Carnegie Mellon University
Research Showcase @ CMU

Computer Science Department | School of Computer Science

10-2008

Scheduling with Uncertain Resources: Learning to Make Reasonable Assumptions

Steven Gardiner
Carnegie Mellon University

Eugene Fink
Carnegie Mellon University

Jaime G. Carbonell
Carnegie Mellon University

Follow this and additional works at: http://repository.cmu.edu/compsci

Published In
Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, 2554-2559.

This Conference Proceeding is brought to you for free and open access by the School of Computer Science at Research Showcase @ CMU. It has been accepted for inclusion in Computer Science Department by an authorized administrator of Research Showcase @ CMU. For more information, please contact research-showcase@andrew.cmu.edu.

Scheduling with Uncertain Resources: Learning to Make Reasonable Assumptions

Steven Gardiner
Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213, USA
gardines@cmu.edu

Eugene Fink
Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213, USA
e.fink@cs.cmu.edu

Jaime G. Carbonell
Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213, USA
jgc@cs.cmu.edu

***Abstract*—We consider the task of scheduling a conference based on incomplete information about resources and constraints, and describe a mechanism for the dynamic learning of related default assumptions, which enable the scheduling system to make reasonable guesses about missing data. We outline the representation of incomplete knowledge, describe the learning procedure, and demonstrate that the learned knowledge improves the scheduling results.**

***Index Terms*—Uncertainty, optimization, elicitation.**

I. Introduction

When we work on a practical scheduling task, we may not have complete knowledge of the available resources and scheduling constraints. For example, when scheduling a conference, we may not know the exact sizes of available rooms or the equipment needs of some speakers. The task of scheduling under uncertainty gives rise to several problems, including the representation of uncertain data, the automated construction of schedules based on these data, and the use of past experience and common sense to make reasonable assumptions about unspecified resources and constraints. To address these problems, we have developed a system for scheduling based on uncertain data [1–6], which has been part of the RADAR project (www.radar.cs.cmu.edu) at Carnegie Mellon University, aimed at building a software agent for assisting an office manager.

We now describe a mechanism for the learning of typical properties of available resources, as well as typical constraints and preferences. We first explain the representation of uncertain data (Section II) and outline the related scheduling mechanism (Section III). We then describe the procedure for the dynamic learning of "common-sense" assumptions (Section IV) and give experiments on its effectiveness (Section V).

II. Representation

We begin with an example of a conference scenario, and use it to illustrate the representation of resources and constraints [2]. Suppose that we need to assign rooms to events at a small one-day conference, which starts at 11:00am and ends at 4:30pm, and that we can use three rooms: auditorium, classroom, and conference room (Table I). Suppose further that these rooms are available for the conference only at the following times:

Auditorium: 11:00am–1:30pm and 3:30pm–4:30pm.
Classroom: 11:00am–2:30pm.
Conf. room: 12:00pm–4:30pm.

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

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

For every room, we define its property values (Table I), as well as its availability, represented by a set of time intervals.

The conference includes five events: discussion, committee meeting, demonstration, tutorial, and workshop (Table II). For each event, we specify its importance, as well as related constraints and preferences. We define constraints by limiting appropriate start times, durations, and room properties. For example, we may indicate that an acceptable start time for the committee meeting is 3:00pm or later, an acceptable duration

TABLE I
Available rooms and their properties.

	Auditorium	Classroom	Conf. room
Size	1200	700	500
Stations	10	5	5
Mikes	5	1	2

TABLE II
Conference events and related constraints and preferences.

		Discussion	Committee	Demo	Tutorial	Workshop
Importance		3	1	5	8	5
Start time	Acceptable	Any	$\geq$ 3pm	Any	11am	Any
	Preferred		3:30pm		11am	
Duration	Acceptable	$\geq$ 30	$\geq$ 30	$\geq$ 60	$\geq$ 30	$\geq$ 60
	Preferred	90	60	150	60	120
Room size	Acceptable	$\geq$ 200	$\geq$ 400	$\geq$ 600	$\geq$ 400	$\geq$ 600
	Preferred	$\geq$ 600	$\geq$ 800	$\geq$ 1200	$\geq$ 600	$\geq$ 1000
Stations	Acceptable	Any	Any	$\geq$ 5	Any	Any
	Preferred			$\geq$ 10	$\geq$ 2	
Mikes	Acceptable	$\geq$ 2	Any	Any	$\geq$ 1	$\geq$ 1
	Preferred	$\geq$ 4			$\geq$ 2	$\geq$ 1

The manuscript was received on March 16, 2008. The described work was supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. FA8750-07-D-0185.

