

# **Information Elicitation in Scheduling Problems**

Ulaş Bardak

February 2006

## **Thesis Proposal**

Language Technologies Institute  
School of Computer Science  
Carnegie Mellon University  
Pittsburgh, PA 15213

### **Thesis Committee:**

Jaime Carbonell, Chair

Eugene Fink

Stephen Smith

Sven Koenig, University of Southern California

## Abstract

While trying to satisfy a user's preferences for a resource, partially stated or completely unknown preferences can be very disrupting. Unfortunately most of the time a user will not specify her preferences perfectly, and her partially stated or unknown preferences will be so numerous that she will not be willing to provide clarifications for all. To complicate things further we may not have perfect information about the resource as well and while we can ask an information source about the resource, the problem of "too many potential questions" remains if we were to expect getting answers for all the imperfectly specified properties. In cases like these, we need a mechanism for figuring out which questions would yield a bigger improvement in allocating resources to users so that we can ask as few questions as it is possible to answer and get as high of a improvement as possible

An example of such a problem is optimization for calculating the best assignment of a set of rooms to a set of sessions. The assignment depends on the various properties of rooms as well as the requirements for each session and the properties of these requirements themselves such as their importance levels as perceived by the system. We consider uncertainty in the properties of rooms, and session requirements and importance of sessions as well as the requirements of each session.

The initial work done is embodied in the Elicitor module of Radar Space/Time. The system looks at users' requests for space as well as the world state and determines which questions to ask. The implemented system produces a ranking of questions based on standard deviation calculations for each uncertain attribute. The experiments in this domain show that the system is able to rank potential questions in such a way that answering less than 10% of the potential questions gave as much improvement in the assignment quality as answering all of them.

I propose three main improvements to the already implemented system: First, I will implement a best-first search for finding good questions. Second, I will design a formal syntax for elicitation rules and third, I will implement an algorithm for automatically learning questions that are good to ask in general based on past elicitation system has done. I will also perform empirical evaluation on the final system.

## Table of Contents

<b>1. Introduction.....</b>	<b>1</b>
<b>2. Related Work .....</b>	<b>2</b>
<b>3. Motivating Example .....</b>	<b>5</b>
<b>4. Representation .....</b>	<b>14</b>
4.1. Rooms .....	14
4.2. Events.....	15
4.3. Schedule.....	17
4.4. Preferences .....	18
4.5. Uncertainty.....	19
4.6. Limitations .....	20
<b>5. Algorithms .....</b>	<b>21</b>
<b>6. Preliminary Results .....</b>	<b>27</b>
<b>7. Plan of Work .....</b>	<b>29</b>
7.1. Completed Work.....	29
7.2. Expected Contributions.....	29
7.3. Research Plan.....	31
<b>8. Bibliography .....</b>	<b>33</b>

## List of Figures

Figure 1. Example schedule .....	6
Figure 2. Preference function, which shows the dependency of the assignment quality on the room size for demo session .....	8
Figure 3. Uncertain values.....	11
Figure 4. Conference schedule based on the uncertain knowledge in Figure 4; its expected quality is 0.41, with a standard deviation of 0.70 .....	11
Figure 5. Conference schedule after user answers with an overall utility of 0.87 .....	13
Figure 6. Room hierarchy.....	14
Figure 7. Example of a partial schedule .....	17
Figure 8. Second filter for handling all uncertain variables except preference functions	24
Figure 9. Helper function .....	25
Figure 10. Second filter for handling uncertain preference functions .....	25
Figure 11. Rules for choosing supplemental questions on room attributes .....	26
Figure 12. Evaluation results .....	28
Figure 13. Example elicitation rules .....	30
Figure 14. Example generalized rule .....	31

## List of Tables

Table 1. Available rooms and their properties.....	6
Table 2. Event constraints.....	7
Table 3. Conference events and related preferences.....	8
Table 4. Evaluation of possible questions.....	13
Table 5. Examples of distance constraints: We specify maximal allowed distances (in feet) between events .....	16
Table 6. Examples of relative-time constraints.....	16
Table 7. Examples of different question classes .....	23

# 1. Introduction

There are many reasons why our knowledge about the world is uncertain. Most of the information we miss is simply unavailable to us. Some of it may be available but prohibitively expensive. Furthermore, there may be unknown real-time changes in the world. For example, consider the case of assigning offices to professors and students at a university. We might know which professor requires a large room and which buildings work best for them, but we may not know all related preferences. Also, we may not know the exact size of offices. Furthermore some professors might indicate their preferences as a range of values (i.e. an office around 400 to 600 square feet) instead of a single value, and they may later change their minds without notifying us.

This set of problems raises several important questions. We need a way to represent uncertain information, and find a near-optimal solution based on this uncertain knowledge. We also need to determine what missing pieces of information are critical and then how to obtain these missing pieces.

The setting specific problem we are solving is a conference. There are speakers with preferences and there are rooms that we can assign to the speakers. Each speaker has an importance; each preference has a weight and is a function relating room attributes to utility.

The initial experiments has shown that in this domain we could reduce the potential questions by 90% while getting an improvement in the overall utility matching that of the case where we answer all questions.

I propose three main additions to the already implemented system:

- I will implement a best-first search for finding good questions.
- I will design a formal syntax for elicitation rules
- I will implement an algorithm for automatically learning conditions in which it is generally a good idea to ask questions about an uncertain concept

After these additions are completed, I will also perform empirical evaluation on the final system.

## 2. Related Work

Preference elicitation is important in a wide range of domains from recommender applications on the web to understand preferences of auction participants [Chen and Pu, 2004]. We review some of these applications and discuss their applicability to our problem.

Burke's [1996] FindME systems use domain knowledge to provide the users with assisted browsing when they cannot specify the attributes of the object they are trying to retrieve. The process involves two steps: an initial query and the subsequent tweaking by the user. The tradeoffs, category boundaries and search strategies within the given domain have to be known, and the user may need to do a long series of attribute tweaks [Burke *et al.*, 1997]. This method is applied to different domains such as video rental and general-purpose shopping [Burke, 1999; Stolze and Ströbel, 2003]. This technique is impractical for our problem because the amount of tweaking needed in a continuous domain would render the problem intractable, and we need to provide an impractically large amount of initial knowledge.

Gajos and Weld [2005] employed preference elicitation to find the best interface design for the user. Their method involves example critiquing and posing actual elicitation questions; however, it is limited to binary queries and to elicitation of preference weights.

Pu [2003b] gave guidelines and empirical results for designing a good interface for preference elicitation. Linden [1997], Torrens [2003], and Pu and Faltings [2000; 2002; 2003a] also employed example critiquing to assist the user in finding the best airline tickets. In all cases the user needed to answer a lot of questions for each flight selection, which makes the underlying technique impractical for our problem, which involves a large number of selections. Faltings stated that, for each travel planning task, thirty examples needed to be shown at each step of tweaking [Faltings *et al.*, 2004]. Even compounding critiques together to form more complex critiques at each step such as McCarthy's work [2005] fail in our domain because although they may reduce the amount of critiquing, they cannot deal with continuous preference functions.

Stolze [2003; 2004] introduced a different kind of preference elicitation. He suggested that users were more comfortable in providing the intended use for the desired object rather than desired attributes. Once top matches are returned, the user can fine-tune attributes to get different matches. This methodology was used commercially by AOL's PersonaLogic system, Active Buyer Guide and PurchaseSource systems [Stolze and Rjaibi, 2001]. [Stolze and Ströbel, 2001] also went over a method for building a decision tree in order to decide which questions to ask.

Boutilier [1997; 2003c] focuses on a similar objective as us: doing elicitation in order to facilitate finding a better solution to an optimization problem. He also used Ceteris Paribus Nets to represent preferences where conditional dependence and independence can be reflected under a ceteris paribus interpretation [Boutilier *et al.*, 2003a, 2004a], as well as generalized additive independence models [Braziunas and Boutilier, 2005]. Boutilier deals with linear utility constraints and allow uncertain preferences which is similar to our problem; he decides which question to ask by considering the possible reduction in maximum regret in general purpose domains [Wang and Boutilier, 2003; Boutilier *et al.*, 2005] and specific domains, such as resource allocation in computing systems [Boutilier *et al.*, 2003b; Patrascu *et al.*, 2005]. His approach is limited to elicitation for discrete variables.

Sometimes, user preferences can be approximated through preferences of other "similar" users. A new user expresses preferences on a set of items and the system finds other users who have had similar ratings. The challenge here is picking the right set of preferences to ask the new users to provide. Some approaches involved selecting items that will be most likely for the user to have a preference about, or items that would potentially give most value to the recommender system [Rashid *et al.*, 2002]. Boutilier used expected value of information in order to choose which questions to ask [Boutilier and Zemel, 2003; Boutilier *et al.*, 2003d]. This "collaborative filtering" approach has been used for making recommendations to the users on Internet news [Resnick *et al.*, 1994], videos [Hill *et al.*, 1995], music [Shardanand and Maes, 1995] and products provided by eBay and Amazon [Schafer *et al.*, 2001]. This approach requires users to rank related items; the sheer number of possible "items" as well as failing to account for

the fine-grained detail in user preferences makes this approach impractical for our problem.

