Interactive, Incremental Scheduling for Virtual Telescopes in Education

Priyang Rathod, Marie desJardins, and Suryakant Sansare
Department of Computer Science and Electrical Engineering
University of Maryland, Baltimore County
1000 Hilltop Circle
Baltimore, MD 21250
{prathod1,mariedj,ssansa1}@cs.umbc.edu

Abstract

The Telescopes in Education (TIE) project, which began in 1992, provides remote access for students to control large observatory telescopes in real time. TIE began with a single telescope, and with manual evaluation and scheduling of student requests. With the sucess of TIE, 20 or more additional telescopes are expected to come on line. With proportionally more student requests, we anticipate that managing the requests and telescope resources will rapidly become too complex and time-consuming to handle manually. To respond to this problem, the Virtual Telescopes in Education (VTIE) project was begun last year. VTIE will provide networked capabilities for automated proposal preparation and evaluation, scheduling, and data archival.

This paper describes the interactive scheduling capability that is under development for VTIE. Accepted observation requests will be incrementally and interactively scheduled by a constraint-based scheduler, and rescheduled as needed in response to dynamically changing weather and telescope conditions. The scheduling system incorporates a novel method for cost-sensitive constraint satisfaction, which will enable modeling of the costs of data gathering and user interaction during schedule repair.

Introduction

Virtual Telescopes in Education (VTIE) is an extension of the Telescopes in Education (TIE) project, which began in 1992. The aim of TIE is to provide remote access to its affiliated telescopes. It enables students to control telescopes from their classrooms and view celestial objects. TIE started with just one telescope, a 24-inch telescope at Mount Wilson Observatory. The students requested time on the telescopes; these requests were scheduled manually. Over the past few years, TIE has received an extremely favorable response from students and educators from hundreds of schools around the world, and from observatory administrators. As a result of this enthusiasm, TIE grew tremendously, from having one telescope at its disposal to more than 20 observatories agreeing to participate in the project. However, manual handling and scheduling of observation requests on these telescopes is clearly not scalable, so there is a need to automate most of these tasks.

The need for scalability and automation led to the inception of VTIE. The objective of VTIE is to integrate the affiliated observatories into one virtual observatory. The emphasis is on making students aware of the science behind observing through a telescope, not just letting them take beautiful pictures. Users will make observation requests to measure some physical property (e.g., brightness or size) of a celestial object. They will also specify a range of observation times that are suitable. Each request will be submitted in the form of a proposal to VTIE; upon successful review, the request will be scheduled on one of the available telescopes. The telescope and time slot will be selected based on telescope availability, telescope properties, and visibility of the object from the telescope at the times specified by the user. Once the schedule is finalized, the user will be notified of the allocated time slot. The user will make the observation in that time slot. Finally, after the observation is complete, the user will submit a paper on the observations made and the results obtained. Upon successful review, these papers will be archived and will become a part of the online VTIE Journal.

VTIE is currently in the implementation phase. A prototype system will be deployed by Fall 2002. More information on the project can be found at the VTIE home page at http://vtie.gsfc.nasa.gov/.

VTIE System Architecture

VTIE is inherently a distributed system, because the observatories are distributed across the globe. Each observatory hosts a telescope server to store information that is local and relevant to each telescope at that observatory, including telescope properties, schedule, and availability. This information is stored in a *virtual telescope object*. There is also a main VTIE server that houses the scheduler, the web interface for the users, and the polling and updating agent, which communicates with the virtual telescopes. The VTIE system architecture is shown in Figure 1.

The requests, submitted in the form of proposals, are preprocessed to determine the telescope properties required; these properties are used to filter the set of telescopes, resulting in a list of telescopes that can satisfy the requirements. A list of time slots that can satisfy the requests, which is a subset of the suitable time slots specified by the user, is also generated. The frequency with which the scheduling process is carried out depends on the type of scheduling used. In batch scheduling, requests are queued in and then sched-

