Effective Redundant Constraints for Online Scheduling

Lise Getoor
CS Dept
Stanford University
Stanford, CA 94305-9010
getoor@cs.stanford.edu

Greger Ottosson
CS Dept
Uppsala University
Box 311, S-751 05
Uppsala, Sweden
greger@csd.uu.se

Markus Fromherz
Xerox PARC
3333 Coyote Hill Road
Palo Alto, CA 94304
fromherz@parc.xerox.com

Björn Carlson
Xerox PARC
3333 Coyote Hill Road
Palo Alto, CA 94304
bcarlson@parc.xerox.com

Abstract

The use of heuristics as a means to improve constraint solver performance has been researched widely. However, most work has been on problem-independent heuristics (e.g., variable and value ordering), and has focused on offline problems (e.g., one-shot constraint satisfaction). In this paper, we present an online scheduling problem for which we are developing a real-time scheduling algorithm. While we can and do use generic heuristics in the scheduler, here we focus on the use of domain-specific redundant constraints to effectively approximate optimal offline solutions. We present a taxonomy of redundant domain constraints, and examine their impact on the effectiveness of the scheduler. We also describe several techniques for generating redundant constraints, which can be applied to a large class of job shop scheduling problems.

Introduction

Scheduling is the task of allocating resources to jobs such that the jobs can be executed in a correct and timely manner. In this paper, we focus on *online scheduling*. In the online problem, the problem input is fed incrementally to the scheduler, and the scheduler has only a portion of the entire job available before it has to start making scheduling decisions. In fact, job submission, scheduling and execution may all happen in parallel and at different speeds. Generally, constraint programming (Van Hentenryck 1989) and the large body of experience with constraint-based scheduling (Zweben & Fox 1994) provide a well-suited foundation for online scheduling. Online scheduling is also a special case of dynamic scheduling and dynamic constraint satisfaction problems (CSPs) (Dechter & Dechter 1988); however in the online case, constraints may be added, but are never removed.

We investigate constraint-based online scheduling in the domain of reprographic machines (photo-copiers, printers, and fax machines). The task of a generic reprographic machine scheduler is to schedule the operations that produce a given document. In particular, we study optimizing the make-span of the schedule, i.e., minimizing the output time of the last sheet. The make-span is usually taken as a measure of a machine's productivity. For high-end machines, productivity improvement often translates proportionally to an increase in perceived value. For expensive machines, even small improvements (e.g., by 5-10%) are significant.

Optimizing overall productivity is difficult in an online environment, and even more so with a real-time algorithm. Not only is it difficult to choose the right schedule without complete job information, but there may not be enough time to find the best schedule. This motivates our work on the heuristics presented in this paper. The use of heuristics as a means to improve solver performance has been researched extensively. Most of this work has been on problem-independent heuristics (e.g., variable and value ordering) that make use of certain properties (e.g., constraint tightness) of the constraint network (Dechter & Meiri 1994; Johnston & Minton 1994; Cheng & Smith 1995). More recent work has focussed on the elimination of symmetries (Crawford *et al.* 1996). We want to build on these improvements with the use of problem-specific heuristics, in particular redundant constraints. While most work on redundant constraints has focused on removing inconsistencies in offline CSPs (e.g., (Dechter & Dechter 1987)), we apply redundant constraints to remove sub-optimal solutions in an optimization problem. Here, we investigate how different classes of redundant constraints improve the performance of an online scheduling algorithm. While there has been research on solution reuse (i.e., heuristics at the solver level) in dynamic CSPs (Schiex & Verfaillie 1993; Van Hentenryck & Le Provost 1991), we are not aware of prior work on redundant constraints (i.e., heuristics at the problem level) for online scheduling.