Burke [2000b] used similarity-based heuristics to make recommendations based on past user ratings of available options. These heuristics include Euclidean distance [Ha and Haddawy, 1998] and probabilistic distance [Ha and Haddawy, 2003]. This collaborative filtering approach requires a lot of knowledge engineering. A lot of past ratings are required though which creates problems in domains like ours. Burke [2000a] also noted a possibility to integrate collaborative filtering with the knowledge-based approach; however, it would not solve our problem either since our domain would require an impractically large knowledgebase.

Another approach using the similarity between utility models of different users is producing clusters of known utility functions [Chajewska *et al.*, 1998]. When a new user's utility function needs to be elicited, the system asks questions that allow finding the related cluster. The shortcomings of this approach are similar to the previous one. A big database of utility models need to be produced in order to allow effective clustering. Furthermore, as the utility models can widely differ and are very complex in our domain, we are very likely to end up with lots of clusters leading to quite a few questions needed to differentiate between them.

Preference Elicitation is also important in reducing the number of queries in combinatorial auctions where the system needs to figure out which bundle of items the user would be most satisfied with [Smith *et al.*, 2002; Boutilier *et al.*, 2004b; Sandholm and Boutilier, 2006]. Common variations on this kind of elicitation includes incremental auctions [Kress and Boutilier, 2004] and using value queries [Zinkevich *et al.*, 2003; Blum *et al.*, 2004].

The existing research in the area has been successful in finding and improving methods for doing preference elicitation in various scenarios. However, in these scenarios either an optimizer is not the target "client" for the elicitation or the elicitation domain is assumed to be simpler than our domain. Our approach does not make an assumption about the discreteness of the domain or number of possible variations in the preferences. Furthermore, our proposed work is tightly connected with the optimizer and its main objective is getting most important information for improving optimization.

### 3. Motivating Example

We begin with an example of a conference-planning scenario and use it to illustrate the main steps of the elicitation. Suppose that we need to assign rooms to sessions at a small one-day conference, which starts at 11:00 am and ends at 4:30 pm. We can use three rooms: a big auditorium, called Acorn Hall; a smaller classroom, Wean 100; and an even smaller meeting room, Wean 250 (Table 1). These rooms host other events on the same day, and they are available for the conference only at the following times:

Acorn Auditorium: 11:00 am – 1:30 pm and 3:30 pm – 4:30 pm

Wean 100: 11:00 am – 2:30 pm

Wean 250: 12:00 pm – 4:30 pm

We describe each room by a set of properties; in this example, we consider three properties:

- *Size*: Room area in square feet.
- *Stations*: Number of demo stations that can be set up in the room.
- *Mikes*: Number of microphones.

The conference includes five events: a demonstration, a discussion, a tutorial, a workshop, and a meeting of the conference committee (Table 2). For each event, the conference committee specifies the related constraints and preferences on its start time, duration, and room properties. The committee also specifies the importance of each session, which is an integer between 1 and 100.

We construct a conference schedule by assigning a room and time slot to every event. We represent an assignment of a time slot by four variables: a pointer to the event, a pointer to a room, a start time, and a duration. All assignments together form a schedule; that is, we represent a conference schedule as a set of assignments that includes exactly one assignment for each event. We give an example schedule in Figure 2(a), and show its representation by a set of assignments in Figure 2(b).

	Acorn Auditorium	Wean 100	Wean 250
Size	1200	700	500
Stations	10	5	5
Mikes	5	1	2

**Table 1. Available rooms and their properties**

	<i>Acorn Auditorium</i>	<i>Wean 100</i>	<i>Wean 250</i>
<i>11:00</i>	Demo	Tutorial	<b><i>Unavailable</i></b>
<i>11:30</i>			
<i>12:00</i>		Workshop	
<i>12:30</i>			
<i>01:00</i>			
<i>01:30</i>			
<i>02:00</i>	<b><i>Unavailable</i></b>	<b><i>Unavailable</i></b>	
<i>02:30</i>			
<i>03:00</i>	Committee Meeting	<b><i>Unavailable</i></b>	Discussion
<i>03:30</i>			
<i>04:00</i>			

**(a) Schedule of a conference that includes five events**

#	<i>Event</i>	<i>Room</i>	<i>Start</i>	<i>Duration (min.)</i>
1	Demo	Auditorium	11:00 am	150
2	Committee	Auditorium	03:30 pm	60
3	Tutorial	Classroom	11:00 am	60
4	Workshop	Classroom	12:30 pm	120
5	Discussion	Meeting Room	03:00 pm	90

**(b) Representing the schedule by five assignments**

**Figure 1. Example schedule**

We define constraints on acceptable schedules by limiting possible assignments. Specifically, we define acceptable start times, durations, and room properties for each event. For example, we may specify that an acceptable start time for the committee meeting is 1:00 pm or later, an acceptable duration is 15 minutes or more, and a minimal acceptable room size is 300 square feet. In Table 2, we give example constraints for all five events; note that the schedule in Figure 1 satisfies these constraints. If some assignment does not satisfy the given constraints, then the overall schedule is unacceptable. For example, we cannot schedule a 15-minute demo, even if the schedule satisfies all other constraints.

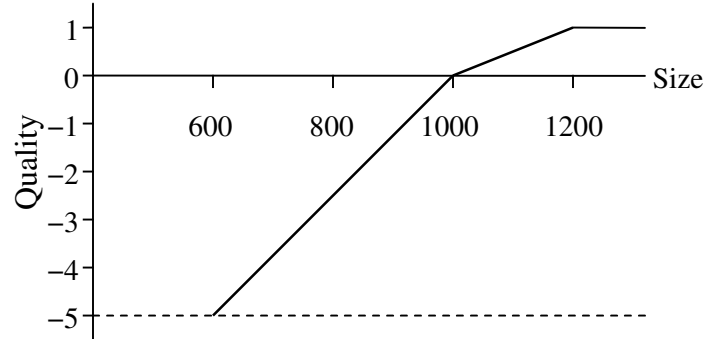
	Demo	Discussion	Tutorial	Committee	Workshop
Importance	50	30	75	10	50
Start Time	11am- $\infty$	11am- $\infty$	11am- $\infty$	3pm- $\infty$	11am- $\infty$
Duration (min)	30- $\infty$	15- $\infty$	15- $\infty$	15- $\infty$	30- $\infty$
Size (sq. feet)	600- $\infty$	200- $\infty$	400-800	400- $\infty$	600- $\infty$
Stations	1- $\infty$	Any	Any	Any	Any
Mikes	Any	Any	Any	Any	Any

**Table 2. Event constraints**

We represent preferences by functions that determine the relative quality of acceptable schedules. We measure a schedule quality on the scale from  $-penalty$  to 1.0, where  $penalty$  is a nonnegative real value that represents the penalty for the worst acceptable schedule. Intuitively, the zero quality corresponds to satisfactory assignments, negative quality values represent poor assignments, and positive values mean unusually good assignments. In the motivating example, we set  $penalty$  to 5.0, which means that all quality values are between  $-5.0$  and 1.0.

For each event, we specify preference functions that represent the desirable selection of the room and time slot. A *preference function* is a piecewise-linear function that shows the dependency of the assignment quality on its start time, duration, or some room property. The domain of this function is the set of acceptable values for the related property, and the range is the quality values between  $-penalty$  and 1.0. Preference functions also have weights, which represent their relative importance.

In Figure 2, we show a preference function for the demo session, which determines the dependency of the assignment quality on the room size. Since the room sizes smaller than 600 square feet are unacceptable (see Table 2), the domain of this function is from 600 to  $+\infty$ . Note that this function is monotonically increasing, and we can specify it by three values of the room size, which correspond to the segment endpoints: the minimal acceptable size (600), which corresponds to the quality of  $-penalty$ ; the satisfactory size (1000), which corresponds to the zero quality; and the minimal values of the “perfect” size (1200), which corresponds to the quality of 1.0. Although the system allows arbitrary piecewise-linear functions, we often use this three-point scheme, because it corresponds to the human intuition. In particular, we use such three-point functions in the motivating example, and we list the respective “minimal,” “satisfactory,” and “best” values in Table 3.