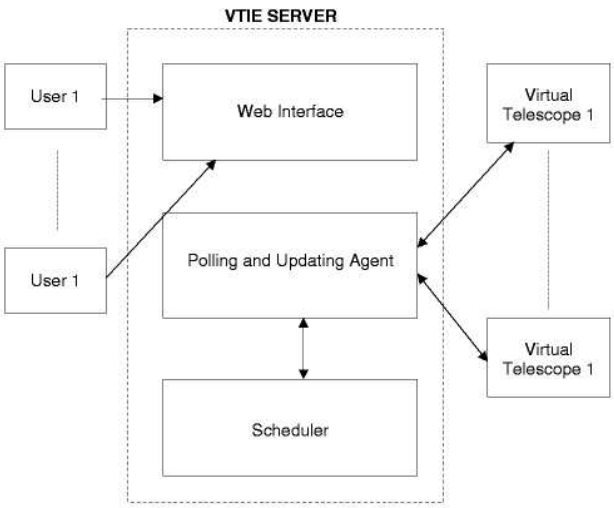

Figure 1: VTIE system architecture

uled as a batch. In incremental scheduling, whenever a request arrives, it is scheduled. Alternatively, requests may be grouped over a period of small duration and then scheduled.

The scheduler asks the polling and updating agent for any information about a telescope it needs to make a scheduling decision. The polling of virtual telescopes takes place during preprocessing as well as scheduling. The polling and updating agent polls the corresponding virtual telescope and asks for the information requested by the scheduler. It receives information from the telescopes regarding their availability and properties, forwarding this information to the scheduler. Once the observations have been scheduled, the scheduler requests the polling and updating agent to send the schedule changes to the appropriate virtual telescopes.

VTIE Scheduling Requirements

The activities to be scheduled are those student requests that have been approved by a review panel. (The scheduling system will also be used to filter infeasible requests—those for which there is no available telescope—early in the review process.) The constraints associated with approved requests include:

- A description of the astronomical phenomenon or specific sky coordinates to observe;
- Telescope characteristics required for the observation, such as diameter and filters;
- The time of day or year that the observation needs to be taken; and
- A priority associated with the requester.
- A preference distribution over time slots.

The domain knowledge to be modeled includes individual telescope capabilities and available time slots, translation between sky coordinates and telescope latitude/longitude and altitude/azimuth, and weather conditions. The scheduling system will use this domain knowledge to identify the telescopes and times that could satisfy each request, allocate time on telescopes to particular student observations, and reschedule observations as needed due to changing telescope and weather conditions.

The conditions that must be satisfied in order for a particular telescope/time combination to satisfy a request include:

- The telescope can point at the specified sky object / coordinates at the selected time.
- The selected time is within the time range specified in the proposal. (If the proposal does not specify a time range, a default is used, currently the next six months.)
- At the selected time, it will be night at the location of the telescope, and the telescope is available (i.e., available for VTIE scheduling and not already scheduled).
- The telescope's capabilities meet the criteria for filters, flux, field of view, etc.

Once the set of possible telescope options has been identified, the scheduler either schedules this individual request in the context of the current schedule or waits for a batch of requests. We are implementing parameterized options to allow scheduling to be done every N requests, every M hours/days, or some combination of these. Once the system is deployed, we will test different approaches to see which works best in practice, under actual scheduler loads.

To increase the user-friendliness of the system, a set of possibilities could be identified and presented to the user, who would select from among the choices. However, this interaction would increase the complexity of scheduling (since multiple time slots would have to be temporarily set aside for a request until a final decision is made), and could introduce significant delays. Again, whether this interactivity is worth the costs can only be determined under actual running conditions.

After the scheduling is complete – whether this is user-driven or scheduler-driven – information about each scheduled request is sent to the appropriate telescope server, which updates its own schedule.

In the next section, we discuss the constraint-based approach that we will be using to developing the automated scheduling capability for VTIE.

Constraint-Based Scheduling

Constraint-based scheduling techniques view a scheduling problem as a constraint satisfaction problem (CSP). A CSP consists of a set of variables, typically with finite domains, and constraints between these variables that restrict the set of legal solutions. A legal solution is an assignment of values to variables such that the constraints are satisfied. (Surveys of CSP methods can be found in (Kumar 1992) and (Barták 1999).) Constraint-based scheduling (see, e.g., (Le Pape 1994)) treats events to be scheduled as variables, and treats times and resources as domains for these variables. In the case of VTIE, student observation requests are the variables; telescope assignments and times are the domains of

these variables. The constraints restrict the resulting schedules to assign appropriate telescopes to observations (as discussed in the previous section), restrict the assignments to times when the telescopes are available for VTIE projects, and include the hard constraint that only one observation can be scheduled on a particular telescope at a time.