This paper is structured as follows. We first present our scheduling problem together with a machine model. We then describe different classes of redundant constraints, and how they can be used in a constraint-based online scheduler. We present our empirical results evaluating the impact of these redundant constraints. We conclude with a discussion

[1] Copyright ©1997, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

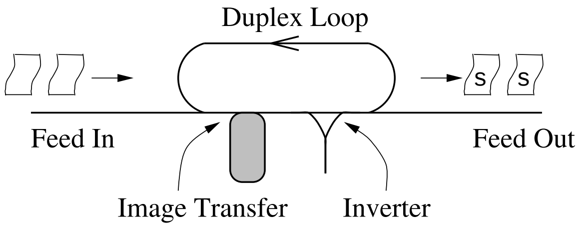


Figure 1: Schematic view of a simple reprographic machine

of the generality of our results and an outline of future work.

Reprographic Machine Scheduling

The task of scheduling networked reprographic machines is defined as follows. A scheduler is given a *machine description* and a finite *sequence of output sheets* that are to be printed by the machine. The machine description is formulated in terms of constraints on sheets. The scheduler's task is to determine when the operations necessary to feed and print sheets have to be performed such that the output sequence is produced.

For this paper, we assume that the machine has one input and one output (Fig. 1). In between, sheets are continuously moving, and once a sheet is fed into the machine, its itinerary through the machine is fixed. Thus, *the scheduler's task is to determine the timed sheet input sequence for a given output sequence*. In the following, we will discuss a simplified but complete form of the problem of scheduling reprographic machines. While high-end machines have more complex constraints, supporting color, buffering and multiple sheet sizes, this simpler version already represents a realistic and non-trivial machine.

Basic Machine Model

In this model of a simple machine (Fig. 1), we are given a configuration with an image transfer component, an inverter component, and a paper path that leads from the input to the image transfer, the inverter component, and then either out or back to the image transfer. A single-sided sheet is printed once and moved to the output without inversion. A double-sided sheet is printed on one side, then inverted and moved back to be printed on the other side, before it is inverted again and moved to the output. Let k be the time difference between first and second printing of a double-sided sheet; it corresponds to the length of the duplex loop.

The requested output sequence $O = o_1, \ldots, o_n$ is an ordered sequence of n single- and double-sided sheets ("`s`" and "`d`" for short) specified by the user. The schedule to be produced by the scheduler is the explicitly timed sequence $S^E = x_1, \ldots, x_n$, where x_i denotes the input time of sheet o_i and ranges over points in time. We also keep track of the time when the last image of sheet o_i is transferred. Let y_i be that time: $y_i = x_i$ if $o_i =$ s, and $y_i = x_i + k$ if $o_i =$ d.

As a simplification, we can assume that all sheets have to be aligned to a period p (corresponding to the sheet length in seconds), i.e., all x_i are multiples of p. Without loss of generality, we can choose $p = 1$. Such a schedule can conveniently be shown in an implicitly timed representation S^I, where element i in the sequence represents what is being printed at time i. The alphabet for this representation is {`s`, `f`, `b`, `_`}, where `s`, `f` and `b` denote single, front and back pages, and "`_`" denotes a blank spot. For example, given $O =$ `dds` and $k = 5$, a correct schedule would be $S^E = (1, 2, 9)$, respectively $S^I =$ `ff___bb_s` (i.e. the pages of o_1 are printed at times 1 and 6, the pages of o_2 are printed at times 2 and 7, and the page of o_3, at time 9).

Given O, the scheduler's task is to find an S such that x_n is minimal and the following constraints are satisfied:

Order constraint. The order of O has to be preserved by S:

$$\forall i = 1, \ldots, n-1 \;.\; y_i < y_{i+1}$$

Loop constraint. A sheet returning through the duplex loop must not collide with an incoming sheet:

$$\forall i = 1, \ldots, n \;.\; \forall j (1 \leq j < i) \;.\; o_j = \text{d} \rightarrow x_i \neq x_j + k$$

Note that for $O =$ `sdd` and $k = 5$, both $S =$ `sff___bb` and $S =$ `ff_s_bb` are valid schedules.

