Scheduling with Uncertain Resources: Representation of Common Knowledge

Eugene Fink e.fink@cs.cmu.edu
P. Matthew Jennings mattj@cs.cmu.edu
Konstantin Salomatin ksalomat@cs.cmu.edu
Jaime G. Carbonell jgc@cs.cmu.edu

School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA

***Abstract*—We describe a system for scheduling a conference based on incomplete information about available resources and scheduling constraints. We explain the representation of uncertain knowledge and related common-sense rules, which allow reasoning based on uncertain and partially missing data.**

***Keywords*—Scheduling, optimization, uncertainty.**

I. Introduction

When we work on a practical scheduling task, we may have incomplete knowledge of the related resources and requirements. We have considered a special case of scheduling under uncertainty, specifically, planning a conference [Bardak, 2008]. We give an example scenario (Section II), explain the representation of uncertain facts (Section III) and "common-sense" rules for making assumptions about missing data (Sections IV–VI), and outline the scheduling algorithm (Section VII).

II. Example problem

We begin with an example scenario, and use it to illustrate the representation of resources and constraints. 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). These rooms host other events, and they 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.
Conference room: 12:00pm–4:30pm.

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

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

The conference includes five events: demonstration, discussion, tutorial, committee meeting, 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, the acceptable start time for the committee meeting is 3:00pm or later, the acceptable duration is 30 minutes or more, and the acceptable room size is 400 square feet or more. We may also specify preferred values for start times, durations, and room properties, which are subsets of acceptable values. In Table II, we give constraints and preferences for all events.

We construct a schedule by assigning a room and time slot to every event. For instance, the schedule in Figure 1 satisfies all constraints and most preferences. The only unsatisfied preferences are the room sizes for the discussion and workshop, and the number of microphones for the discussion and tutorial.

III. Resources and constraints

We describe the representation of resources, scheduling requirements, and specific schedules [Bardak *et al*., 2006].

Rooms: We represent resources by a set of available rooms; the description of a room includes its name and a list of numeric properties (Table I). The user can define an arbitrary list of properties, and then specify their values for each room. She can also specify the availability of each room, represented by a collection of time intervals.

Events: The description of an event includes its name, importance, and related constraints and preferences (Table II). The importance is a positive integer, the constraints are sets of acceptable values for start time, duration, and room properties, and the preferences are sets of preferred values. In addition, the user can define constraints and preferences for relative times of events with respect to other events. For example, she may indicate that two tracks of a workshop must start at the same time.

Uncertainty: We may have incomplete information about resources, event importances, constraints, and preferences. We represent an uncertain value as an interval, encoded by the minimal and maximal possible values. For example, we may specify that the conference-room size is between 500 and 750, the importance of the demo is between 4 and 6, and its minimal acceptable duration is between 60 and 90.

Manuscript received on March 31, 2009. This material is based on work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. NBCHD030010.

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.

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

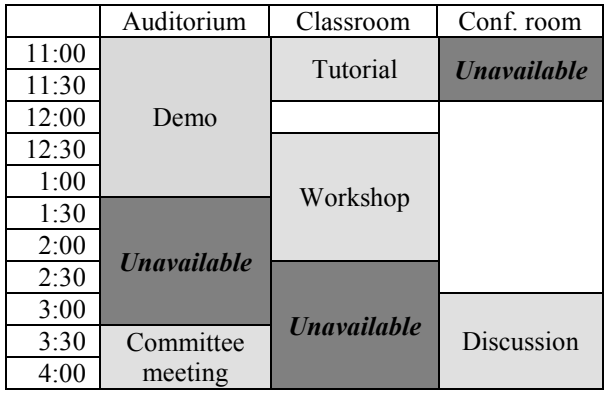

Figure 1. Schedule for the scenario given in Tables I and II.

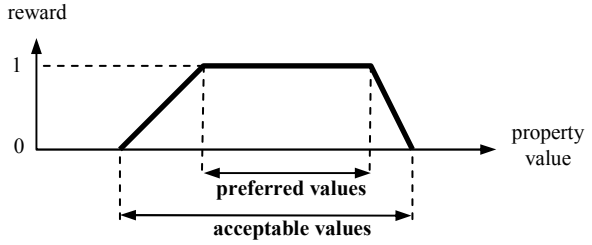

Figure 2. Reward for satisfying a preference.