**Figure 2. Preference function, which shows the dependency of the assignment quality on the room size for demo session**

		Demo	Discussion	Tutorial	Committee	Workshop
Importance		50	30	75	10	50
Duration (minutes)	Minimum	60	30	15	15	60
	Satisfactory	120	60	30	30	75
	Best	150	90	60	60	120
Size (square feet)	Minimum	600	200	400	400	600
	Satisfactory	1000	400	600	600	800
	Best	1200	600	800	800	1000
Stations	Minimum	3		0		
	Satisfactory	10	Any	1	Any	Any
	Best	15		2		
Mikes	Minimum			0	0	
	Satisfactory	Any	Any	1	3	Any
	Best			2	4	

**Table 3. Conference events and related preferences**

To compute the quality of an assignment, we instantiate the respective start time, duration, and room properties into the event’s preference functions, and take the weighted sum of their values. In other words, if an event has  $k$  preference functions with values  $q_1, \dots, q_k$  and weights  $w_1, \dots, w_k$ , then the assignment quality is

$$\frac{w_1 \cdot q_1 + w_2 \cdot q_2 + \dots + w_k \cdot q_k}{w_1 + w_2 + \dots + w_k} .$$

The overall schedule quality is a weighted sum of quality values of all events. In other words, if a schedule has  $n$  assignments with quality values  $Qual_1, Qual_2, \dots, Qual_n$ , and respective importances  $imp_1, imp_2, \dots, imp_n$ , then the overall schedule quality is

$$\frac{imp_1 \cdot Qual_1 + imp_2 \cdot Qual_2 + \dots + imp_n \cdot Qual_n}{imp_1 + imp_2 + \dots + imp_n} .$$

We use an optimization algorithm that inputs the description of rooms and events, and searches for a high-quality assignment. The algorithm is based on randomized hill-climbing; it does not guarantee optimality, but it usually finds near-optimal solutions.

When scheduling a conference, we may have incomplete information about room properties, preferences, and importance values; for instance, we may not know the exact size of the auditorium, or the relative importance of the demo session and discussion. We represent uncertain values by probability density functions, approximated by collections of uniform distributions. Specifically, we encode an uncertain value by a set of disjoint intervals that may contain it, with a probability assigned to each interval; the sum of these probabilities must be 1.0.

For example, suppose that the exact size of Wean 250 is unknown. The recent measurements suggest that its size is between 500 and 700 square feet, whereas the old records show that it is between 1000 and 1200. If we trust the measurements more than the old records, but not completely, we may assume that the size is between 500 and 700 with 0.75 probability, and between 250 and 300 with 0.25 probability; in Figure 3(c), we show the corresponding probability density function.

If the description of rooms and events includes uncertainty, the optimizer searches for the schedule that maximizes the mathematical expectation of the quality, and then the system determines the standard deviation of the expected quality. First, it determines the standard deviation for the value of each preference, and then computes the deviation for each event. If an event has  $n$  preference functions, their standard deviations are  $\sigma(q_1), \sigma(q_2), \dots, \sigma(q_n)$ , and the respective preference weights are  $w_1, w_2, \dots, w_n$ , then the standard deviation of the assignment quality is

$$\sqrt{\frac{w_1 \cdot \sigma(q_1)^2 + w_2 \cdot \sigma(q_2)^2 + \dots + w_n \cdot \sigma(q_n)^2}{w_1 + w_2 + \dots + w_n}} .$$

Finally, the system computes the standard deviation of the overall schedule quality. If the standard deviations of assignment quality values are  $\sigma(Qual_1), \sigma(Qual_2), \dots, \sigma(Qual_n)$ , and the respective importances are  $imp_1, imp_2, \dots, imp_n$ , then the standard deviation of the schedule quality is

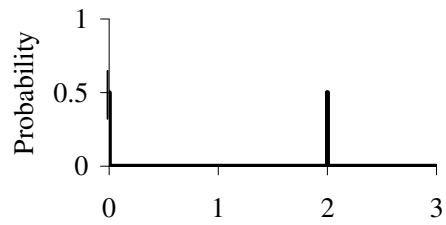
$$\sqrt{\frac{imp_1 \cdot \sigma(Qual_1)^2 + imp_2 \cdot \sigma(Qual_2)^2 + \dots + imp_n \cdot \sigma(Qual_n)^2}{imp_1 + imp_2 + \dots + imp_n}} .$$

This computation is based on the assumption that the probability distributions of uncertain values are independent. If some values are dependent, this computation does not give the exact standard deviation, but it usually provides a good approximation.

If the standard deviation is high, we can try to reduce uncertainty by asking the user to provide more accurate data. For example, we may ask the user to measure the size of a given room, or to find out the exact importance of a given event. If the available knowledge includes a lot of uncertain values, the system may consider thousands of potential questions, some of which may be more important than others. Furthermore, answering some of the questions may be easier than answering others, and we represent the difficulty of answering questions by numeric costs. In this example we consider questions about room attributes, which have difficulty of 10, and questions about event importances, with the difficulty of 50. The system has to analyze the trade-off between the difficulty of answering questions and the resulting uncertainty reduction, and identify the most effective questions.

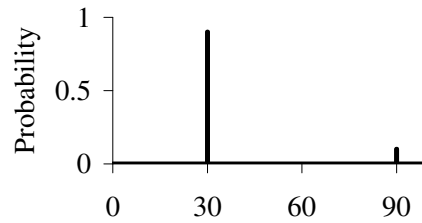
Suppose that we face the uncertainty summarized in Figure 3; that is, we are unsure about the number of microphones in Wean 100, importance of the discussion session, and size of Wean 250. We first apply the optimizer to build a schedule based on the uncertain knowledge, which leads to the schedule in Figure 4.

Probability	Value
0.5	0
0.5	2



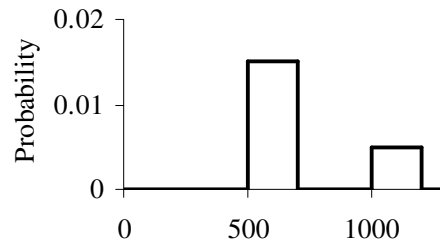
(a) Mikes in Wean 100

Probability	Value
0.9	30
0.1	90



(b) Importance of the discussion session

Probability	Value Range
0.75	500–700
0.25	1000–1200



(c) Size of Wean 250

Figure 3. Uncertain values

	<i>Acorn Auditorium</i>	<i>Wean 100</i>	<i>Wean 250</i>
11:00	Demo	Workshop	<b>Unavailable</b>
11:30			
12:00		Discussion	Tutorial
12:30			
01:00	<b>Unavailable</b>	<b>Unavailable</b>	
01:30			
02:00		Committee Meeting	
02:30			
03:00			
03:30			
04:00			

Figure 4. Conference schedule based on the uncertain knowledge in Figure 4; its expected quality is 0.41, with a standard deviation of 0.70

After generating a schedule, the system computes the impact of each uncertain preference on the overall uncertainty of the schedule quality. To determine this impact, the system calculates the standard deviation of the preference value, and then multiplies it by the preference weight and by the weight of the corresponding request. It also computes the impact of each uncertain room property on the overall uncertainty. Specifically, it computes the impact of the uncertain property on each preference value affected by this property, multiplies the related standard deviations of the preference values by the respective preference and request weights, and then finds the standard deviation of the sum of these preference values. In Table 4, we give the impact of each uncertain value in the motivation example. In Section 6, we will describe this computation in more detail and give the related algorithms.

After computing the impact of uncertain values, the system evaluates the utility of questions about these values. It estimates the utility of a question by subtracting the cost of obtaining the answer from the related impact on the uncertainty. In Table 4(a), we give the utility of each possible question in the motivating example.

The system discards questions with negative utilities, and then uses the remaining questions to elicit additional data; it first asks the highest-utility question, then the question with the second highest utility, and so on. Note that the user does not have to answer every question; she may answer any of these questions and skip the others. If the user answers some of the questions, the system modifies the schedule based on her answers.

For example, suppose that the user has answered two questions as shown in Table 4(b), and skipped the others. The system then updates the schedule using these new data, and produces the schedule shown in Figure 5; the expected quality of this schedule is 0.87, and the standard deviation of its quality is 0.43

<i>Uncertain Value</i>	<i>Impact on the Overall Uncertainty</i>	<i>Question Cost</i>	<i>Question Utility</i>
Capacity preference for Demo	36.10	30	6.10
Stations preference for Tutorial	38.10	40	-1.90
Capacity Preference in Tutorial	20.50	30	-9.50
Microphone Preference in Committee Meeting	0.00	50	-50.00
Importance of Discussion	54.33	50	4.33
Capacity of the meeting room	20.50	10	10.5

(a) Utilities of the potential questions

<i>Uncertain Value</i>	<i>User's Answer</i>
Capacity of the meeting room	1100 square feet
Capacity preference for Demo	Minimum: 600 square feet Satisfactory: 1000 square feet Best: 1200 square feet

(b) Answers provided by the user

Table 4. Evaluation of possible questions

	<i>Auditorium</i>	<i>Classroom</i>	<i>Meeting Room</i>
11:00	Demo	Discussion	<b>Unavailable</b>
11:30			
12:00			
12:30			
01:00			
01:30	<b>Unavailable</b>		Tutorial
02:00			
02:30			
03:00		<b>Unavailable</b>	
03:30			Workshop
04:00	Committee Meeting		

Figure 5. Conference schedule after user answers with an overall utility of 0.87

## 4. Representation

We next describe the main objects in the scheduling domain, which represent available resources, scheduling requirements and preferences, and a specific conference schedule.