Inversion constraint. The inversion of a sheet takes p seconds, i.e., it is proportional to the length of the sheet, while bypassing the inverter takes no time. Therefore, an inverted sheet (either front or back side of a double-sided sheet) cannot directly be followed by a non-inverted sheet (or else they would jam):

$$\begin{aligned} &\forall i = 1, \ldots, n \;. \\ &\quad (o_i = \text{d} \rightarrow \forall j (\max(i-k, 1) \leq j < i) \;.\; o_j = \text{s} \rightarrow \\ &\qquad x_i \neq x_j - 1) \\ &\quad \wedge\; ((o_i = \text{s} \wedge i > 1) \rightarrow o_{i-1} = \text{d} \rightarrow \\ &\qquad x_i \neq x_{i-1} + k + 1) \end{aligned}$$

So, for $O =$ `sdd` and $k = 5$, $S =$ `ff_s_bb` is a valid schedule, while $S =$ `ffs__bb` is not.

Properties of the Basic Model

One of the properties that makes scheduling for certain machine models in this domain tractable is that the make-span for the activities of individual sheets is bounded: because of the strong output ordering requirements of this domain, and the fixed paper path per sheet type in the basic models, there are a limited number of plausible timings for the sheets. This is a very desirable domain property that allows us to impose quite strong local constraints on the placement of individual sheets in the job in the schedule. Note, however, that a scheduling decision for a sheet does impact the scheduling decisions for sheets arbitrarily far away from it in the input; thus there is no simple greedy algorithm that is

Document O	`sddd`
Naive S	`sf____bf____bf____b`
Greedy S	`sfff__bbb`
Optimal S	`fff_sbbb`

Figure 2: Alternative schedules S for the same document O

guaranteed to find the optimal solution (Fromherz & Carlson 1994). However, scheduling for the basic machine is not NP-hard; there is a pseudo-polynomial time algorithm for finding the optimal schedule (Fromherz & Carlson 1994).

Online Scheduling

As mentioned, it is usually not possible to wait until the scheduler has received a complete job and generated a schedule for it. Instead, scheduling and schedule execution typically happen in parallel. At discrete time instants (p seconds with the above machine model), the scheduler is interrupted and asked if a sheet is to be fed (and which). A typical value for p is 1. The scheduler typically gets a few tenths of that time for its processing.

Fig. 2 shows different schedules that could be generated for a given job, using either a naive, greedy, or optimal (look-ahead) algorithm. In the online context, it has been shown that a good strategy leading to overall optimal productivity for the type of machine shown above is *full optimization with minimal commitment* (Fromherz & Carlson 1994): whenever the scheduler is interrupted, generate an optimal schedule for the job known at that time, but commit only to that part of the schedule that has to be returned for execution. In our experience, the schedules produced for randomly generated jobs are 5% worse than optimal on the average.

We thus take as a premise for the rest of this paper that it is desirable to generate an optimal schedule for the currently known job whenever the scheduler is interrupted. This provides the motivation for investigating heuristics that enable the efficient approximation of optimal schedules.

Redundant Constraints for Online Scheduling

The constraints defined by the above machine model are sufficient to produce feasible solutions to our scheduling problem. Here, in order to improve the time to generate an optimal schedule, we investigate categories of *redundant* domain constraints.

Given a constraint set C, a new constraint c is redundant if it is logically entailed by the constraints in C, i.e., $C \vdash c$. Thus, c can be added to C without removing any feasible solutions for C. For constraint optimization problems, we define a constraint as redundant if it can be added to the problem without removing any optimal solutions. We are of course interested in redundant constraints that remove sub-optimal solutions or at least large parts of the search space.

A Taxonomy of Redundant Constraints

For our problem, we first distinguish two categories of redundant constraints: those that constrain the placement of individual sheets relative to the beginning of the schedule, and those that constrain sets (typically pairs) of neighboring sheets relative to each other. We call the former *global constraints* and the latter *local constraints*. In addition, since we are operating in an online environment where we may have solutions to prefixes of the current job, we also distinguish constraints that make use of previously computed information. We call these *incremental (global) constraints*. The scopes of these three classes of redundant constraints are depicted schematically in Fig. 3.