Uncertain value:

$prob_1$: from min_1 to max_1

...

$prob_m$: from min_m to max_m

where

$min_1 \leq max_1 \leq ... \leq min_m \leq max_m$

$prob_1 + ... + prob_m = 1$

Fig. 1. Representation of an uncertain value, which includes multiple disjoint intervals and their respective probabilities.

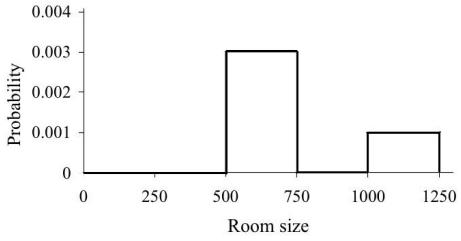

Fig. 2. Probability density function for uncertain room size, which is between 500 and 750 square feet with 0.75 probability, and between 1000 and 1250 with 0.25 probability.

is 30 minutes or more, and an acceptable room size is 400 square feet or more. We may also select preferred values for start times, durations, and room properties, which are subsets of the acceptable values. In Table II, we give constraints and preferences for all events.

If we lack information about some room properties, event importances, constraints, and preferences, we represent the unknown parameters as NIL, indicating the absence of the related data. If we have partial or approximate data about some parameters, we represent them by probability density functions, approximated by sets of uniform distributions. That is, we encode each partially known value by a set of disjoint intervals that may contain it, with a probability assigned to each interval (Figure 1).

For example, suppose that the exact size of the conference room is unknown. Recent measurements suggest that it is between 500 and 750, whereas old records show that it is between 1000 and 1250. If we trust the measurements more than the old records, but not completely, we may assume that the size is between 500 and 750 with 0.75 probability, and between 1000 and 1250 with 0.25 probability; in Figure 2, we show the corresponding probability density function.

III. Scheduling

When constructing a schedule, the system assigns a specific room, start time, and duration to every event. In Figure 3, we show event assignments for the example problem, which satisfy all constraints and most preferences in Table II.

We measure the quality of these assignments on the scale from 0.0 (lowest) to 1.0 (highest). If an assignment violates any hard constraints, its quality is zero. Else, we compute its quality through the rewards for satisfying the related preferences. If a start time, duration, or room property is within the preferred range, the respective reward is 1.0. If it is outside this range, the reward decreases linearly with the distance from the range (Figure 4). If the event has k preferences, and their respective rewards are $r_1, \ldots, r_k$, then the assignment quality is $(r_1 + \ldots + r_k)/k$.

	Auditorium	Classroom	Conf. room
11:00	Demo	Tutorial	***Unavailable***
11:30			
12:00			
12:30		Workshop	
1:00			
1:30			
2:00	***Unavailable***		
2:30		***Unavailable***	
3:00			Discussion
3:30	Committee meeting		
4:00			

Fig. 3. Schedule for the conference scenario in Tables I and II.

The schedule quality is the weighted sum of the quality values for individual assignments. That is, if a schedule includes n events, their quality values are $Qual_1, \ldots, Qual_n$, and their importances are $imp_1, \ldots, imp_n$, then the overall quality is

$$\frac{imp_1 \cdot Qual_1 + \ldots + imp_n \cdot Qual_n}{imp_1 + \ldots + imp_n}.$$

For example, if we use the preferences in Table II, and the schedule is as shown in Figure 3, then the overall schedule quality is 0.86.

If some relevant parameters are unknown, the system evaluates each assignment based on the worst-case scenario; that is, it replaces the unknown parameters with the values that give the lowest score. If some values of an unknown parameter are unacceptable, the system assumes that it is outside the acceptable interval, and thus the assignment quality is zero. If all values are acceptable, but some are less preferable than others, the system uses the least preferable value.