Schedule: To build a schedule, the system assigns a room and time slot to each event. It represents this assignment by four variables: event name, room name, start time, and duration. Alternatively, it may decide that an event is not in the schedule, which is also considered an assignment. Note that an assignment does *not* include uncertainty; that is, if the system includes an event into the schedule, it selects a specific room and time slot.

Schedule quality: We measure quality on the scale from 0.0 to 1.0; higher values correspond to better schedules. The quality of an assignment depends on how well the selected room and time match the constraints. If the start time, duration, some room property, or time with respect to another event is outside the acceptable set, then the assignment quality is zero. For example, if we allocate a 30-minute slot for the demo, then the assignment quality is zero, even if it satisfies all other constraints. If an event is not part of the schedule, the quality of its assignment is also zero.

If an assignment satisfies all hard constraints, we determine the rewards for satisfying the preferences. If the start time, duration, room property, or time with respect to another event is within the preferred set, then the respective reward is 1.0. If it is outside the preferred set, the reward depends on its distance from this set; it linearly decreases with the distance, as shown in Figure 2. If the event has k preferences, and the respective rewards are $r_1,\ldots, r_k$, then the assignment quality is $(r_1 + \ldots + r_k) / k$.

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

$$\left(imp_1 \cdot Qual_1 + \ldots + imp_n \cdot Qual_n\right) / \left(imp_1 + \ldots + imp_n\right).$$

For example, if we use the preferences in Table II, and the schedule is as shown in Figure 1, then the quality of the assignment for the demo is 1.0, for the discussion is 0.75, for the tutorial is 0.8, for the committee meeting is 1.0, and for the workshop is 0.85, and the overall schedule quality is 0.86.

Expected quality: If the description of rooms and events includes uncertainty, the system computes the mathematical expectation of quality. The computation is based on the assumption that uncertain values are uniformly distributed in their intervals, and that these distributions are independent. The system determines the expected quality of individual assignments, $\mathrm{E}(Qual_1),\ldots, \mathrm{E}(Qual_n)$, as well as the expected values of their importances, $\mathrm{E}(imp_1),\ldots, \mathrm{E}(imp_n)$, and uses them to compute the overall schedule quality, which is

$$\left(\mathrm{E}(imp_1) \cdot \mathrm{E}(Qual_1) + \ldots + \mathrm{E}(imp_n) \cdot \mathrm{E}(Qual_n)\right) / \left(\mathrm{E}(imp_1) + \ldots + \mathrm{E}(imp_n)\right).$$

IV. COMMON-SENSE RULES

When a conference organizer constructs a schedule, she usually makes reasonable assumptions about unknown resources and constraints. For instance, if she does not have data about sizes of some rooms, she may assume that

auditoriums are likely to include several hundred seats, whereas conference rooms are much smaller. If she does not know requirements for specific events, she may assume that tutorials and workshops require projectors and microphones, demos require demo stations, and so on.

We have implemented a mechanism for encoding these assumptions by inference rules. We give two example rules for rooms in Figure 3(a), and two rules for events in Figure 3(b). The room rules specify that the typical size of an auditorium is between 1000 and 2000 square feet, and that large rooms usually include one or two microphones. The event rules indicate that the required room size and number of demo stations is proportional to the attendance, and that workshops with at least twenty attendees need a microphone.

We represent an inference rule by its preconditions and effects. The preconditions are a logical expression that determines when the rule is applicable; for example, Event-Rule A in Figure 3(b) is applicable to demos with at least twenty attendees. The effects are typical values of room or event properties, which may include arithmetic expressions for computing unknown properties based on known properties. For instance, Event-Rule A shows how to compute the required room size and number of demo stations based on attendance. The effects may include ranges for uncertain values. For example, Room-Rule B specifies that large rooms usually include either one or two microphones.

The system allows the user to provide hand-coded rules, and it also has a learning mechanism that generates basic rules automatically [Gardiner *et al*., 2008]. Furthermore, it includes a truth-maintenance procedure for fast propagation of inferences. When the user updates properties of rooms or events, it recomputes the related inferences.

V. Conflict resolution

When the user provides simple intuitive rules, they may occasionally have contradictory effects. For example, consider the rules in Figure 4, where the first rule shows that the number of demo stations that would fit in a room is proportional to the room size; the second indicates that classrooms are not suitable for demos; and the third states that small rooms in the Main Building hold two to four stations. If we apply these rules to a 500-square-foot classroom in the Main Building, they give three different results.