These constraints can further be either *job-independent* or *job-specific*. Job-specific constraints make use of job characteristics such as the number or density of particular sheet types, while job-independent constraints don't use any information about the particular job. The distinction is of interest since job-independent redundant constraints are easier to find, but job-specific constraints tend to more tightly constrain variable domains.

Finding Redundant Constraints

Scheduling reprographic machines as described above is related to the flow shop scheduling problem, where jobs consist of activities that can be executed on one or more resources, and the order of activities and jobs is given (Blazewicz *et al.* 1993). Our problem differs in that there is usually a single resource per activity, there are no buffers between resources, and we allow for loops in the execution path. Concretely, this is akin to assembly-line scheduling problems: each sheet in our a model is a "job", consisting of activities like "print front", "print back", etc., and jobs move from resource to resource to be processed.

We have identified a set of techniques for finding redundant constraints that are generally applicable to this kind of online scheduling problem. We can describe these techniques only very briefly.

Given the problem and our constraint solving approach, constraints on lower and upper bounds of individual jobs (aka sheets) promise a good trade-off between pruning in search and overhead in propagation. Necessarily, the redundant constraints that we construct often compile in a set of selected constraints together with propagation. The simplest way to find *lower-bound global constraints* for individual jobs is to relax certain constraints, and identify the "shortest possible schedule" for any job. (For example, use only order constraints, and pretend each sheet is single-sided.) A way to find *upper-bound global constraints* on the

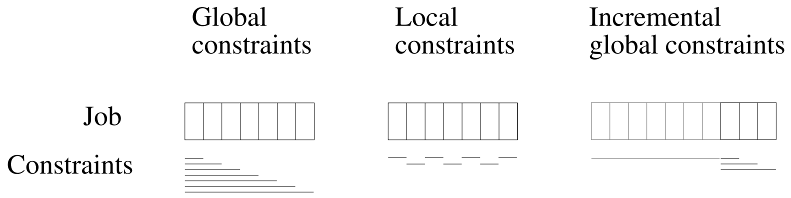


Figure 3: Schematic view of the scopes of three classes of redundant constraints

last job (make-span constraints) is to proceed in the same way, identifying the "longest possible schedule" for any job. (For example, pretend each sheet is double-sided.)

A central concept for the latter is the *local upper bound*. This bound is the maximum distance that two consecutive jobs can be scheduled apart such that the complete schedule will still be optimal. This bound exists because there are no buffers in the system, and because of the predetermined job order, and thus a larger distance between jobs would lead to unusable idle time between them. (In our model, the local upper bound is k.)

The above approach to find global bounds can also be used to find *job-specific constraints* by identifying a high-level characterization of the jobs (and their activities) and using this to construct more precise bounds. (For example, distinguish between single- and double-sided sheets, and summarize over each class separately instead of the entire document.) In general this may be difficult; it is more likely to be possible in cases with a limited number of types of activities. An area for future research is the automatic construction or learning of appropriate high-level characterizations.

Local redundant constraints are found through an analysis of the interactions between different types of jobs, while *incremental constraints* can be derived by generalizing global constraints and reasoning about the interaction between previous schedule and new jobs. In both cases, the local upper bound is a crucial element of that reasoning.

Note that while the constraint-relaxation method described first can find lower-bound constraints on all jobs (in part because of the order constraint), it can only find upper-bound constraints on the last job. This is because in an optimal schedule, only the last job is actually scheduled optimally. However, we have developed two techniques for finding upper-bound constraints on all jobs by combining make-span upper-bound constraints with other constraints. One technique generates an individual upper-bound constraint by "subtracting" an appropriately adapted individual lower-bound constraint from a make-span upper-bound constraint. The other technique generates an individual upper-bound constraint by adding the local upper bound to a make-span upper-bound constraint and applying the result to individual jobs.