If some parameters are represented by probability density functions, the system evaluates the expected quality of the related assignments, and then computes the expected schedule quality as the weighted sum of the expected quality values of individual assignments:

$$\frac{E(imp_1) \cdot E(Qual_1) + \ldots + E(imp_n) \cdot E(Qual_n)}{E(imp_1) + \ldots + E(imp_n)}.$$

The purpose of scheduling is to maximize this expected quality; the system includes a local-search scheduling procedure, which does not guarantee optimality, but usually finds near-optimal solutions [6].

If more accurate data about some uncertain values may help to improve the schedule, the system asks the user to find out this data. The system's elicitation mechanism analyzes the expected schedule improvements due to potential data requests, selects the most important requests, and presents them to the

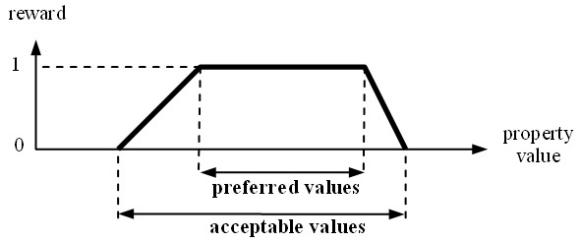

Fig. 4. Reward for satisfying a preference.

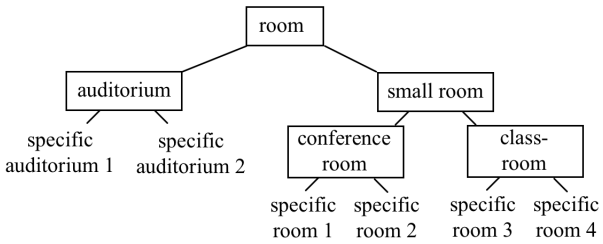

Fig. 5. Example of a room hierarchy.

user [3]. If the user provides the requested data, the system uses them to improve the schedule.

IV. Learning of Defaults

We describe a procedure for the learning of typical room properties and scheduling requirements, which enables the system to make "reasonable assumptions" about missing data. For example, it may learn that most auditoriums are large, whereas most classrooms are small, and that workshops usually require large rooms with microphones. We first describe the data structure for storing typical values of room properties, event importances, constraints, and preferences, which is based on tree-structured object hierarchies, and then present the procedures for the learning and dynamic update of these values.

Object hierarchies: We arrange rooms and events into two tree-structured hierarchies, which specify room and event types. We give an example hierarchy in Figure 5, which divides rooms into auditoriums and small rooms, and subdivides small rooms into classrooms and conference rooms. The system includes a clustering procedure for construction of these hierarchies based on similarities among rooms and events.

Default values: We specify default room properties in the nonleaf nodes of the room hierarchy, and default importances, constraints, and preferences in the nonleaf nodes of the event hierarchy. We represent defaults in the same way as parameters of specific rooms and events; that is, a default value may be a number, NIL, or a set of disjoint intervals with probabilities.

If the values of some room properties, event importances, constraints, and preferences are NIL, the system uses the related defaults. In Figure 6, we give a procedure for retrieving a default room property, which inputs a specific room, and finds its lowest ancestor with a known default; the procedure for events is similar. If the hierarchy does not have the related default in any ancestor node, the system uses NIL in scheduling, as described in Section III.

The procedure inputs a pointer to some *property* of a specific *room*. It returns the property value (if available) or the related default.

VALUE(*room*, *property*)
if *value*[*room*, *property*] is not NIL
  then return *value*[*room*, *property*]
room-type = *parent*[*room*]
while *default*[*property*, *room-type*] is NIL
    and *room-type* is not the root **do**
  room-type = *parent*[*room-type*]
return *default*[*room-type*, *property*]

Fig. 6. Retrieval of a room property. If the specified property of a given room is unknown, the system finds the lowest ancestor with known default, and returns this default.

For example, if the system does not know the size of a specific classroom, it uses the default classroom size. If the "classroom" node does not include this default, the system looks for it in the "small room" node, and then in the root.

