problem instance is defined by a set of available objects and their types, an initial state description, and a goal specification. The initial state declares the set of literal facts that are known to be true, and assigns initial values to each of the metrics. A goal specification defines the three components needed to evaluate the success or failure of the planning and execution session:

- A function $U(s)$ that estimates state utility from the current metric values in the state
- A minimum utility value for successful termination G_{thresh}
- A minimum satisfiability threshold S_{thresh}

The utility function defines the relative importance of the knowledge goals, costs, and resources. It is used to estimate progress towards our information goals, and guides the action selection process by comparing the expected utilities of the resulting states. Thus, to be useful, a utility function must accurately reflect the user's goals by providing a relative ordering on the states consistent with the user's preferences, correctly mapping goal states to high utility values and non-goal states to low values.

Our domain language currently limits utility functions to weighted combinations of functions for individual metrics m in the domain, each of which produces a normalized value between zero and one:

$$U(s) = \frac{\sum_m w_m U_m(s)}{\sum_m w_m}$$

The utility threshold G_{thresh} specifies the *minimum* utility value required for a satisficing solution. Any executed sequence with a utility value greater than this is assumed to have achieved the goals.

For example, the problem statement shown in Figure 2 defines a simple utility function comprised of two cost metrics and two quality metrics. Each metric has its own utility function (e.g., `system-time-spent` maps to C_1), and the goal statement declares the relative weights to assign to each. The diagrams at the left of the statement depict two very simple mappings; others are possible.

In addition to a utility threshold, a goal declares a satisfiability threshold S_{thresh} between zero and one, indicating how confident we must be in our success likelihood before we can declare planning "successful". A lower value indicates a greater tolerance for risking a lower utility outcome and wasted execution. We elaborate on the relationship between this and our planning and execution algorithm in the next section.

Interleaving Planning and Execution

As previously noted, the incompletely-specified knowledge goals of information-seeking tasks necessitate use of a forward-chaining algorithm. Moreover, although we can estimate the likelihood that our plan succeeds, until we receive feedback from the environment (via evaluation of retrieved documents and explicit user-feedback), we cannot be certain we have actually achieved the goals. Thus, execution and replanning must be an integral part of the planning process.

We use an integrated planning and execution algorithm that performs a best-first search across *belief states*. Our algorithm, presented in Table 1, is supplied with a domain model D, and a problem statement consisting of: an initial belief state I, a user-defined utility-function U, a utility success threshold G_{thresh}, and a value specifying a confidence threshold for termination S_{thresh}.

GenerateOrExecutePlan(domain D, belief state I, utility function U, goal threshold G_{thresh}, satisfiability threshold S_{thresh})

1. Initialize the current belief state and candidate successor belief states
   $B_{cur} \leftarrow I$
   $C \leftarrow$ Successors(D, B_{cur}, U)

2. If ProbabilisticallySatisfies(B_{cur}, U, G_{thresh}, S_{thresh})
   $P \leftarrow$UnexecutedActionSequence(B_{cur})
   If ($P \neq \emptyset$)
      $B_{new} \leftarrow$ Execute(PopFront(P))
      Update(B_{cur}, C, B_{new})
      If (ChooseReplanOrContinue(B_{cur}, C) == replan)
         GenerateOrExecutePlan(D, B_{new}, U, G_{thresh}, S_{thresh})
      Else goto 2.
   Else return success.

3. Else if (ReachedSearchLimit() or
   ($C == \emptyset$ and UnexecutedActionSequence(B_{cur}))== $\emptyset$)) return failure.

4. If (ChooseExecutionOrExpansion(B_{cur},C) == expand)
   $B_{cur} \leftarrow$ ChooseBestOne(C)
   $C \leftarrow (C - B_{cur}) \cup$ Successors(D, B_{cur}, U)
   Goto 2

5. Else
   $P \leftarrow$UnexecutedActionSequence(B_{cur})
   $B_{new} \leftarrow$ Execute(PopFront(P))
   Update(B_{cur}, C, B_{new})
   If ChooseReplanOrContinue(B_{cur}, C) == replan)
      GenerateOrExecutePlan(D, B_{new}, U, G_{thresh}, S_{thresh})
   Else goto 2.