The conflict-resolution procedure is based on assigning integer-valued *priorities* to rules, which represent their relative reliability; greater priority means higher reliability. For instance, the priority of Room-Rule C is 1, and that of Room-Rules D and E is 2 (Figure 4); thus, Room-Rule C is less reliable than the other two. When the user specifies rules, she should mark their priorities; if she does not mark priorities for some rules, the system sets them to 0.

When the system applies rules, it propagates their priorities to the resulting property values. The priority of the values explicitly specified by the user is $+\infty$. On the other hand, the priorities of values derived through the application of inference rules are finite integers. When the system applies a rule, it calculates the *minimum* among the priorities of values used in its preconditions and the priority of the rule itself, and assigns this minimum priority to the derived values. Intuitively, it means that the reliability of a conclusion is determined by the least reliable of its premises.

For instance, consider a room in the Main Building with explicitly specified size of 400. The system applies Room-Rule E (Figure 4) and concludes that the room can hold two to four stations. Since the rule priority is 2, the priority of this conclusion is also 2.

As another example, consider an auditorium of unknown size. The system applies Room-Rule A (Figure 3) and concludes that its size is in the [1000…2000] interval. Since this rule does not have a specified priority, the system assumes that it is 0 by default, and thus the priority of the derived room size is also 0. The system then applies Room-Rule C (Figure 4), thus concluding that the number of stations is in the [8…20] interval. The priority of this rule is 1, and the priority of its precondition is 0, which means that the priority of the conclusion is min(0, 1) = 0.

If several rules give different values for the same property, the system selects the highest-priority value. If several conflicting values have the same highest priority, the system generates the interval that spans all these values. For instance, suppose that it applies the rules in Figure 4 to a 500-square-foot classroom in the Main Building, thus getting the following effects:

Room-Rule C: Stations = [4…5], priority 1
Room-Rule D: Stations = 0, priority 2
Room-Rule E: Stations = [2…4], priority 2

The system discards the effect of Room-Rule C, which has low priority, and then constructs the interval that spans the other values, thus concluding that the number of stations is in the [0…4] interval.

VI. Local and global properties

When a rule generates a property value, it may use other properties of the same object (room or event) in the computation, but it cannot use properties of other objects. For instance, we may specify that the number of demo stations in a room depends on other properties of the *same* room, but we cannot encode its dependency on properties of *another* room.

Thus, the system supports rules that describe dependencies among properties of an object, but it does not allow dependencies among properties of different objects. While this restriction limits the rule expressiveness, it allows very efficient truth maintenance. The system keeps a separate small table of dependencies for each object. When the user updates some properties of an object, the system locally recomputes its other properties, and it does not need to propagate any changes to other objects.

(a) Inference rules for rooms.

ROOM-RULE A
Pre: Type = auditorium
Eff: Size = [1000…2000]

ROOM-RULE B
Pre: Size ≥ 1000
Eff: Mikes = [1…2]

(b) Inference rules for events.

EVENT-RULE A
Pre: Type = demo and Attendance ≥ 20
Eff: Min-Room-Size = 20 · Attendance
Min-Stations = [0.25 · Attendance … 0.5 · Attendance]

EVENT-RULE B
Pre: Type = workshop and Attendance ≥ 20
Eff: Min-Mikes = 1

Figure 3: Examples of inference rules.

ROOM-RULE C
Priority: 1
Pre: Size ≥ 400
Eff: Stations = [Size / 125 … Size / 100]

ROOM-RULE D
Priority: 2
Pre: Type = classroom
Eff: Stations = 0

ROOM-RULE E
Priority: 2
Pre: Building = main and Size ≤ 500
Eff: Stations = [2…4]

Figure 4: Example of conflicting rules, which specify the number of demo stations for different categories of rooms.

(a) Global conference properties.

Total-Attend: Overall attendance
Budget: Conference budget

(b) Inference rule for global properties.

GLOBAL-RULE A
Pre: Total-Attend ≥ 100
Eff: Budget = [600 · Total-Attend … 800 · Total-Attend]

(c) Inference rule for rooms.

ROOM-RULE F
Pre: Type = auditorium and Budget ≥ 100000
Eff: Mikes = 2

(d) Inference rule for events.