Redundant Constraints for the Basic Model

In this subsection, we present examples for the different classes of redundant constraints for the basic model. Note that we assume that schedules start at time 1. All constraints are on variables $y_j, j = 1, \ldots, n$, where y_j is the time of the last image transfer for sheet j, and n is the number of sheets in the job (see above).

Global Constraints. Global constraints set lower and upper bounds on each sheet with respect to the start of the schedule.

Job-independent constraints (GJI). We can impose the following weak, job-independent global constraints on lower and upper bounds of each variable:

$\forall j = 1, \ldots, n \;.\; y_j \geq j$

$\forall j = 1, \ldots, n \;.\; y_j \leq 2j + jk$

Job-specific constraints (GJS). Job-specific global constraints can be imposed by studying certain characteristics of the job. Given a sheet index j, let C_s be the number of single-sided sheets in O, C_d the number of double-sided sheets in O, and C_{ds} the number of switches from double- to single-sided sheets in O, each time up to and including sheet j. Then we can impose the following job-specific lower and upper bounds:

$\forall j = 1, \ldots, n \;.\; y_j \geq C_s + 2C_d$

$\forall j = 1, \ldots, n \;.\; y_j \leq C_s + 2C_d + kC_{ds} + k - 1$

Local Constraints. The following local constraints relate pairs of neighboring sheets to each other.

Job-independent constraints (LJI). We do not consider local job-independent constraints, as they turn out to exactly correspond to the constraints imposed by the machine model.

Job-specific constraints (LJS). We found that the following job-specific constraints were useful. They come from a simple analysis of how specific combinations of single- and double-sided sheets must be placed to be part of an optimal schedule, using the local upper bound. The lower bounds are identical to the order and the inversion constraints, so they are not shown.

We have the following upper bounds:

$\forall j = 2, \ldots, n-1 \;.$

$(o_j = \mathrm{s}) \rightarrow y_j \leq y_{j-1} + k$

$(o_j = \mathrm{d}) \rightarrow y_j \leq y_{j-1} + k + 1$

Since we know that the last sheet (in a specific optimization) is not followed by other sheets, we can impose stronger upper bounds on it:

$$(o_n = \mathrm{s} \wedge o_{n-1} = \mathrm{s}) \rightarrow y_n \leq y_{n-1} + 1$$
$$(o_n = \mathrm{s} \wedge o_{n-1} = \mathrm{d}) \rightarrow y_n \leq y_{n-1} + 2$$
$$(o_n = \mathrm{d}) \rightarrow y_n \leq y_{n-1} + k + 1$$

Note that this last constraint is local to this optimization and must be retracted when more sheets are added to the job.

Incremental Global Constraints. The following incremental constraints build on an optimal solution for a prefix of the current problem. We could consider incremental versions of both global and local constraints. Incremental local constraints are equivalent to local constraints, though. In contrast, incremental global constraints become a useful generalization of global constraints (by subtracting (respectively adding) the local upper bound, k).

In the following, let i be the number of sheets that were scheduled previously, and y_i^* the value of y_i in that previous schedule.

Job-independent constraints (IJI). The following constraints provide job-independent lower and upper bounds:

$$\forall j = i+1, \ldots, n \,.\, y_j \geq y_i^* + j - i - k$$
$$\forall j = i+1, \ldots, n \,.\, y_j \leq y_i^* + 2(j-i) + (j-i)k$$

Job-specific constraints (IJS). Job-specific lower and upper bounds are given by the following constraints (cf. the global constraints above):

$$\forall j = i+1, \ldots, n \,.\, y_j \geq y_i^* + C_s + 2C_d - k$$
$$\forall j = i+1, \ldots, n \,.\, y_j \leq y_i^* + C_s + 2C_d + kC_{ds} + 2k - 1$$