Constraint-based scheduling has previously been applied in SPIKE (Johnston & Miller 1994) and the Associate Principal Astronomer (APA) (Bresina *et al.* 1996). SPIKE is used for scheduling the Hubble Space Telescope (HST). The nature of the scheduling problem is quite different from VTIE: a yearly schedule for the HST is produced in a batch process; there are often complex constrained multiple observations; and there is only a single telescope being scheduled. APA is used to automate nightly observations at the Fairborn Observatory in Arizona. The objective of this work is to provide feedback to the scheduler at execution time that can dynamically modify the schedule of observations in order to produce high-quality data. In particular, the duration of observations may vary from what was anticipated by the original schedule. Priorities of observations are a key component of the scheduling problem. Again, the APA domain is quite different from VTIE: there is only one telescope, and the goal is to fully automate the telescope.

In contrast to SPIKE and APA, VTIE has several distinguishing characteristics: First, there is a need for incremental scheduling, since observation requests arrive continuously and must be scheduled far enough in advance to allow students and teachers to prepare for their observation time. Second, the scheduling process needs to be sensitive to the costs of data gathering (e.g., polling the telescopes) and of user interaction (e.g., asking a user whether they are willing to be rescheduled to a different time slot). Third, the scheduling needs to be able to respond to dynamically changing conditions, such as weather conditions, high-priority requests to observe rare astronomical phenomena (such as supernovae), and telescope malfunctions. Our approach will provide all of these capabilities, in a flexible framework that will allow the VTIE administrators to modify the scheduling parameters in response to changing usage patterns.

Our development approach is iterative, producing a series of scheduler releases with successively greater capabilities: (1) batch scheduling, (2) incremental scheduling, (3) cost-sensitive scheduling, and (4) dynamic scheduling. The first two of these have been implemented. The cost-sensitive constraint satisfaction techniques have been implemented and tested, as described in Section , and are currently being applied to develop the latter two capabilities. We now turn to a brief discussion of each of the scheduler releases.

Batch Scheduling

In the first version of the scheduler, a group of observation requests is aggregated over a period of time and scheduled in a batch process. (The group of requests might be all of the requests over the course of some time period – e.g., one week – or the batch scheduler might be run every N requests – e.g., N=20.) This approach allows tradeoffs to be made in scheduling sets of observations. An alternative approach of first-come-first-serve scheduling could cause locally preferable assignments to be made at the expense of globally optimal solutions, potentially violating VTIE program criteria, such as giving priority to certain groups or maximizing usage of the telescope resources. An automated batch approach allows the system to represent and optimize global criteria for a good schedule.

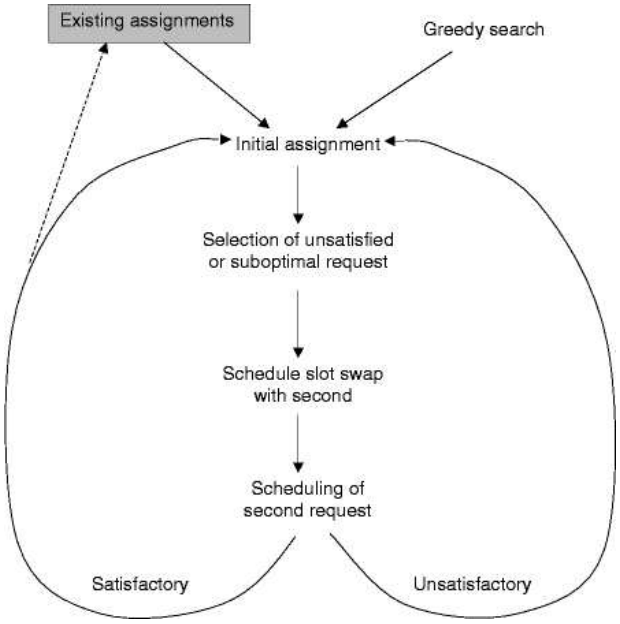

Figure 2: Overview of the scheduling process

In the batch scheduler, requests that were previously scheduled will be treated as unchangeable (i.e., those slots would be unavailable for new requests). As shown in Figure 2, a greedy assignment algorithm is run first, to create an initial assignment of requests to telescope/time slots. An iterative repair method then identifies possible swaps in the schedule—that is, an unsatisfied or suboptimally scheduled request, which could be satisfied by another request's allocated time slot. If the latter request can be scheduled in a different time slot without sacrificing quality, the swap is made. This process of identifying and evaluating swaps repeats until no further improvement can be made.