EVENT-RULE C
Pre: Type = keynote
Eff: Attendance = [0.6 · Total-Attend … 0.8 · Total-Attend]

Figure 5: Example of using global conference properties.

On the other hand, we allow the use of *global conference properties*, such as the overall number of attendees, in computing properties of rooms and events. The user may define a list of numeric conference properties, much in the same way as she defines a list of room properties. She may explicitly provide values for some of these properties, and she may also specify rules with dependencies among global properties. She may then use global properties in room and event rules.

We give an example in Figure 5, where we use two global properties: the total number of attendees and the conference budget. Global-Rule A in Figure 5 indicates that the budget is proportional to the number of attendees; Room-Rule F states that, if the budget is $100,000 or more, we can afford installing two microphones in every auditorium; and Event-Rule C specifies that the attendance of keynote talks is proportional to the overall conference attendance.

The system maintains a separate dependency table for the global properties. Furthermore, for each object (room or event), it keeps track of the dependency of its local properties on the global-property table. Note that it does *not* need to maintain dependencies among properties of different objects. If the user modifies properties of a room or event, the system propagates this change locally within the object's table. If the user updates a global property, the system recomputes other global properties and then propagates the change to the local tables of the affected objects.

VII. SCHEDULING ALGORITHM

The purpose of scheduling is to increase the expected schedule quality. The system begins with the empty schedule and gradually improves it; at each step, it either assigns a slot to some unscheduled event or moves some scheduled event to a better slot. In Figure 6, we give the main steps of the hill-climbing algorithm, which processes the events in the decreasing order of their expected importances; its more detailed version has been presented in another paper [Fink *et al*., 2006]. When processing an event, the algorithm tries placing it in every time slot consistent with the event's constraints, and selects the slot that gives the greatest quality increase. After processing all events, it returns to the beginning of the sorted list of events and repeats the processing. It terminates the search when the last iteration through all events has not led to any improvements.

```
Sort the events in decreasing order of expected importances.
For every event:
   Create a list of the rooms and time slots that are consistent
      with the sets of acceptable values for this event.
Create the empty schedule;
   that is, mark all events as unscheduled.
Repeat until finding no improvements:
   For every event, in the order of decreasing importances:
      For every room and time slot consistent with this event:
         Move the event into this room and time slot.
         If some events overlap with this slot,
            then remove them from the schedule.
         Recompute the expected schedule quality.
         If the new quality is no greater than the old quality,
            then undo the related schedule changes.
```

Figure 6: Search for a high-quality schedule.

VIII. Concluding remarks

We have described representation of incomplete knowledge in scheduling and related use of "common-sense" rules, which generate assumptions about missing data.

The developed system allows practical conference scheduling, and we have successfully tested it with several real-life scenarios [Bardak, 2008]; however, it addresses only some aspects of conference planning, and leaves many tasks to the human administrator. For instance, one of the IEEE SMC reviewers has pointed out that we have not provided tools for distributing specific presentations among sessions, or for deciding how to divide accepted papers among oral presentations, poster sessions, and workshops. As another example, the system does not consider the use of portable resources, such as overhead projectors, which can be moved from room to room between sessions. The related future challenge is to develop a more general toolset, which will account for most aspects of organizing conferences.

Acknowledgments

We are grateful to Helen Mukomel for her detailed comments, which have helped to focus the presentation.

References

[Fink *et al*., 2006] Eugene Fink, P. Matthew Jennings, Ulas Bardak, Jean Oh, Stephen F. Smith, and Jaime G. Carbonell. Scheduling with uncertain resources: Search for a near-optimal solution. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics,* pages 137–144, 2006.

[Bardak *et al*., 2006] Ulas Bardak, Eugene Fink, and Jaime G. Carbonell. Scheduling with uncertain resources: Representation and utility function. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics,* pages 1486–1492, 2006.

[Bardak, 2008] Ulas Bardak. Information Elicitation for Improving Optimization: Applications in Scheduling Problems. VDM Verlag, Saarbrücken, Germany, 2008.

[Gardiner *et al*., 2008] Steven Gardiner, Eugene Fink, and Jaime G. Carbonell. Scheduling with uncertain resources: Learning to make reasonable assumptions. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics,* pages 2554–2559, 2008.