Table 1: The planning and execution algorithm.

Beginning with the initial belief state as the root and an empty plan, the planner evaluates all successor belief states reachable from the current state by applying a single operator, and selects the one with the highest expected utility. The current belief state B_{cur} is updated to the projected belief state, and the selection process repeats. The actual plan, the sequence of operator applications required to transform the initial state into the projected current belief state B_{cur}, is implicitly maintained within each belief state by storing its generating operator and parent belief state.

At each step, the algorithm considers the tradeoff between executing the first unexecuted operator in the plan and continuing to plan with the uncertain outcomes of the projected belief states. If an execution step is carried out, it is followed by an assessment of the need for replanning.

The algorithm terminates when all steps in the plan have been executed and the confidence and utility thresholds for goal satisfaction are met, or there are no additional actions the planner can take. We now provide details on the belief state representation and supporting functions.

State Representation

We represent the state-space at two levels: the *individual state* level, at which individual operators are applied, and the *belief state* level, at which the planning and execution algorithm operates.

An individual state consists of grounded metrics and literals. (Note that we do not make a closed-world assumption; the absence of a fact does not imply its falsehood.) In turn, a belief state consists of a finite set of individual states representing all possible current states and their corresponding likelihoods. This gives us the ability to represent the degree of uncertainty in our state knowledge.

During the planning and execution process, new belief states are generated by the Successors() function shown in Table 2.

Successors(domain D, belief state B, utility function U)

1. Generate the set of all applicable operators A for the current belief state B.

$$A = \bigcap_{s \in states(B)} \{o \mid o \in operators(D), preconds(o) \subseteq s\}$$

2. For each operator $a \in A$, generate a successor belief state B_a,
   and calculate it's expected utility EU

$$B_a = \{s' \mid \forall s \in states(B), \forall e \in effects(a) : literals(s') = literals(s) \cup adds(e) - deletes(e); metrics(s') = apply(mfun(e), metrics(s)) P(s') = P(s)P(e)\}$$

$$B_a.parent \leftarrow B$$

$$B_a.action \leftarrow a$$

$$B_a.EU = \sum_{s' \in states(B_a)} P(s')U(s')$$

3. Return the set of successor belief states.

Table 2: The Successors algorithm.

Given an applicable operator, the function simulates the effects of applying it to the current belief state B_{cur}. The result is a new belief state containing the set of all possible outcome states and the probabilities of seeing each outcome. Note that a pair of outcome states within a belief state may contradict one another. However, once we actually *execute*

the plan steps leading to this belief state, only a single outcome state will remain.

An operator is considered to be *applicable* in the current belief state if all of its preconditions are satisfied in *every state* within the belief state. This ensures we have a valid plan regardless of which state we are actually in. Although it is not explicitly represented in Table 2, the generation of the successor belief states works with fully instantiated operators. Figure 3 shows a pictorial view of the generation process.

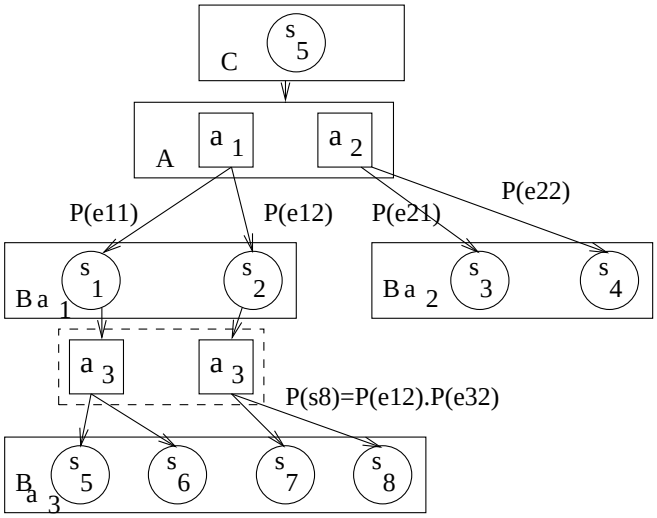


Figure 3: A pictorial view of the generation process.