Experiments

In our experiments, we use the following search and propagation mechanisms for doing full optimization. For each interrupt, we initiate a restarting branch-and-bound search (Van Hentenryck 1989) for the known scheduling problem, in which we use straight-forward chronological backtracking to find incrementally better solutions. Constraint propagation is done in an algorithm reminiscent of a generalized AC-3, but maintaining *interval consistency* (also known as *arc-B-consistency* (Lhomme 1993)) rather than arc-consistency. We also make use of both job- and machine-independent search control heuristics in our experiments. Well-known examples are variable labeling in the order of constraint tightness, enumeration with domain splitting, and reuse of variable assignments in repeated searches (Van Hentenryck & Le Provost 1991). For these results, we use naive variable and value ordering. These techniques are now commonly available in state-of-the-art constraint solvers (e.g., (Carlsson 1995)).

To evaluate the impact of each class of constraints, we benchmark their performance on a set of 20 randomly generated documents. The length of the documents averages 20 sheets, and the requests come at intervals with 4 new sheets per request, thus the results are the average total scheduling times for 100 requests.

	First Solution		Optimal Solution	
Constraints	Find	Qual.	Find	Prove
NONE	113.10	15%	42416.90	61625.95
GJI	97.38	15%	42317.86	61404.29
GJS	94.52	15%	38827.38	54524.29
LJS	170.00	15%	5160.24	7524.29
IJI	83.57	14%	4851.90	9453.10
IJS	194.05	12%	1663.81	3408.10
ALL	161.43	12%	1200.71	1916.43

Figure 4: Constraint classes compared for a set of randomly generated documents. Time in milliseconds, collected on a SPARCstation-10.

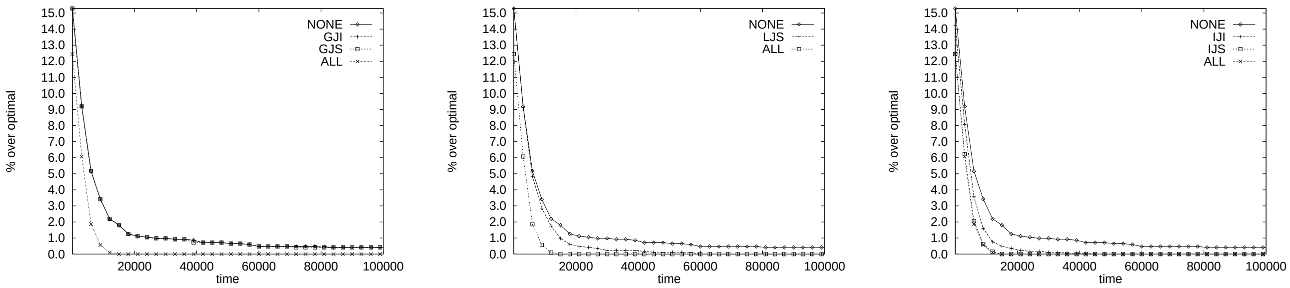
The results are depicted in Fig. 4. Column 1 shows the average time to find the first solution, column 2 shows the average quality of the first solution measured as the percentage over optimal solution length, and columns 3 and 4 shows the average time to find and prove the optimal solution, respectively. Fig. 5 summarizes these results; it shows the scheduler's anytime profile, i.e., the graphs of the solution quality at various times for each constraint class. As can be seen, the incremental constraints have the most effect. More specifically, IJS leads to the greatest improvement in optimization time, while IJI is best for finding the first solution.

This illustrates the classic trade-off for flexible algorithms: Is it of prime importance to find the first solution fast, or is our goal set on finding the optimal solution? While redundant constraints generally lead to a better first solution, adding all of them can result in significant overhead. Also, the choice of constraint class will depend on how much time our solver has available at each request: if our time is quite limited, we may choose to use a class such as IJI only; if we have more time available, we would certainly include IJS (and perhaps just add ALL). This shows that it is important to analyze both the constraint classes and the application environment in order to make the appropriate choices.