We use *room* objects to represent resources; *event* objects to represent conference events, and related constraints; and *assignment* objects to represent the selection of a room and time slot for each event.

We define each of these objects, explain the computation of the quality of a specific schedule, describe the representation of uncertain data, and show its use in the quality computation.

### 4.1. Rooms

The only resource represented in the current system is rooms, which may be in multiple buildings. The system does not keep data about other relevant resources, such as portable equipment or services.

We represent a room by a name and a list of properties, such as its size, number of microphones and the building containing it. The system allows the user to define an arbitrary list of room properties, where each property is either numeric or nominal; for instance, the size of a room is a number, whereas the building that contains a room is a nominal value. In Table 1 (page 8), we give an example of three rooms, represented by three properties.

We also specify distances between rooms, which represent some measure of the difficulty of getting from one room to another. The distance may not be in feet; for example, we may measure distances in walking events.

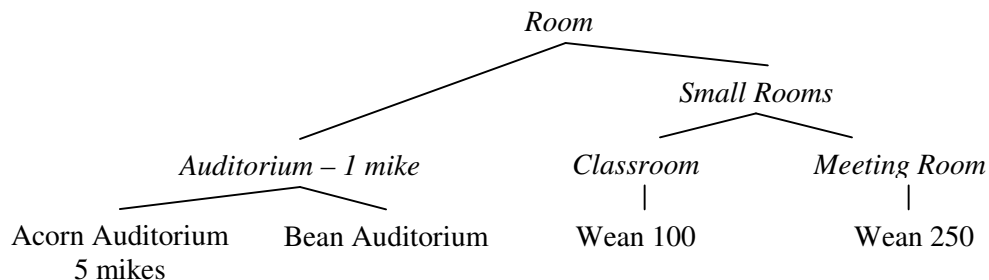


Figure 6. Room hierarchy

For each room, we specify its availability for the conference, represented by a collection of time intervals. For instance, the auditorium in the motivating example is available for two intervals: 11 am – 1:30 pm and 3:30 pm – 4:30 pm.

We arrange rooms into a tree-structured hierarchy, where the top node corresponds to the set of all rooms, other nonleaf nodes are specific room types, and the leaves are individual rooms. In Figure 6, we give an example of a room hierarchy with four room types: auditoriums, small rooms, classrooms, and meeting rooms.

We use the hierarchy to specify default property values, which are inherited by descendent nodes. For example, consider the hierarchy in Figure 6. The default number of microphones for the auditoriums is 1, and this value is inherited by any auditorium without a specified number of microphones. For example, Bean Auditorium inherits this value, whereas Acorn Auditorium has an explicitly specified value, which differs from the default.

## **4.2. Events**

We next describe the representation of conference events, such as workshops and demo sessions, and related scheduling constraints. The representation of an event includes its name, importance, list of hard constraints on scheduling the event, and list of soft preferences.

We represent an event importance by a positive integer; the higher this value, the greater the importance. For instance, the importance of the workshop in the motivating example is 50, whereas the importance of the committee meeting is 10; intuitively, it means that finding a good time slot of the workshop is five times more important than that for the committee meeting.

When specifying an event, we can impose hard constraints on its start time, duration, and properties of a room allocated for this event, as well as its time and position with respect to other events. We define constraints by sets of acceptable values for each of these parameters. We give an example of such constraints in Table 2 (page 9), where the workshop requires a room of size at least 600 square feet, and it has to start no earlier than 11 am and continue for at least 30 minutes.

- **Room-property constraints:** For each room property, we can define acceptable values, which limit the rooms that can be used for the event. For a numeric property, we define acceptable values by a list of nonoverlapping intervals; for a nominal property, we define a list of acceptable values.
- **Distance constraints:** We may constrain the maximal allowed distance of an event from other events, as shown in Table 5, the workshop needs to be at least 50 and at most 100 feet from the demo session. Note that these constraints are different from room-property constraints in that they involve pairs of events.
- **Temporal constraints:** We may define two types of temporal constraints. The first type is the constraints on the start time and duration of an event, defined by a set of acceptable intervals. For example, the demo in the motivating example should start no earlier than 11 am, and its duration should be at least 30 minutes (see Table 2 on page 9). The second type is the constraints on the relative start times of events. We may specify that the difference between the start times of two events should be within a certain interval. For example, we may specify that the discussion should be immediately before the tutorial, or that the demo should start between 2 and 3 hours before the discussion (see Table 6). We may also specify that the times of two events must not overlap.

First Event	Second Event	Maximal Distance
Demo	Tutorial	300
Demo	Committee	250
Demo	Workshop	100
Discussion	Committee	100
Discussion	Workshop	150
Tutorial	Committee	200

**Table 5. Examples of distance constraints: We specify maximal allowed distances (in feet) between events**

First Event	Second Event	Constraint
Demo	Discussion	2-3 hours before
Discussion	Tutorial	Immediately before
Workshop	Demo	Immediately after

**Table 6. Examples of relative-time constraints**

### 4.3. Schedule

When the system builds a schedule, it must satisfy all hard constraints; for instance, if the constraints are as shown in Table 8, it must not consider a schedule where the discussion is after the tutorial.

To build a schedule, we need to assign a specific room and time slot to each event. We represent this assignment by four variables: a pointer to the event, a pointer to a room, a start time, and a duration. Alternatively, we can decide that an event is not a part of the schedule, which is also considered an assignment. We call such an event *rejected*, and represent it internally by setting the related room pointer to nil.

A *partial schedule* is a set of assignments, where different assignments correspond to different events, and some events do not have assignments. A *full schedule* is a set of assignments that includes exactly one assignment for each event. For example, the schedule in Figure 7 is a partial schedule, where we have assigned rooms for the committee meeting, tutorial and discussion, and decided that we do not include demo into the conference, but we have not yet made a decision for the workshop.

Note that we distinguish unassigned and rejected events. An event is rejected if we have decided that it is not a part of the conference, whereas an event is unassigned if we have not yet made any decision. When the system or human user builds a new schedule, it can temporarily mark some events as unassigned; however, the final schedule must not have unassigned events.

	<i>Acorn Auditorium</i>	<i>Wean 100</i>	<i>Wean 250</i>
<i>11:00</i>		Tutorial	<b><i>Unavailable</i></b>
<i>11:30</i>			
<i>12:00</i>			
<i>12:30</i>			
<i>01:00</i>			
<i>01:30</i>	<b><i>Unavailable</i></b>		
<i>02:00</i>			
<i>02:30</i>			
<i>03:00</i>	Committee Meeting	<b><i>Unavailable</i></b>	Discussion
<i>03:30</i>			
<i>04:00</i>			
<i>Rejected: Demo</i>			
<i>Unassigned: Workshop</i>			

Figure 7. Example of a partial schedule

Also note that we always have an option of rejecting an event, regardless of its constraints, and a schedule with rejected events is valid. Thus, we can build a valid schedule by simply rejecting all events, regardless of their constraints; however, its quality would be very poor.

#### 4.4. Preferences

We next define the schedule quality, based on the notion of preferences, which represent soft constraints. We define preferences for the same values as hard constraints; that is, we can specify preferences for room properties, distances between events, and start times and durations.

We represent a preference by a piecewise-linear function, which shows how the quality of an assignment depends on a specific aspect of this assignment. For example, the function in Figure 2 (page 10) shows how the quality of a demo assignment depends on the room size. We define preference functions only on the values allowed by hard constraints; for example, the function in Figure 7 is defined only on the values greater than 600.

For each event, we may specify preferences for each room property, start time, duration, distances from other events, and relative start times with respect to other events. For each preference, we specify its weight, which is a positive integer that shows its relative importance compared to other preferences.