Initial learning: If the user does not specify default values, the system learns them from the available data. For each unspecified default, it constructs a probability distribution by combining the known values of the respective leaves.

For example, suppose that the system needs to learn the default classroom size, and it has data about three classrooms. The first known size is exactly 590, the second is between 500 and 600, and the third is between 550 and 680 (Figure 7a). Then, the combined distribution based on these values is as shown in Figure 7(b), which is also illustrated by the solid line in Figure 7(d).

The construction of the exact combined distribution is computationally expensive, and we use a faster approximation technique. For each parameter, we divide the range of its possible values into equal-width intervals, and compute the probability of each interval. For instance, if we divide possible room sizes into intervals of width 50, then we obtain the distribution in Figure 7(c), also illustrated by dashes in Figure 7(d). The user may specify different interval widths for different parameters, thus controlling the trade-off between the learning speed and accuracy. In Figure 8, we give a procedure that constructs the distributions for room-property defaults; the learning of defaults for events is similar.

Dynamic update: The system allows manual modifications of room properties, event importances, constraints, and preferences. The user may update these data because of changes in resources and constraints, and she may also provide more accurate data in response to elicitation requests. When the user makes such updates, the system dynamically modifies the learned defaults. In Figure 9, we give a procedure for the update of default room properties; the procedure for events is similar.

These procedures allow the integration of the elicitation

(a) Known room sizes:
Room 1: exactly 590
Room 2: from 500 to 600
Room 3: from 550 to 680

(b) Exact distribution:
prob 0.167: from 500 to 549
prob 0.236: from 550 to 589
prob 0.339: exactly 590
prob 0.056: from 591 to 600
prob 0.203: from 601 to 680

(c) Approximate distribution:
prob 0.167: from 500 to 549
prob 0.628: from 550 to 599
prob 0.128: from 600 to 649
prob 0.077: from 650 to 699

(d) Probability density:

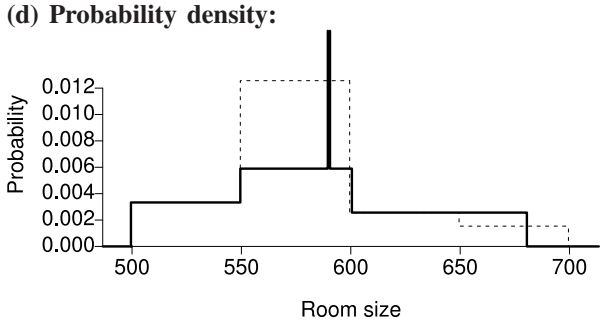

Fig. 7. Example of constructing a typical probability distribution based on partially known values. We show known room sizes (a), the exact distribution based on these sizes (b), and a learned approximate distribution (c). We also give a graphical view of the exact (solid line) and approximate (dashed line) probability density function (d).

process with the learning of "reasonable assumptions." Specifically, when the system receives answers to its questions about particular rooms and events, it learns not only about these rooms and events, but also about related typical values. This integration helps the system to focus on the most essential questions, and avoid questions with "common-sense" answers, such as whether a workshop needs a microphone.

V. Experiments

We have applied the developed system to schedule a four-day conference, which includes eighty-four events, where each event has twenty-three related constraints and preferences. The available resources in this problem include eighty-eight rooms, where each room has twenty-one properties. The problem representation comprises about 9500 parameters; the values of 1200 parameters are uncertain, which means that the system may potentially ask 1200 questions.

The system has first constructed a schedule based on the limited initial data; then, it has asked for additional data, five requests at a time, and modified the schedule after receiving answers to each "batch" of five requests. We have first run these experiments with the learning procedure, which has up-

The procedure inputs a pointer to a default *property* of a specific *room type*, and initializes this default to NIL.

```
INITIAL-DEFAULT(room-type, property)
binned-probs = bins[room-type, property]
for i = 0 to num-bins[room-type, property] − 1 do
    binned-probs[i] = 0
counter[room-type, property] = 0
default[room-type, property] = NIL
```