Conclusions and Future Work

In this paper we investigate how to exploit domain knowledge to improve optimization performance in an online scheduling problem. We present a taxonomy of redundant constraints in terms of *global*, *local* and *incremental* constraints on one axis, and *job-specific* and *job-independent* on the other. We evaluate the performance of redundant constraints from the cross product of these classes, and discuss their impact. We conclude from these tests that adding domain-specific redundant constraints can have a significant impact on optimization performance.

It is our belief that this classification and the examples of redundant constraints are sufficiently general to be useful for a range of scheduling problems. The global constraints,



a) b) c)

Figure 5: Anytime profile for a) Global Constraints b) Local Constraints and c) Incremental Constraints.

which constrain the end of the schedule to the start of the schedule, arise naturally in most scheduling problems. The incremental constraints, which reuse previously made partial optimizations, effectively exploit the online property. The local constraints that bind neighboring tasks together work well for scheduling problems where there are tight precedence constraints. In general, this might not always be the case, but local constraints could be applied for subsets of a problem, e.g., for an independent subset of tasks in a job-shop scheduling problem.

One of the most challenging tasks is finding the domain-specific constraints; the automatic generation and verification of such constraints will be an important next step in our research. Also, constraint propagation and redundant constraints can interact, and it would be interesting to study the effectiveness of redundant constraints under varying levels of propagation. Further paths for future work are to apply our work to further types of machines, and to try classifying redundant domain constraints for other domains and problems based on our taxonomy.

Acknowledgments

The work reported in this paper is part of the Machine Control Project at Xerox PARC. We gratefully acknowledge the collaboration with former and current members of this project, including Vijay Saraswat, Rajeev Motwani, Moses Charikar and Danny Bobrow. We would also like to thank Jeremy Frank, Ari Jonsson and our anonymous reviewers for comments on this paper.

References

Blazewicz, J.; Ecker, K.; Schmidt, G.; and Weglarz, J. 1993. *Scheduling in Computer and Manufacturing Systems*. Springer-Verlag.

Carlsson, M. 1995. *SICStus Prolog User's Manual*. Swedish Institute of Computer Science. Release 3.

Cheng, C., and Smith, S. 1995. Applying constraint satisfaction techniques to job shop scheduling. Technical Report CMU-R1-TR-95-03, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Crawford, J. M.; Ginsberg, M.; Luks, E.; and Roy, A. 1996. Symmetry breaking predicates for search problems. In *Proc. of KR'96*, 148–159.

Dechter, A., and Dechter, R. 1987. Removing redundancies in constraint networks. In *Proc. of AAAI'87*, 105–109.

Dechter, R., and Dechter, A. 1988. Belief maintenance in dynamic constraint networks. In *Proc. of AAAI'88*, 37–42.

Dechter, R., and Meiri, I. 1994. Experimental evaluation of preprocessing algorithms for constraint satisfaction problems. *Artificial Intelligence* 68:211–241.

Fromherz, M., and Carlson, B. 1994. Optimal incremental and anytime scheduling. In Lim, P., and Jourdan, J., eds., *Proc. Workshop on Constraint Languages/Systems and their Use in Problem Modeling at ILPS'94*, 45–59. TR 94-38.

Johnston, M., and Minton, S. 1994. Analyzing a heuristic strategy for constraint-satisfaction and scheduling. In *(Zweben & Fox 1994)*, 257–290.

Lhomme, O. 1993. Consistency techniques for numeric CSPs. In *Proc. 13th International Joint Conference on Artificial Intelligence (IJCAI-93)*, 232–238.

Schiex, T., and Verfaillie, G. 1993. No-good recording for static and dynamic constraint satisfaction problems. In *Proc. of the Fifth IEEE Int. Conf. on Tools with Artificial Intelligence*.

Van Hentenryck, P., and Le Provost, T. 1991. Incremental search in constraint logic programming. *New Generation Computing* (9):257–275.

Van Hentenryck, P. 1989. *Constraint Satisfaction in Logic Programming*. Logic Programming Series. Cambridge, MA: MIT Press.

Zweben, M., and Fox, M. S., eds. 1994. *Intelligent Scheduling*. San Francisco, Calif.: MK.