If we reject an event, then this assignment has the worst possible quality, which is *-penalty*. If an event has a room and time slot, then we compute the value of each preference function for this event, and determine the quality of the assignment as the weighted mean of these values. That is, if an event has  $k$  preference functions, their values are  $q_1, \dots, q_k$ , and their weights are  $w_1, \dots, w_k$ , then the assignment quality is

$$\frac{w_1 \cdot q_1 + w_2 \cdot q_2 + \dots + w_k \cdot q_k}{w_1 + w_2 + \dots + w_k}.$$

The overall quality of a full schedule is the weighted sum of the quality values for individual assignments. That is, if a schedule includes  $n$  assignments, their quality values

are  $Qual_1, Qual_2, \dots, Qual_n$ , and their respective importances are  $imp_1, imp_2, \dots, imp_n$ , then the overall schedule quality is

$$\frac{imp_1 \cdot Qual_1 + imp_2 \cdot Qual_2 + \dots + imp_n \cdot Qual_n}{imp_1 + imp_2 + \dots + imp_n}.$$

#### 4.5. Uncertainty

We allow the representation of uncertain information in the system; in particular, room properties, preference functions, and importances of preferences and events may be uncertain. We represent uncertain values by probability density functions, approximated by collections of uniform distributions. Each distribution in the collection has a respective probability, and the sum of the probabilities of all distributions in the collection is 1.0 (see Figure 3 on page 13).

Uncertain preference functions are represented in two different ways by the system. One way is storing a collection of functions with probabilities, where the sum of all the probabilities is one. We also allow storing multiple possible utility values for each turning point of the preference function.

We evaluate a specific assignment by the mathematical expectation of its quality:

$$\frac{E(w_1) \cdot E(q_1) + E(w_2) \cdot E(q_2) + \dots + E(w_n) \cdot E(q_n)}{n}.$$

The expected quality is  $-penalty$  if any hard constraint is violated with a nonzero probability. Otherwise, we calculate the expected quality of a certain preference by the weighted sum of the possible different quality values for that preference. These quality values are based on the relevant room attribute (for example number of microphones, for the preference function for minimum number of microphones) and the shape of the preference function.

We calculate the expected quality of a full schedule as the weighted sum of expected quality values for individual assignments:

$$\frac{E(imp_1) \cdot E(Qual_1) + E(imp_2) \cdot E(Qual_2) + \dots + E(imp_n) \cdot E(Qual_n)}{E(imp_1) + E(imp_2) + \dots + E(imp_n)}.$$

## 4.6. Limitations

We now summarize the main limitations of the described representation; we plan to extend the representation and remove these limitations as part of the proposed research.

- The system does not support multiple inheritance in the room hierarchy; for example, we cannot specify that a room is both a classroom and a meeting room. This limitation prevents computation of appropriate default properties for multi-purpose rooms, and requires manual specification of properties for such rooms.
- The description of scheduling scenarios does not include equipment that can be moved from room to room, such as notebook computers and portable projectors. Also, it does not include services related to the conference, such as food delivery and short-term equipment rental.
- The system does not support uncertainty in hard constraints. If the representation of hard constraints includes uncertainty, the system ensures that the probability of their violation is zero, which means that it internally replaces uncertain constraints with the respective worst-case constraints.
- We do not allow uncertainty for room availability; that is, we cannot represent the chances of unexpectedly losing access to a room during the conference.
- We cannot represent the cost of renting additional rooms, and the system does not analyze the trade-off between these costs and the resulting improvements to the schedule.

## 5. Algorithms

The problem we are addressing is the identification of critical uncertain information. We consider the case of a conference where we need to assign sessions to locations. Each session has a set of requirements together with their respective importance levels and each location has a set of properties. These pieces of information may be uncertain. The final goal is creating a schedule of high quality and reducing the uncertainty in the final schedule. We propose an algorithm which would create questions leading to a significant schedule improvements at a low cost. We assume that all the variables are independent, which is a simplification.

We calculate the overall utility in terms of utility of assigning each conference session to a certain location. We add session utilities, weighing each session by its importance.

$$(5.1) \text{ Utility} = \sum_{s \in \text{Sessions}} \left( \text{weight}(s) \cdot \sum_{p \in \text{Pref}(s)} \text{weight}(p) \cdot \text{utility}(p, s) \right)$$

The utility is a real value between predefined minimum and maximum utility values. In our system, we use -5 and 1. For each preference we calculate the happiness by plugging the relevant location attribute into the relevant preference function. 1 denotes maximum possible happiness and -5 denotes that the organizer of the event would be just as happy without an assignment. We also allow handling of cases where if hard preferences are violated, the whole assignment becomes unacceptable. If the happiness for *any* such preference of a session is violated, we assume that the utility for that whole session is -6 capturing the fact that having this assignment is worse than having no assignment. The weights are scaling factors such that the overall utility itself ends up being in the range -6 and 1.

Uncertainty can exist in the system in weights, in preference functions and in location attributes. All of the uncertain variables, except for preference functions, are composed of a set of value ranges and probabilities. For each of these variables we define the mean and standard deviation to be:

$$(5.2) \quad \bar{x} = \sum p_i \cdot x_i$$

$$(5.3) \quad \sigma_x = \sqrt{\sum p_i \cdot (x_i - \bar{x})^2}$$

The information about the available locations, preferences of session organizers and attributes of the organizers form what the *world state*. We provide this world state as an input to the optimizer.

We show the types of questions the system can generate in Table 7. Note that the system generates questions based on question classes which means that if a new element is added to an existing class, for example a new room attribute called “number of available chairs”, the system can readily pick up that attribute and generate related questions. The system also allows more specialized questions. For example we can define a question template which depends on a specific attribute rather than a class of attributes.

We build a list of all possible questions, and then apply a series of filters to remove unimportant questions.

The first filter ranks each uncertain variable based on the standard deviation of overall utility due to that uncertainty. While we calculate the standard deviation due to a particular variable, we average all the other variables which may factor in. For example, in calculation for an uncertain room capacity preference if the actual room capacity of the assigned room is uncertain as well, we average the room capacity. We also weigh the standard deviation by the importance of the session and the particular preference.

$$(5.4) \quad StandardDeviation(x) = \sum_{s \in sessions} weight(s) \cdot weight(x) \cdot \sigma_{utility(s,x)}$$

<i>Question Class</i>	<i>Sample Elicitation Statement</i>
Preference Function	Find out what discussion session's organizer's preference for room capacity is.
Room Attribute	Find out the capacity of the McConomy Auditorium.
Preference Importance	Find out the importance of organizer's room capacity preference in discussion session.
Session Importance	Find out the importance of discussion session.
Person Importance	Find out the importance of discussion session's organizer.

**Table 7. Examples of different question classes**

After the system makes the ranking, we eliminate all uncertain variables that get a score less than a preset threshold. If the number of variables we have after that elimination is bigger than the number which can be asked, we apply the second filter. There are two versions of the second filter, the first version deals with all uncertain variables except preference functions, whereas the second version takes care of uncertain preference functions. The second filter produces a Boolean decision for each uncertain variable stating whether a variable is worth asking about. The second filter employs a close link with the optimizer, making changes to the world state with respect to the uncertain variable and getting new overall utility scores. It uses these new scores and cost of asking the related question in order to determine the potential gain from posing a question about a particular uncertain variable. We give the pseudo-code for both versions of the second filter in Figures 10, 11, and 12.

Sometimes less questions than the user can potentially answer get generated. In these cases, we use a simple heuristic to generate supplemental questions about room properties. The heuristic takes into account the sessions assigned to each room with uncertain properties as well as the importance of the sessions and the requesters. We give this heuristic in Figure 11.

```

function QUESTION-EVALUATION(pref, cost,  $\epsilon$ , maxSteps) returns a Boolean
inputs: pref, a preference function with domain  $\mathbf{x}$ 
           cost, cost of asking questions
            $\epsilon$ , acceptable difference between question cost and potential gain
           maxSteps, maximum number of iterations
 $\Delta_l \leftarrow$  getHappinessIncrease( $x_{min}[pref]$ , pref)
 $\Delta_r \leftarrow$  getHappinessIncrease( $x_{max}[pref]$ , pref)
interval  $\leftarrow$  createNewInterval( $x_{min}[pref]$ ,  $x_{max}[pref]$ ,  $\Delta_l$ ,  $\Delta_r$ , 1)
highSum  $\leftarrow$   $\Delta_{high}[interval]$ 
lowSum  $\leftarrow$   $\Delta_{low}[interval]$ 
pq  $\leftarrow$  createPriorityQueue(index on diff)
insert(pq, interval)
while(true)
    if (highSum  $\leq$   $(1 + \epsilon) \cdot cost$ ) return false
    if (lowSum  $\geq$   $(1 - \epsilon) \cdot cost$ ) return true
    if (size[pq]  $\geq$  maxSteps) return true
    interval  $\leftarrow$  pop(pq)
    highSum  $\leftarrow$  highSum  $- \Delta_{high}[interval] \cdot prob[iinterval]$ 
    lowSum  $\leftarrow$  lowSum  $- \Delta_{low}[interval] \cdot prob[iinterval]$ 
     $x_{mid} \leftarrow$  getPointAtMidProbability(pref,  $x_l[iinterval]$ ,  $x_r[iinterval]$ )
     $\Delta_{mid} \leftarrow$  getHappinessIncrease( $x_{mid}$ , pref)
    intervall  $\leftarrow$  createNewInterval( $x_l[iinterval]$ ,  $x_{mid}$ ,  $\Delta_l[iinterval]$ ,  $\Delta_{mid}$ ,  $0.5 \cdot prob[iinterval]$ )
    intervalr  $\leftarrow$  createNewInterval( $x_{mid}$ ,  $x_r[iinterval]$ ,  $\Delta_{mid}$ ,  $\Delta_r[iinterval]$ ,  $0.5 \cdot prob[iinterval]$ )
    highSum  $\leftarrow$  highSum  $+ \Delta_{high}[interval_l] \cdot prob[interval_l] + \Delta_{high}[interval_r] \cdot prob[interval_r]$ 
    lowSum  $\leftarrow$  lowSum  $+ \Delta_{low}[interval_l] \cdot prob[interval_l] + \Delta_{low}[interval_r] \cdot prob[interval_r]$ 
    insert(pq, intervall)
    insert(pq, intervalr)