The procedure inputs a pointer to a default *property* of a specific *room type*, and re-computes it based on the "binned" probabilities.

```
NEW-DEFAULT(room-type, property)
if counter[room-type, property] is 0
    then default[room-type, property] = NIL
    else binned-probs = bins[room-type, property]
         num-values = counter[room-type, property]
         construct the probability distribution for
             default[room-type, property] by dividing the
             probabilities of fixed-width intervals, binned-probs,
             over the number of known values, num-values
```

The procedure inputs a pointer to a default *property* of a specific *room type,* along with a *value* of this property for some room, and updates the default to account for the new value.

```
DEFAULT-UPDATE(room-type, property, value)
binned-probs = bins[room-type, property]
for i = 0 to num-bins[room-type, property] − 1 do
    determine the probability p that value is in the ith bin
    binned-probs[i] = binned-probs[i] + p
counter[room-type, property] = counter[room-type, property] + 1
NEW-DEFAULT(room-type, property)
```

The procedure inputs a pointer to a *property* of a specific *room,* along with a *value* of this property, and updates the defaults of all ancestor nodes to account for this value.

```
ANCESTOR-UPDATE(room, property, value)
if value is NIL then return
room-type = parent[room]
DEFAULT-UPDATE(room-type, property, value)
while room-type is not the root do
    room-type = parent[room-type]
    DEFAULT-UPDATE(room-type, property, value)
```

The procedure learns defaults based on the known property values.

```
INITIAL-LEARNING
for every room-type in the hierarchy do
    for every property of rooms do
        INITIAL-DEFAULT(room-type, property)
for every room do
    for every property do
        ANCESTOR-UPDATE(room, property, value[room, property])
```

Fig. 8. Learning of typical room properties. For each nonleaf node in the hierarchy, the procedure generates its default values by combining the known leaf values into a probability distribution.

The procedure inputs a pointer to a *property* of a specific *room*, and a *new value* of this property. It updates the defaults to account for this value change, and then sets the property to the new value.

It uses the ANCESTOR-UPDATE subroutine in Figure 8, which adds the new value to the default distributions. It also uses an ANCESTOR-REMOVE procedure, similar to ANCESTOR-UPDATE, which subtracts the old value from the distributions.

DYNAMIC-UPDATE(*room*, *property*, *new-value*)
old-value = *value*[*room*, *property*]
ANCESTOR-REMOVE(*room*, *property*, *old-value*)
ANCESTOR-UPDATE(*room*, *property*, *new-value*)
value[*room*, *property*] = *new-value*

Fig. 9. Dynamic update of default room properties. When the user modifies the value of some property of a specific room, the system invokes this procedure to update the related defaults.

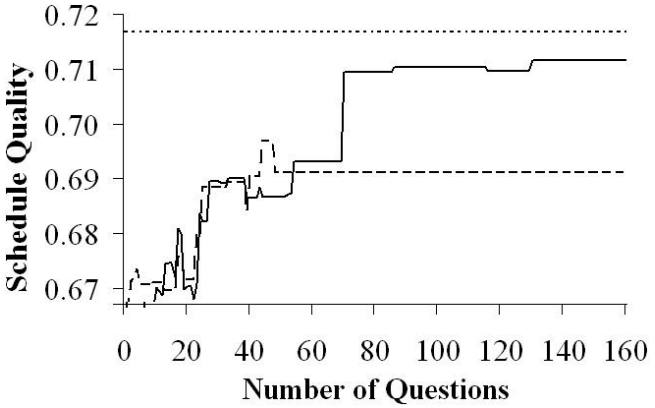

Fig. 10. Dependency of the schedule quality on the number of questions, for the experiments with learning of typical values (solid line) and without learning (dashed line). We also show the quality of the schedule constructed based on the full knowledge of all parameters (horizontal dotted line at the top).

dated the default values after receiving each batch of answers, and then repeated it without learning.

We summarize the results in Figure 10, which shows the dependency of the schedule quality on the number of questions, for the experiment with learning (solid line) and without learning (dashed line). The results confirm that the learning helps to improve the scheduling and elicitation effectiveness, and enables the system to construct a near-optimal schedule after asking fewer questions.