When a new belief state is created, we also compute its expected utility EU. This value is equal to the weighted sum of the estimated utilities of each outcome state comprising the belief state. It is used for action selection by the function ChooseBestOne() that simply selects the belief state with the highest expected utility. B_{cur} is advanced to this new state, and the operator that generated the belief state becomes the next step in the plan.

Execution

The main advantage of executing in the IR domain is that it provides the planner with additional information that can be used to estimate how well the retrieval process is going, reducing the uncertainty and the number of possible states the planner must consider during forward projection. It may also allow us to terminate earlier if we are fortunate enough to discover our actual state was much better than our projections.

The main disadvantage of execution in this domain is that we cannot recover resources such as time once they are consumed, potentially leading to a worse (less optimal) result than if we had continued planning. In the worst case, where resources are severely limited, foolishly executing steps without sufficient lookahead may result in failure to find *any* solution because we no longer have the resources available to complete the task.

Our system supports three different execution strategies, the first two of which clearly represent the extremes of possible approaches:

- **Conservative (Deliberative)** - Always plan until we are forced to execute, i.e., no more planning is possible
- **Reactive** - Always execute as soon as an unexecuted plan step is available
- **Informed** - Base our execution decision on the features of our current belief state and alternate candidates.

We are currently exploring several different versions of informed strategies. Table 3 shows a simple one.

ChooseExecutionOrExpansion(belief state B, candidates C)

1. If ((C == $\emptyset$) or (**Var(Utilities(** $states(B)$)) > R_{thresh}))
   return execute
2. Else return expand

Table 3: Sample decision for execution vs. planning.

This function chooses execution when it is the only choice available, or when the variance of utilities in the current belief state exceeds a fixed threshold. The intuition here is that planning is probably not a very good option in belief states with bimodal utility distributions consisting of very high and very low utility outcomes. If executing puts us in a high utility state, we may be done without additional planning; if it puts us in a low utility state, then now we know that we are probably better off pursuing other options. Either way, we benefit.

In addition to considering the distribution of utilities in our current belief state and the number of alternatives we have, other strategies include consideration of the potential that execution has to change our view of the world (i.e., the number of alternatives execution introduces), and how costly the next step is to execute.

After executing an operator, an update function records the plan step as having been executed, removes candidates at branches of the search tree above the executed node, and recomputes our current belief state.

Replanning

Currently, for simplicity, we always choose to replan after executing a step. However, we eventually intend to replace this with a true decision point, taking into consideration:

- How consistent our new belief state is with respect to the original plan.
- Whether operators that were previously not applicable have now become applicable.
- The relative cost of updating the existing belief state tree versus the cost of replanning from scratch.

Termination

There are three different termination conditions that may occur during the planning and execution process: a solution may be found, the process may hit a pre-defined search limit, or it may fail because there are no additional actions available to take. We limit out discussion to just the success condition as both failure cases are straightforward.

The function ProbabilisticallySatisfies() (Tabel 4) uses the S_{thresh} satisfiability threshold to determine when the goal is considered "satisfied". The intuition behind this function is that if we have a plan that is already very likely to produce a successful outcome, additional planning may not be very useful. It is probably more productive to switch into execution mode to obtain feedback verifying or disproving the possibility that we are done.

ProbabilisticallySatisfies(belief state B_{cur}, utility function U, goal threshold G_{thresh}, satisfiability threshold S_{thresh})

$$\sum_{s \in B_{cur}} P(s)\delta(s) \geq S_{thresh}$$

where:

$$\delta(s) = \begin{cases} 1 & \text{if } U(s) \geq G_{thresh} \\ 0 & \text{otherwise} \end{cases}$$

Table 4: Probabilistic goal satisfaction.