```

**Figure 8. Second filter for handling all uncertain variables except preference functions**

**function** CREATENEWINTERVAL( $x_l, x_r, \Delta_l, \Delta_r, prob$ ) **returns** an Interval

**inputs:**  $x_l$ , left border

$x_r$ , right border

$\Delta_l$ , change in happiness on left half

$\Delta_r$ , change in happiness on right half

$prob$ , interval probability

$i \leftarrow newInterval( )$

$x_l[i] \leftarrow x_l$

$x_r[i] \leftarrow x_r$

$\Delta_l[i] \leftarrow \Delta_l$

$\Delta_r[i] \leftarrow \Delta_r$

$\Delta_{high}[i] \leftarrow \max(\Delta_l[i], \Delta_r[i])$

$\Delta_{low}[i] \leftarrow \min(\Delta_l[i], \Delta_r[i])$

$prob[i] \leftarrow prob$

$diff[i] \leftarrow (\Delta_{high}[i] - \Delta_{low}[i]) \cdot prob[i]$

**return**  $i$

Figure 9. Helper function

**function** QUESTION-EVALUATION( $pref[ ]$ ,  $cost, \delta$ ) **returns** a Boolean

**inputs:**  $pref$ , an array of preference functions with probabilities **prob**

$cost$ , cost of asking questions

$\delta$ , required difference between question cost and potential gain

sortAscending( $pref$  on  $prob$ )

$accumulatedGains \leftarrow 0$

**for each**  $prefFunction$  **in**  $pref$  **do**

$certainWorldstate \leftarrow \text{replace}(pref \text{ in } worldstate \text{ with } prefFunction)$

$accumulatedGains \leftarrow accumulatedGains + \text{getHappinessIncrease}(worldState, newWorldState)$

$\cdot prob[prefFunction]$

**if** ( $accumulatedGains > cost + \delta$ ) **return true**

**return false**

Figure 10. Second filter for handling uncertain preference functions

```

function SUPPLEMENTAL-QUESTIONS(rooms[ ], attrib[ ], ) returns a ranked list
inputs: rooms, an array of rooms
           attrib, a hashtable of uncertain attributes with key room

for each room in rooms do
    questionList ← newQuestionArray( )
    roomWeight ← 1
    for each event in room do
        requesterWeight ← getRequesterWeight(event)
        eventWeight ← getEventWeight(event)
        roomWeight ← roomWeight + requesterWeight · eventWeight
    for each attribute in attrib[room] do
        attributeWeight ← getAttributeWeight(attribute)
        questionWeight ← roomWeight · attributeWeight
        questionList ← insert(attribute with questionWeight)
    sortDescending(questionList on questionWeight)
return questionList

```

**Figure 11. Rules for choosing supplemental questions on room attributes**

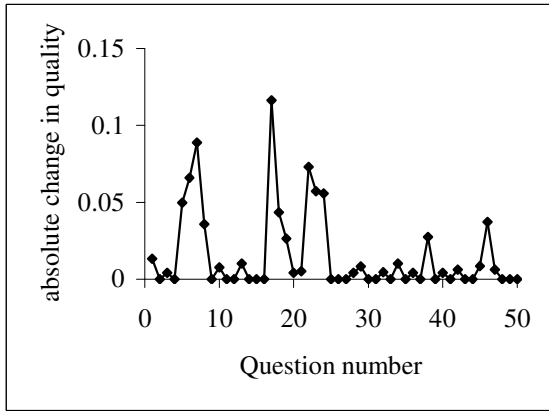
## 6. Preliminary Results

We tested our system on a large conference example with 88 rooms and 84 sessions. Each room had 29 attributes. There were about 700 uncertain room attributes in total and no uncertainty in preferences. We asked the system to produce questions with nonzero weights which gave us 50 questions, ranked in the order of most important to least important from the system's perspective.

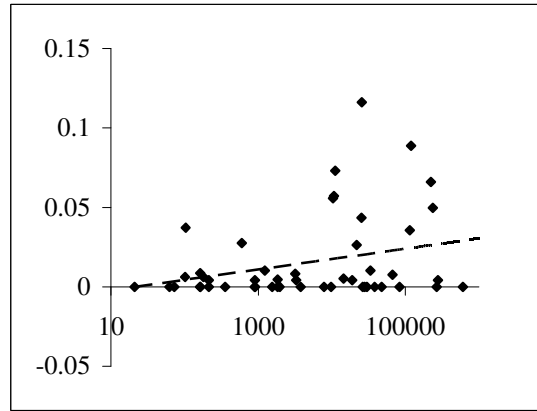
First, we answered each question individually and observed the change in the quality of the schedule the optimizer produces. In this scenario, we only removed one uncertain attribute at a time, and replaced the uncertainty before moving onto the attribute relating to the next question in line. As expected, questions which are ranked higher generally have a much bigger impact on the schedule quality when answered. More specifically, questions in the top 50% (first 25 questions) result in an average impact of 0.03 while the questions in the bottom 50% only have an average impact of 0.005. In Figure 9(a) we show the measured change in schedule quality when each individual question is answered.

We have also considered the case where we answer questions incrementally – that is, we don't put back the uncertain attribute when moving on to the next question. For comparison, we calculated the schedule quality when we run the optimizer with a completely certain world state. For this particular problem, the optimizer produces a schedule of quality -0.65 with a fully uncertain world state whereas the fully certain world state can be used to produce a schedule of quality -0.33. In answering the questions produced by the elicitor, we were able to get a score of -0.36 with only 20 top questions

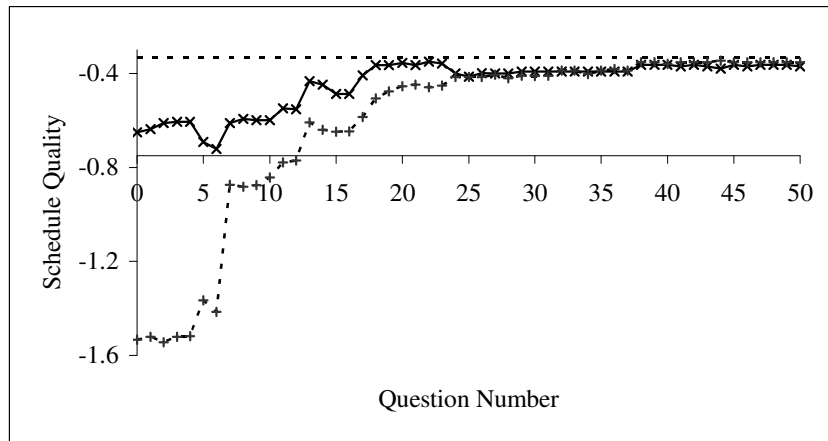
out of the 700 possible questions to ask. The behavior of the schedule quality as more and more questions are answered is shown in Figure 12(c).



(a) Change in schedule quality for each individual question answered individually



(b) Correlation between the elicitation weight of the questions and the absolute change in quality as each question is answered individually.



(c) Schedule quality (bold) as questions are answered incrementally. Dotted line corresponds to the utility achieved with complete certainty. The third series show the estimated schedule quality by the optimizer as questions are answered.

**Figure 12. Evaluation results**

## **7. Plan of Work**

We now summarize the work completed to date (Section 8.1), list the main goals and expected contributions of the thesis research (Section 8.2), and outline the research plan (Section 8.3).

### **7.1. Completed Work**

We have implemented the algorithm described in Section 5, which selects questions based on their potential to reduce the uncertainty of the schedule quality. We have also implemented simple heuristics for choosing questions, listed in Figure 11, which supplement the main algorithm. We have run extensive experiments on the utility of this algorithm (Section 6), which have confirmed that it selects appropriate questions and helps to improve the schedule.