To verify the statistical significance of these results, we have experimented with 950 randomly generated scheduling problems, where the number of uncertain parameters varied from 1 to 1200. For each problem, we first solved it using the learned defaults, and then generated another solution without learning.

We summarize the results in Figure 11, where each point corresponds to one of the problems. The horizontal axis shows the number of uncertain parameters in a problem, whereas the vertical axis in the quality difference between the schedule based on the learned defaults and the schedule constructed

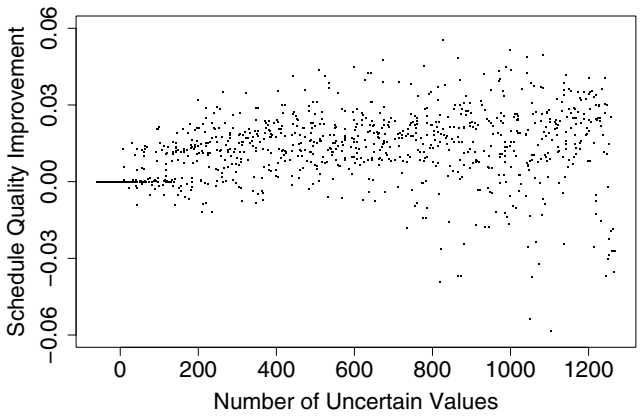

Fig. 11. Experiments with 950 randomly generated problems. The horizontal axis shows the number of uncertain parameters in a problem, and the vertical axis is the increase in the schedule quality due to learning.

without learning. Observe that this difference is positive for most problems, which means that the learning usually leads to a better schedule. We have applied a z-test to verify this observation; it has given the z-value of 27.2, which means that the confidence level is greater than 0.9999.

VI. Concluding remarks

We have described a technique for the dynamic learning of typical resource properties and scheduling requirements, which form the "common-sense" knowledge of the scheduler. The system makes assumptions based on the initial information about resources and constraints, and then dynamically refines these assumptions during the elicitation of additional data.

We have shown that the learned knowledge helps to increase the schedule quality. Furthermore, it improves the elicitation process; specifically, it helps to focus on the most essential questions, which do not have obvious answers. The system quickly learns the standard common-sense requirements and avoids "stupid" questions, such as whether it needs to provide food for lunch or projectors for conference talks.

Acknowledgment

We are grateful to P. Matthew Jennings, Mehrbod Sharifi, Peter Smatana, Blaze Iliev, and Ulas Bardak for their work on the scheduling and elicitation system, which has served as the foundation of the described learning algorithm. We also thank Andrew Yeager for his work on testing and evaluating this system.

References

[1] U. Bardak. "Information elicitation in scheduling problems." Ph.D. Thesis, Language Technologies Institute, Carnegie Mellon University, 2007.

[2] U. Bardak, E. Fink, and J. G. Carbonell. "Scheduling with uncertain resources: Representation and utility function." In *Proceedings of the* IEEE *International Conference on Systems, Man, and Cybernetics*, pp. 1486–1492, 2006.

[3] U. Bardak, E. Fink, C. R. Martens, and J. G. Carbonell. "Scheduling with uncertain resources: Elicitation of additional data." In *Proceedings of the* IEEE *International Conference on Systems, Man, and Cybernetics*, pp. 1493–1498, 2006.

[4] A. Carpentier, M. Sharifi, E. Fink, and J. G. Carbonell. "Scheduling with uncertain resources: Learning to ask the right questions." In *Proceedings of the* IEEE *International Conference on Systems, Man, and Cybernetics*, 2008.

[5] E. Fink, U. Bardak, B. Rothrock, and J. G. Carbonell. "Scheduling with uncertain resources: Collaboration with the user." In *Proceedings of the* IEEE *International Conference on Systems, Man, and Cybernetics*, pp. 11–17, 2006.

[6] E. Fink, P. M. Jennings, U. Bardak, J. Oh, S. F. Smith, and J. G. Carbonell. "Scheduling with uncertain resources: Search for a near-optimal solution." In *Proceedings of the* IEEE *International Conference on Systems, Man, and Cybernetics*, pp. 137–144, 2006.