As defined earlier, this user-specified threshold S_{thresh} is a value between zero and one that indicates how willing the user is to risk a suboptimal solution and wasted execution. A threshold of one indicates a user who is not willing to tolerate any risk (a strong satisfiability requirement); a small non-zero value indicates a user who is very risk tolerant (a weak satisfiability requirement). It is worth pointing out that although this threshold influences how suboptimal the result may be, it does not make any guarantees for optimality. Even in the risk-averse case, we are still only looking for satisficing solutions.

Implementation

We have implemented this approach as part of the INSPIRE (INtegrated System for Planning and Information REtrieval) architecture pictured in Figure 4. INSPIRE consists of three major components: the planner (IRplan), the execution manager (IRexecute), and the user interface. The user interface module is responsible for handling all the direct interactions with the user. It also takes care of translating the user input into an abstract representation that the planner can use. Upon receiving a new request in the form of a text query, the interface stores it in the data-repository, and generates a new problem statement containing the identifier by which the query was indexed, the query features and other state information, and a goal statement. This in turn initiates the planning and execution algorithm contained within the IRplan module. When an execution decision is made, the planner sends an execution request to the IRexecute module. The

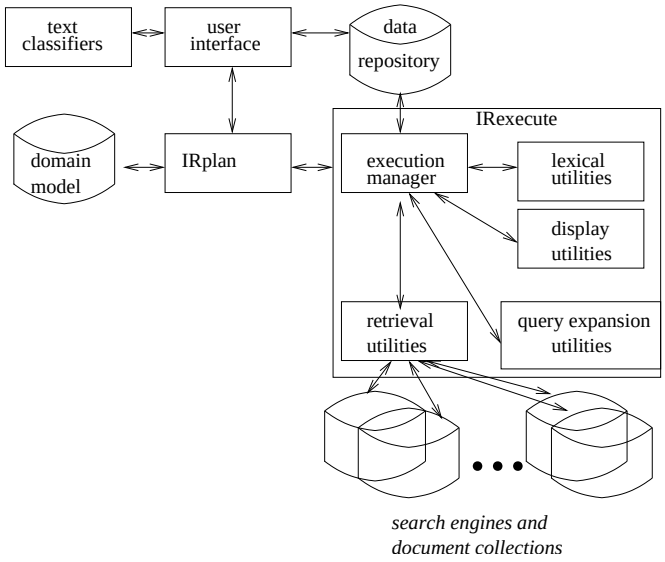


Figure 4: The INSPIRE Architecture.

execution manager retrieves any data it needs from the data repository and proceeds to carry out the requested task. After execution, it saves any new data to the repository, generates a results summary for the planner and returns control to the process.

At present, the display tools required to support post-processing operators are not implemented (as indicated by the dotted box). The results returned to the user consist of the first few lines from the 10 most-highly-ranked documents. We also do not provide mechanisms that allow the user to interrupt the system. User-control is returned only after the planning and execution process is finished or when a feedback request is executed.

Estimating Domain Parameters

One important aspect of our approach, which we have yet to address here, is the origin of the parameters used in the domain model. Currently, we use static prior values for the probabilities of operator outcomes and estimates of metric effects (including quality estimates). However, the development of better models and the addition of the infrastructure necessary to support learning is the major focus of our present research. We briefly outline two of the more interesting issues we are currently considering below.

- **Categorization of information needs**: Users of information retrieval systems have very diverse information needs ranging from requests for a specific document, to exploratory searches (Belkin *et al.* 1995). The ability to pre-classify incoming requests into a set of general goal categories could help us generate problem statements with more precise goal requirements.
- **Quality Estimates**: Research in information retrieval has long focused on defining useful measures of document 'relevance' (Rijsbergen 1979). However, for the purposes of planning, estimating the quality of intermediate

products and resources such as queries and sources are also important. Similar to (Kekäläinen & Järvelin 1998), we are currently evaluating methods for predicting query quality both in stand-alone requests and in the context of incremental search.

Related Work

The challenges presented by the IR task touch upon many different areas of planning research. In this section we describe a representative sample of this work.