We have integrated the elicitation procedure with the other two components of the scheduling system, which are the optimizer that generates schedules and the graphical interface. The experiments with human subjects have confirmed that the integrated system provides a significant help in building conference schedules.

### **7.2. Expected Contributions**

We plan to develop advanced elicitation strategies, integrate them with the current procedure, design a language for encoding elicitation heuristics, and investigate techniques for learning new heuristics.

**Search for optimal questions:** We will develop a procedure for evaluating the expected utility of questions, based on the algorithm described in Section 5. It will identify questions that allow the greatest potential improvement to the schedule, and analyze the trade-off between the expected improvement and the cost of obtaining the answers. We also plan to develop an algorithm that will perform the best-first search for the highest-utility questions, and compare it with the algorithm in Section 6.

We will integrate the new algorithms with the current elicitation procedure. Specifically, we will apply the current procedure to prune ineffective questions, and then use the best-first search to identify the most effective among the remaining questions.

**Elicitation rules:** We will develop a mechanism for encoding elicitation heuristics by “if-then” rules, which specify appropriate questions based on properties of uncertain data. In Figure 13, we give two example rules, which specify that if the size of an auditorium or a classroom is unknown, the system should try to elicit this information under certain conditions. The procedure for applying these rules will identify the rules that match the available data and resolve conflicts among them.

<p><b>rule</b> UNCERTAIN-AUDITORIUM-SIZE(<i>room</i>)</p> <p><b>Conditions:</b> <math>type[room] = \mathbf{Auditorium}</math></p> <p style="padding-left: 40px;"><math>mean(size[room]) &gt; 10,000</math></p> <p style="padding-left: 40px;"><math>std-dev(size[room]) &gt; 5,000</math></p> <p><b>Action:</b> <math>elicit(size[room])</math></p>
<p><b>rule</b> UNCERTAIN-CLASSROOM-SIZE(<i>room</i>)</p> <p><b>Conditions:</b> <math>type[room] = \mathbf{Classroom}</math></p> <p style="padding-left: 40px;"><math>mean(size[room]) &gt; 10,000</math></p> <p style="padding-left: 40px;"><math>std-dev(size[room]) &gt; 5,000</math></p> <p><b>Action:</b> <math>elicit(size[room])</math></p>

**Figure 13. Example elicitation rules**

We will integrate the rule-based elicitation with the search for the highest-utility questions. The integration will be based on the evaluation of rule reliability; that is, the system will estimate the reliability of applicable rules, and use it to decide whether to rely on the rules or search for the optimal questions.

**Learning of rules:** We will develop a mechanism for learning elicitation rules based on the analysis of past elicitation episodes, analogous to the learning of control rules in the Prodigy planning architecture [Minton, 1988; Perez and Etzioni, 1992; Perez and Carbonell, 1994; Veloso *et al.*, 1995]. The system will identify successful use of questions in the elicitation logs and generate related rules. For example, if the exact information about the size of a large auditorium almost always helps to generate a better schedule, the system may generate the first rule in Figure 13.

The system will use inductive learning to generalize these rules [Borrajo and Veloso, 1993, 1994, 1997]; for example, if the system generates the two rules in Figure 13, it may later generalize them to the rule in Figure 14. Furthermore, the system will evaluate the reliability of these rules by comparing their effectiveness with that of questions selected by the search algorithm.

<p><b>rule</b> UNCERTAIN-ROOM-SIZE(<i>room</i>) <b>Conditions:</b> <math>\text{mean}(\text{size}[room]) &gt; 10,000</math> <math>\text{std-dev}(\text{size}[room]) &gt; 5,000</math> <b>Action:</b> <math>\text{elicit}(\text{size}[room])</math></p>
---

Figure 14. Example generalized rule

### 7.3. Research Plan

I plan to interleave implementation of the algorithms with the analytical and empirical evaluation of their performance, and with writing the corresponding parts of the thesis.

The work will involve four main steps; I expect to spend about six months for each step, and I aim to graduate by the end of 2007.

**Search for optimal questions:** I will implement the best-first search algorithm for identifying the most effective questions. I will then evaluate its efficiency and scalability, and compare the utility of the resulting questions with that of the current elicitation procedure.

**Elicitation rules:** I will develop a syntax for the encoding of elicitation heuristics and hand-code a collection of basic heuristics. I will then evaluate the utility of questions generated by these heuristics, and compare it with that of questions generated by the best-first search. I will also integrate these heuristics with the search algorithm, and evaluate the resulting reduction of the elicitation time.

**Learning of rules:** I will implement a mechanism for learning of the elicitation rules, evaluate its sample and time complexity, and compare the effectiveness of the learned rules with that of hand-coded rules. I expect that the learned rules will be more accurate than hand-coded rules, and that their use will significantly reduce the elicitation time.

**Empirical evaluation:** I will evaluate the integrated elicitation system, which will include all developed techniques. These experiments will include (1) evaluation of the overall effectiveness of the system, (2) ablation studies aimed at determining the contribution of each module into the overall performance, (3) evaluation of the system's learning curve, and (4) comparison of its performance with manual elicitation by human subjects.

## 8. Bibliography

- [Blum *et al.*, 2004] Avrim Blum, Jeffrey Jackson, Tuomas Sandholm, and Martin A. Zinkevich. Preference elicitation and query learning. *Journal of Machine Learning Research*, 5, pages 649–667, 2004.
- [Borrajo and Veloso, 1993] Daniel Borrajo, and Manuela Veloso. Bounded explanation and inductive refinement for acquiring control knowledge. In *Proceedings of the Third International Workshop on Knowledge Compilation and Speedup Learning*, pages 21–27, 1993.
- [Borrajo and Veloso, 1994] Daniel Borrajo, and Manuela Veloso. Incremental learning of control knowledge for nonlinear problem solving. In *Proceedings of the European Conference on Machine Learning*, pages 64–82, 1994.
- [Borrajo and Veloso, 1997] Daniel Borrajo, and Manuela Veloso. Lazy incremental learning of control knowledge for efficiently obtaining quality plans. *Artificial Intelligence Review Journal Special Issue on Lazy Learning*, 11, (1–5), pages 371–405, 1997.
- [Boutilier *et al.*, 2003a] Craig Boutilier, Ronen Brafman, Carmel Domshlak, Holger H. Hoos, and David Poole. CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. In Martha E. Pollack editor, *Journal of Artificial Intelligence Research*. 21. 2003a.
- [Boutilier *et al.*, 2004a] Craig Boutilier, Ronen Brafman, Carmel Domshlak, Holger H. Hoos, and David Poole. Preference-based constraint optimization with CP-nets. *Computational Intelligence*, 20, (2), pages 137–157, 2004a.
- [Boutilier *et al.*, 1997] Craig Boutilier, Ronen Brafman, Chris Geib, and David Poole. A constraint-based approach to preference elicitation and decision making. In *Proceedings of the AAAI Spring Symposium on Qualitative Decision Theory*, pages 19–28, 1997.
- [Boutilier *et al.*, 2003b] Craig Boutilier, Rajarshi Das, Jeffrey O. Kephart, Gerald Tesauro, and William E. Walsh. Cooperative negotiation in autonomic systems using incremental utility elicitation. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*, pages 89–97, 2003b.
- [Boutilier *et al.*, 2003c] Craig Boutilier, Relu Patrascu, Pascal Poupart, and Dale Schuurmans. Constraint-based optimization with the minimax decision criterion. In *Proceedings of the Ninth International Conference on Principles and Practice of Constraint Programming*, pages 168–182, 2003c.
- [Boutilier *et al.*, 2005] Craig Boutilier, Relu Patrascu, Pascal Poupart, and Dale Schuurmans. Regret-based utility elicitation in constraint-based decision problems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 929–934, 2005.
- [Boutilier *et al.*, 2004b] Craig Boutilier, Tuomas Sandholm, and Rob Shields. Eliciting bid taker non-price preferences in (combinatorial) auctions. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, pages 204–211, 2004b.
- [Boutilier and Zemel, 2003] Craig Boutilier, and Richard S. Zemel. Online queries for collaborative filtering. In *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*, pages n/a, 2003.