The overly simplified model of telescope slots where only one request can be scheduled in a slot leads to inefficient telescope utilization. In this simple approach, only one request is scheduled in a telescope slot even if the slot is, for example, of four hour-duration and the request requires an hour. There are two alternative improvements over this approach. In the first alternative, the requests are scheduled in a telescope slot until the combined time required by all the requests scheduled in the slot exceeds the size of the slot. However, this approach makes scheduling very difficult, since it requires dealing with a complex representation of time. In the second alternative approach, the telescope slot is divided into discrete fixed-size blocks of half-hour

intervals. (This representation works well for VTIE, since observations requests and available telescope times are almost always made in half-hour chunks.) The requests are also treated as requiring one or more of such blocks. Thus a telescope slot of length four hours is divided into eight such blocks. The request requiring an hour can be scheduled in two of the eight blocks and the remaining six blocks can be used to schedule other requests. However, during iterative repair, when requests are swapped around among slots, the slots may develop gaps in them which may be unusable. For example, if a request requiring three blocks and scheduled in the middle of a long telescope slot is swapped out by replacing it with a request requiring two blocks, then it leaves a gap of length one block in the middle of the slot. This gap can only be used to schedule a short observation request requiring one block. In the absence of such short requests, after a few iterative repair cycles, the telescope slot may end up having many empty blocks which, if they were contiguous, could have been used to schedule a longer observation request. To overcome this problem, we have also implemented a "defragmentation" process following iterative repair, in which schedules are adjusted in a postprocessing step to minimize the number of gaps in the telescope schedules.

Incremental Scheduling

Extending the batch scheduler to behave incrementally is straightforward in principle. Existing requests are subject to being rescheduled if a new request could make more effective use of that time slot. In Figure 2, the existing assignments (shaded box) feed into the initial assignment, and a feedback loop is added from the search process to possibly reassign those existing requests. Heuristic repair methods (Smith 1994) will be applied to identify schedule modifications that are likely to result in overall improvements.

In the current approach, requests are scheduled as soon as they arrive, i.e., one at a time. The scheduler tries to schedule the newly arrived request in the first available vacant slot in which it can be scheduled. In the absence of a vacant slot, it replaces an already scheduled lower-priority request with the new request if the replaced request can be scheduled somewhere else. There are many factors that need to be considered before rescheduling a request. For example, the scheduler should avoid rescheduling a request that is, for example, scheduled for the next day. If the schedule has already been sent to the user, the scheduler may need to obtain the user's permission before rescheduling that request. There is a cost involved in asking a user and waiting for the reply. In the next phase of development (section), we will explicitly model these costs and develop a method to minimize them.

We have performed a set of experiments with synthetic data representing an overconstrained scheduling problem (500 observations to be scheduled in 2 months). The batch scheduler and the incremental scheduler were tested with the same set of requests and telescopes. The observation requests were generated randomly: requests arrive sequentially, separated by randomized time intervals; the duration of each observation's requested time window is selected from a uniform distribution with specified minimum and maximum lengths. the performance of the incremental scheduler was comparable to that of the batch scheduler. (That is, the same or less number of requests were left unsatisfied.) Once the VTIE prototype is complete, we plan to perform a more comprehensive set of experiments with more realistic telescope and observation data.

Cost-Sensitive Scheduling

We are developing a method of cost-sensitive constraint satisfaction (CSCS) that incorporates the costs of constraint checking into the constraint reasoning process (Sansare 2002). Using this method will allow the scheduler to be sensitive to the cost of evaluating alternatives (testing constraints). Constraints that are expensive or difficult to test are treated as having an additional cost that must be minimized. The goal is to develop a constraint satisfaction method that minimizes the total cost of constraint checking.

The number of constraint checks is commonly used as an evaluation metric for the time performance of constraint methods, but to our knowledge, there is no previous research that allows constraints to have varying costs, or that treats the cost of constraint checks as an explicit optimization criterion. In our initial development, we have used a combination of variable-ordering heuristics (e.g., placing the highest-cost constraints at the end of the search, attempting to minimize the total cost of constraint checks), intelligent backtracking (to avoid re-checking where possible), and constraint propagation over low-cost constraints (to reduce the domains of the variables as much as possible before checking the more costly constraints). We have tested the CSCS methods in two domains, N-queens and a simplified sports scheduling domain. Our initial results indicate that using constraint satisfaction methods that are sensitive to constraint-checking costs can significantly reduce the search cost associated with finding a solution.