The area that is perhaps most relevant to the current work is the problem of planning for information gathering. It has been the focus of considerable attention within the planning community in recent years (e.g.. (Golden 1998; Knoblock 1996; Levy, Rajaraman, & Ordille 1996; Kwok & Weld 1996; Barish *et al.* 2000)). As in the general IR task, the goal of information gathering is to obtain information that satisfies a user's query, often in the face of incomplete knowledge and resource constraints. Unlike the current work however, many of these systems are modeled after database paradigms: they are restricted to querying structured sources, require a model of source contents, and assume that the information goal is perfectly represented by the query. Information gathering systems such as Sage (Knoblock 1995), and XII (Golden, Etzioni, & Weld 1994) interleave execution to gather information, but adopt the simple policy of delaying execution for as long as possible (based on (Ambros-Ingerson & Steel 1988)). PUCCINI (Golden 1998) adopts a slightly more relaxed approach to execution. It allows execution to occur even when it is uncertain that the plan will be satisfied, as long as the outcome is verified afterwards.

In addition to information gathering systems that interleave planning and execution to support sensing, others have considered this problem in a more general planning context. Stone and Veloso (Stone & Veloso 1996) describe an extension to the Prodigy planner to support user-guided execution. They also suggest other execution policies based on abstraction hierarchies and learning from observing the user. Nourbakhsh (Nourbakhsh 1997) develops a set of three types of termination conditions under which the execution can start. They are based on abstraction, assumptive planning, and relative partial plan quality.

The problem of planning for uncertain outcomes has been addressed by several researchers within the context of conditional planning (e.g. (Peot & Smith 1992; Pryor & Collins 1996; Weld, Anderson, & Smith 1998)) and probabilistic planning (e.g. (Draper, Hanks, & Weld 1994; Kushmerick, Hanks, & Weld 1995; Blythe 1998; Blum & Langford 1998)). However, unlike the current work these systems generally work with completely specified goals and don't include execution support. Typically the probabilisitic systems declare success when a probability threshold value is exceeded.

Recently, there has been increased interest in heuristic search techniques for planning. One example is the GPT planner (Bonet & Geffner 2000), which uses forward chaining search heuristics, allows incomplete information, and has a belief-space representation similiar to the work here. We address this challenge within the context of information processing tasks.

Decision-theoretic approaches to planning (e.g. (Haddaway & Suwandi 1994; Haddawy & Hanks 1998; Williamson & Hanks 1994; Boutilier, Dean, & Hanks 1999)) are also similar to the current work in that they use a utility function to evaluate alternative plans and make it easy to take into account resource contraints.

Other relevant work from outside the planning community includes Microsoft's Lumiere project (Horvitz *et al.* 1998), which infers users' needs within the context of a software help system. Microsoft has also worked with developing utility-based models for user interfaces (Horvitz 1999).

Conclusions

In this paper, we have defined a set of challenges presented by the general real-world information retrieval task and described an integrated planning and execution system designed to address them. We have presented a represention that models the various sources of uncertainty in the domain, and uses the uncertainty in the state to guide our decisions for planning, execution, and replanning.

From a planning perspective, the IR domain and our approach provide an interesting framework in which to address a variety of difficult planning problems including: interleaving execution and planning, planning with incompletely specified goals, partial and incremental goal satisfaction, and considering the role of the user in the planning/execution loop. The INSPIRE architecture aims at addressing these issues that are of great importance in bringing automated planning tools to support user's goal decision making and goal achievement. From an IR perspective, the INSPIRE architecture presents an opportunity to provide users with better support in the information retrieval process, as well as to gain greater insight into the value of various IR techhniques.

Our approach has been implemented and thorough evaluation with real users and tasks is part of our ongoing and future work. We are in fact currently addressing question-answering tasks to further focus the general IR task. The INSPIRE architecture is the substrate for our work.

Although the questions of interleaving planning and execution and planning under uncertainty have been previously addressed, our research aims at grounding these questions within the task of information processing. Within this challenging domain that includes users, we envision learning the values of the parameters of our approach.

References

Ambros-Ingerson, J., and Steel, S. 1988. Integrating planning, execution, and monitoring. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, 83–88.

Barish, G.; DiPasquo, D.; Knoblock, C. A.; and Minton, S. 2000. A dataflow approach to agent-based information management. In *Proceedings of the 2000 International Conference on Artificial Intelligence (ICAI-2000)*, 138–139.