- [Boutilier *et al.*, 2003d] Craig Boutilier, Richard S. Zemel, and Benjamin Marlin. Active collaborative filtering. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*, pages 98–106, 2003d.
- [Braziunas and Boutilier, 2005] Darius Braziunas, and Craig Boutilier. Local utility elicitation in (GAI) models. In *Proceedings of the Proceedings of the Twentyfirst Conference on Uncertainty in Artificial Intelligence*, pages 42–49, 2005.
- [Burke, 1999] Robin D. Burke. The Wasabi Personal Shopper: A case-based recommender system. In *Proceedings of the Eleventh National Conference on Innovative Applications of Artificial Intelligence*, pages 844–849, 1999.
- [Burke, 2000a] Robin D. Burke. Knowledge-based recommender systems. In Allen Kent editor, *Encyclopedia of Library and Information Systems*. 69, Supplement 32. CRC Press, New York, NY. 2000a.
- [Burke, 2000b] Robin D. Burke. Semantic ratings and heuristic similarity for collaborative filtering. In *Proceedings of the AAAI Workshop on Knowledge-Based Electronic Markets*, pages 14–20, 2000b.
- [Burke *et al.*, 1996] Robin D. Burke, Kristian J. Hammond, and Benjamin C. Young. Knowledge-based navigation of complex information spaces. In *Proceedings of the The Proceedings of The Thirteenth National Conference on Artificial Intelligence*, pages 462–468, 1996.
- [Burke *et al.*, 1997] Robin D. Burke, Kristian J. Hammond, and Benjamin C. Young. The FindMe approach to assisted browsing. *IEEE Expert*, 12, (4), pages 32–40, 1997.
- [Chajewska *et al.*, 1998] Urszula Chajewska, Lise Getoor, Joseph Normal, and Yuval Shahar. Utility elicitation as a classification problem. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 79–88, 1998.
- [Chen and Pu, 2004] Li Chen, and Pearl Pu. Survey of preference elicitation methods *Technical Report No. IC/200467, Swiss Federal Institute of Technology in Lausanne*, pages 1–23, 2004.
- [Faltings *et al.*, 2004] Boi Faltings, Pearl Pu, Marc Torrens, and Paolo Viappiani. Designing example-critiquing interaction. In *Proceedings of the Ninth International Conference on Intelligent User Interfaces*, pages 22–29, 2004.
- [Gajos and Weld, 2005] Krzysztof Gajos, and Daniel S. Weld. Preference elicitation for interface optimization. In *Proceedings of the Eighteenth Annual ACM Symposium on User interface Software and Technology*, pages 173–182, 2005.
- [Ha and Haddawy, 1998] Vu Ha, and Peter Haddawy. Toward case-based preference elicitation: Similarity measures on preference structures. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 193–201, 1998.
- [Ha and Haddawy, 2003] Vu Ha, and Peter Haddawy. Similarity of personal preferences: Theoretical foundations and empirical analysis. *Artificial Intelligence*, 146, (2), pages 149–173, 2003.
- [Hill *et al.*, 1995] Will Hill, Larry Stead, Mark Rosenstein, and George Furnas. Recommending and evaluating choices in a virtual community of use. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 194–201, 1995.
- [Kress and Boutilier, 2004] Alexander Kress, and Craig Boutilier. A study of limited-precision, incremental elicitation in auctions. In *Proceedings of the Third International*

- Joint Conference on Autonomous Agents and Multiagent Systems - Volume 3*, pages 1344–1345, 2004.
- [Linden *et al.*, 1997] Greg Linden, Steve Hanks, and Neal Lesh. Interactive assessment of user preference Models: The Automated Travel Assistant. In *Proceedings of the Sixth Conference on User Modeling*, pages 67–78, 1997.
- [McCarthy *et al.*, 2005] Kevin McCarthy, James Reilly, Lorraine McGinty, and Barry Smyth. Experiments in dynamic critiquing. In *Proceedings of the Tenth International Conference on Intelligent User Interfaces*, pages 175–182, 2005.
- [Minton, 1988] Steven Minton. Learning search control knowledge: An explanation-based approach. Kluwer Academic Publishers, Boston, MA. 1988.
- [Patrascu *et al.*, 2005] Relu Patrascu, Craig Boutilier, Rajarshi Das, Jeffrey O. Kephart, Gerald Tesauro, and William E. Walsh. New approaches to optimization and utility elicitation in autonomic computing. In *Proceedings of the National Conference on Artificial Intelligence*, pages 140–145, 2005.
- [Perez and Carbonell, 1994] M. Alicia Perez, and Jaime Carbonell. Control knowledge to improve plan quality. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, pages 323–328, 1994.
- [Perez and Etzioni, 1992] M. Alicia Perez, and Oren Etzioni. DYNAMIC: A new role for training problems in EBL. In *Proceedings of the Ninth International Conference on Machine Learning*, pages 367-372, 1992.
- [Pu and Faltings, 2000] Pearl Pu, and Boi Faltings. Enriching buyers' experiences: the SmartClient approach. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 289–296, 2000.
- [Pu and Faltings, 2002] Pearl Pu, and Boi Faltings. Personalized navigation of heterogeneous product spaces using SmartClient. In *Proceedings of the Seventh International Conference on Intelligent User Interfaces*, pages 212–213, 2002.
- [Pu *et al.*, 2003a] Pearl Pu, Boi Faltings, and Marc Torrens. User-involved preference elicitation. In *workshop notes, workshop on Configuration, the Eighteenth International Joint Conference on Artificial Intelligence*, 2003a.
- [Pu *et al.*, 2003b] Pearl Pu, Pratyush Kumar, and Boi Faltings. User-involved tradeoff analysis in configuration tasks. In *workshop notes, the Third International Workshop on User-Interaction in Constraint Satisfaction, Ninth International Conference on Principles and Practice of Constraint Programming*, 2003b.
- [Rashid *et al.*, 2002] Al Mamunur Rashid, Istvan Albert, Dan Cosley, Shyong K. Lam, Sean M. McNee, Joseph A. Konstan, and John Riedl. Getting to know you: Learning new user preferences in recommender systems. In *Proceedings of the International Conference on Intelligent User Interfaces*, pages 127–134, 2002.
- [Resnick *et al.*, 1994] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. GroupLens: An open architecture for collaborative filtering of netnews. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work*, pages 175–186, 1994.
- [Sandholm and Boutilier, 2006] Tuomas Sandholm, and Craig Boutilier. Preference Elicitation in Combinatorial Auctions. In Peter Cramton, Yoav Shoham and Richard Steinberg editor, *Combinatorial Auctions*. The MIT Press, Cambridge, MA. 2006.

- [Schafer *et al.*, 2001] Ben J. Schafer, Joseph A. Konstan, and John Riedl. E-commerce recommender applications. *Data Mining and Knowledge Discovery* 5, (1/2), pages 115–153, 2001.
- [Shardanand and Maes, 1995] Upendra Shardanand, and Pattie Maes. Social information filtering: Algorithms for automating "word of mouth". In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pages 210–217, 1995.
- [Smith *et al.*, 2002] Trey Smith, Tuomas Sandholm, and Reid Simmons. Constructing and clearing combinatorial exchanges using preference elicitation. In *Proceedings of the AAAI-02 Workshop on Preferences in AI and CP: Symbolic Approaches*, pages 87–93, 2002.
- [Stolze and Nart, 2004] Markus Stolze, and Fabian Nart. Well-integrated needs-oriented recommender components regarded as helpful. In Elizabeth Dykstra-Erickson and Manfred Tscheligi editor, *Extended abstracts of the 2004 Conference on Human Factors in Computing Systems*. 1. ACM Press, Vienna, Austria. 2004.
- [Stolze and Rjaibi, 2001] Markus Stolze, and Walid Rjaibi. Towards scalable scoring for preference-based Item recommendation. *IEEE Data Engineering Bulletin*, 24, (3), pages 42–49, 2001.
- [Stolze and Ströbel, 2001] Markus Stolze, and Michael Ströbel. Utility-based decision tree optimization: A framework for adaptive interviewing. In *Proceedings of the Eighth International Conference on User Modeling*, pages 105–116, 2001.
- [Stolze and Ströbel, 2003] Markus Stolze, and Michael Ströbel. Dealing with learning in eCommerce product navigation and decision support: the Teaching Salesman Problem. In *Proceedings of the Second Interdisciplinary World Congress on Mass Customization and Personalization*, pages n/a, 2003.
- [Torrens *et al.*, 2003] Marc Torrens, Patrick Herzog, Loic Samson, and Boi Faltings. Electronic commerce technologies: reality: a scalable intelligent travel planner. In *Proceedings of the 2003 ACM Symposium on Applied Computing*, pages 623–630, 2003.
- [Veloso *et al.*, 1995] Manuela Veloso, Jaime Carbonell, Alicia Perez, Daniel Borrajo, Eugene Fink, and Jim Blythe. Integrating planning and learning: The prodigy architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, 7, (1), pages 81–120, 1995.
- [Wang and Boutilier, 2003] Tianhan Wang, and Craig Boutilier. Incremental utility elicitation with the minimax regret decision criterion. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 309–316, 2003.
- [Zinkevich *et al.*, 2003] Martin A. Zinkevich, Avrim Blum, and Tuomas Sandholm. On polynomial-time preference elicitation with value queries. In *Proceedings of the ACM Conference on Electronic Commerce*, pages 176–185, 2003.