We analyzed four search heuristics for solving the N-queens problem and three heuristics for sports scheduling. The heuristics vary in their variable and value selection policies, and in when certain high-cost constraints are evaluated. In each domain, we used a forward-checking algorithm (Haralick & Elliot 1980), modified by the heuristics' recommendations about when to check high-cost constraints.

The N-queens problem is a "puzzle-style" CSP with very regular structure. The sports scheduling domain is a simple model of a common scheduling problem. Artificial costs were injected into the domains to simulate the constraint costs for the domains, assuming there does exist a resource or temporal cost to retrieve the data. We used random constraint set costs in this domain, to investigate the effectiveness of cost-sensitive heuristics where costs have no natural interpretation or inherent structure. The N-queens problem is useful as a benchmark, because it is well understood and difficult to solve for large N. However, it is unnatural in many ways, because the structure of the constraint network is so regular. The sports scheduling domain has more similarities to the VTIE scheduling domain.

The heuristics used for N-queens problem were:

- Most Constrained First (MCF) variable ordering.
- Deferred Constraints (DC): the most costly constraints are deferred until later in the search.
- Lowest Cost First (LCF): chooses the variable that has the lowest cost to instantiate next.
- Highest Cost First (HCF): picks the most costly variable for instantiation.

The results for N-queens problem showed that the Lowest Cost First (LCF) and Highest Cost First (HCF) heuristics give better results than the first two algorithms since they intelligently alter the order of processing of the queens.

The N-queens problem is an example of a fully connected constraint network. The DC heuristic performs quite badly in this domain because of this tight connectivity: there are few solutions, and the variable instantiations are tightly interdependent, so deferring the constraints to the end of the search results in a substantial amount of backtracking. One might expect the DC heuristic to work better in a domain where the expensive constraints are more likely to succeed, or where there are fewer interrelationships among the variables. In the VTIE domain, the costliest constraints are those that involve rescheduling a previously scheduled request. It might be desirable to postpone contacting the affected user until all of the other constraints have been checked. That is, if no valid solution exists that includes the rescheduled request, there is no point in checking to see whether a swap can be made.

LCF does slightly better than MCF, but not much. This can be explained by the uniformity of the constraint network and the random distribution of costly constraints: instantiating the variables with more zero-cost constraints doesn't reduce the number of higher-cost constraints; it just postpones them. Instantiating the variables with the largest number of costly constraints first works better than MCF or LCF. Our hypothesized explanation hinges on the fact that in the N-queens domain, most of the backtracking occurs late in the search process. In any case, the reduction in cost of HCF versus MCF is small, so exploring alternative heuristics could lead to still more promising approaches. Our working hypothesis, though, is that for this type of tightly connected constraint network with randomly allocated costs, no cost-sensitive heuristic will do much better than the best cost-insensitive heuristic.

The Sports Scheduling CSP corresponds to a scheduling problem where times and locations must be assigned to a collection of games among N teams. There are multiple stadiums (locations) in which the games can be played. The set of games to be scheduled is generated either by having all teams play all other teams or randomly between the teams. Here, the costs are generated by drawing an analogy to a natural source of costs – namely, checking on the availability of locations. We associate a constraint value cost with checking a stadium's availability at a specified time. These costs are drawn from different probability distributions. The sports scheduling problem provides a more realistic model for the use of costs than N-queens. It has a less regular constraint network than N-queens, and associating costs with a particular subset of the constraints seems to correspond better to cost distributions in the real world.

The heuristics used in the sports scheduling problem were:

- Random value-ordering: when instantiating each game variable, the stadium-time pair is chosen randomly.
- Lowest Cost First (LCF): analogous to the LCF heuristic in N-queens. However, in this case, since the costs are associated with constraint values rather than constraints, the heuristic is a value-ordering heuristic. It is essentially a "greedy" approach to constraint checking in the presence of costs.
- Most Constrained Team First (MCF): selects the team based on the weighted average of the constraints of all the games it will play in.