Belkin, N. J.; Cool, C.; Stein, A.; and Thiel, U. 1995. Cases, scripts, and information-seeking strategies: On the design of interactive information retrieval systems. *Expert Systems and Applications* 9:379–395.

Blum, A., and Langford, J. 1998. Probabilistic planning in the graphplan framework. In *AIPS98 Workshop on Planning as Combinatorial Search*, 8–12.

Blythe, J. 1998. *Planning Under Uncertainty in Dynamic Domains*. Ph.D. Dissertation, Carnegie Mellon University.

Bonet, B., and Geffner, H. 2000. Planning with incomplete information as heuristic search in belief space. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling*.

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research* 11:1–94.

Draper, D.; Hanks, S.; and Weld, D. 1994. Probabilistic planning with information gathering and contingent execution. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems (AIPS-94)*, 31–36.

Golden, K.; Etzioni, O.; and Weld, D. 1994. Omnipotence without omniscience: Efficient sensor management for planning. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94)*, volume 2, 1048–1054. AAAI Press.

Golden, K. 1998. Leap before you look: Information gathering in the puccini planner. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems (AIPS-98)*, 70–77.

Haddaway, P., and Suwandi, M. 1994. Decision-theoretic refinement planning using inheritance abstraction. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*.

Haddawy, P., and Hanks, S. 1998. Utility models for goal-directed decision-theoretic planners. *Computational Intelligence* 14(3):392–429.

Horvitz, E.; Breese, J.; Heckerman, D.; Hovel, D.; and Rommelse, K. 1998. The lumiere project: Bayesian user modeling for inferring the goals and needs of software users. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, 256–265.

Horvitz, E. 1999. Principles of mixed-initiative user interfaces. In *Proceedings of CHI'99, ACM SIGCHI Conference on Human Factors in Computing Systems*, 159–166.

Kekäläinen, J., and Järvelin, K. 1998. The impact of query structure and query expansion on retrieval performance. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 130–137. Melbourne, Austrailia: ACM.

Knoblock, C. A. 1995. Planning, executing, sensing, and replanning for information gathering. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, 1686–1693.

Knoblock, C. A. 1996. Building a planner for information gathering: A report from the trenches. In *Artificial Intelligence Planning Systems: Proceedings of the Third International Conference (AIPS-96)*, 134–141.

Kushmerick, N.; Hanks, S.; and Weld, D. S. 1995. An algorithm for probabilistic planning. *Artificial Intelligence* 76:239–286.

Kwok, C. T., and Weld, D. 1996. Planning to gather information. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-96)*, volume 1, 32–39. AAAI Press.

Levy, A. Y.; Rajaraman, A.; and Ordille, J. J. 1996. Querying heterogeneous information sources using source descriptions. In *Proceedings of the Twenty-second International Conference on Very Large Databases*, 251–262. Bombay, India: VLDB Endowment, Saratoga, Calif.

Nourbakhsh, I. 1997. *Interleaving Planning and Execution*. Ph.D. Dissertation, Stanford University.

Peot, M. A., and Smith, D. E. 1992. Conditional nonlinear planning. In *Proceedings of the First International Conference on Artificial Intelligence Planning Systems*, 189–197.

Pryor, L., and Collins, G. 1996. Planning for contingencies: A decision-based approach. *Journal of Artificial Intelligence Research* 4:287–339.

Rijsbergen, C. V. 1979. *Information Retrieval, 2nd edition*. Dept. of Computer Science, University of Glasgow.

Stone, P., and Veloso, M. M. 1996. User-guided interleaving of planning and execution. In Ghallab, M., and Milani, A., eds., *New Directions in AI Planning*. IOS Press. 103–112.

Weld, D. S.; Anderson, C. R.; and Smith, D. E. 1998. Extending graphplan to handle uncertainty and sensing actions. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 897–904.

Williamson, M., and Hanks, S. 1994. Optimal planning with a goal-directed utility model. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems (AIPS-94)*, 176–181.