In this problem, costs were assigned to each stadium-time pair separately randomly. The results showed that the LCF heuristic is substantially more cost effective than the other heuristics. This is a natural and expected result, since we are taking costs into consideration and taking measures to reduce the overall cost. The fact that LCF is so effective here, but doesn't perform especially well in the N-queens domain, is due–at least in part–to the less connected nature of the constraint network, and the more natural (less random) distribution of costs. We built a general hypothesis based on these results that for this domain, given the current design, no variable-ordering heuristic will perform much better than the LCF value-ordering heuristic.

Since we have only looked at a relatively simple variant of the scheduling problem, and only three very straightforward heuristics , we cannot draw strong conclusions about the best heuristic for such domains. More complex domains can be explored by adding temporal and preferential constraints on the teams and the games, and by developing and applying additional heuristics. The preliminary results do reinforce our belief that specialized, cost-sensitive heuristics will be of great benefit in solving constraint-checking costs.

These cost-sensitive methods will enable two extensions to the VTIE scheduler: first, the user-interaction cost of rescheduling can be explicitly modeled; second, a tighter integration with the identification of available time slots is possible. For example, initially the telescopes could be polled for their availability over the next two months. If there were requests that could not be satisfied by the resulting time slots, another query could be issued to all or some of the telescopes. The cost of gathering this information will be modeled explicitly and incorporated into the search process.

Dynamic Rescheduling

The final extension will be to enable rescheduling in the face of a dynamically changing situation. For example, the weather forecast could indicate that visibility will be unsatisfactory; a telescope could break or require unscheduled maintenance; or a scheduled observation could be preempted by a higher-priority request. In this case, as in the

case of incremental scheduling, the goal is to rapidly reassign the affected requests in a way that minimally impacts other scheduled requests. The costs are different than in incremental scheduling, because of the rapid turnaround time for modifications. In incremental scheduling, the cost associated with rescheduling a request is that of communicating the decision to the user whose request was replaced. On the other hand, when a request has to be rescheduled due to an unexpected event, there is an element of urgency in scheduling that request. The affected request should ideally be scheduled on a different telescope in the same time slot. This can be done easily if the schedule is not tightly packed already. There is a trade-off to be obtained between high resource utilization, which is achieved by having a packed schedule, and low cost of rescheduling. In the absence of vacant slots, the cost is increased because some other request will need to be replaced which must also be scheduled as soon as possible.

The incremental and cost-sensitive scheduling methods described previously will directly support a dynamic rescheduling system. The incremental scheduling methods can be used to identify and evaluate alternative slots for an affected request. Cost-sensitive scheduling allows this process to minimize the amount of ripple effect that results from rescheduling the affected requests.

Summary

We have begun to implement an automated constraint-based scheduling system for VTIE. The current implementation provides batch and incremental scheduling capabilities, using a greedy algorithm combined with a simple iterative repair technique to assign telescope slots to student requests. Future versions of the scheduler will provide cost-sensitive constraint checking and dynamic rescheduling in response to changing weather and telescope conditions.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 0121531. We also received support in the form of a fellowship from the GEST Center at UMBC.

References

Barták, R. 1999. Constraint programming: In pursuit of the holy grail. (Invited lecture).

Bresina, J.; Edgington, W.; Swanson, K.; and Drummond, M. 1996. Operational closed-loop observation scheduling and execution. In *Proceedings of the AAAI Fall Symposium on Plan Execution: Problems and Issues*. Available as AAAI Technical Report FS-96-01.

Haralick, R. M., and Elliot, G. L. 1980. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence* 14:263–314.

Johnston, M. D., and Miller, G. E. 1994. SPIKE: Intelligent scheduling of Hubble Space Telescope observations. In *Intelligent Scheduling*. Morgan Kaufmann. 391–422.

Kumar, V. 1992. Algorithms for constraint-satisfaction problems: A survey. *AI Magazine* 13(1):32–44.

Le Pape, C. 1994. Scheduling as intelligent control of decision making and constraint propagation. In *Intelligent Scheduling*. Morgan Kaufmann. 67–98.

Sansare, S. 2002. Incorporating constraint checking costs in constraint satisfaction problems. M.S. thesis.

Smith, S. F. 1994. OPIS: A methodology and architecture for reactive scheduling. In *Intelligent Scheduling*. Morgan Kaufmann. 29–66.