This Is a Publication of
The American Association for Artificial Intelligence

This electronic document has been retrieved from the
American Association for Artificial Intelligence
445 Burgess Drive
Menlo Park, California 94025
(415) 328-3123
(415) 321-4457
info@aaai.org
http://www.aaai.org

(For membership information,
consult our web page)

The material herein is copyrighted material. It may not be
reproduced in any form by any electronic or
mechanical means (including photocopying, recording,
or information storage and retrieval) without permission
in writing from AAAI.

A Generic Framework for Constraint-Directed Search and Scheduling

J. Christopher Beck and Mark S. Fox

■ This article introduces a generic framework for constraint-directed search. The research literature in constraint-directed scheduling is placed within the framework both to provide insight into, and examples of, the framework and to allow a new perspective on the scheduling literature. We show how a number of algorithms from the constraint-directed–scheduling research can be conceptualized within the framework. This conceptualization allows us to identify and compare variations of components of our framework and provides new perspective on open research issues. We discuss the prospects for an overall comparison of scheduling strategies and show that firm conclusions vis-a-vis such a comparison are not supported by the literature. Our principal conclusion is the need for an empirical model of both the characteristics of scheduling problems and the solution techniques themselves. Our framework is offered as a tool for the development of such an understanding of constraint-directed scheduling and, more generally, constraint-directed search.

Constraint-directed search is a powerful search methodology founded on the use of a rich constraint representation not only to model the problem knowledge but also to guide search to a solution. It is the latter point that is responsible for much of the power of the technique: Constraints do not simply represent the problem but are also active in narrowing the space of possible solutions and form an integral basis for heuristic decision making.

The goals of this article are to introduce a generic framework for constraint-directed search; illustrate the framework with the constraint-directed scheduling literature; and, finally, place constraint-directed–scheduling research within the framework, enabling a new perspective on various scheduling algorithms and components thereof.

Constraint-Directed Search

Constraint-directed search, broadly defined, is an approach to problem solving that explores the problem space under the guidance of the relationships, limitations, and dependencies among problem objects. These relationships, limitations, and dependencies together are known as *constraints*. The approach requires that these constraints are represented and, furthermore, are represented in a way that search techniques can make use of them for guidance.

We return to such representation and search issues later, after providing the definition of the most basic example of a constraint-directed search problem in the following section.

The Constraint-Satisfaction Problem

The finite constraint-satisfaction problem (CSP) (Tsang 1993; Mackworth 1977) consists of the following: Given are a set of n variables $Z = \{x_1, \ldots, x_n\}$ with discrete, finite domains $D = \{D_1, \ldots, D_n\}$ and a set of m constraints $C = \{c_1, \ldots, c_m\}$ that are predicates $c_k(x_i, \ldots, x_j)$ defined on the Cartesian product D_i x … x D_j. If c_k is true, the valuation of the variables is said to be consistent with respect to c_k, or equivalently, c_k is satisfied. A *solution* is an assignment of a value to each variable, from its respective domain, such that all constraints are satisfied.

An instance of a CSP (Z, D, C) can be conceptualized as a constraint graph, $G = \{V, E\}$. For every variable v in Z, there is a corresponding node n in V. For every set of variables connected by a constraint c in C, there is a corresponding hyperedge e in E. Other conceptualizations of a CSP exist, including the dual constraint graph and join graph (Dechter, Dechter, and Pearl 1990). For binary constraint problems, where each constraint involves at most two variables, the hyperedges are simply edges.

In figure 1, we present a constraint graph of

Copyright © 1998, American Association for Artificial Intelligence. All rights reserved. 0738-4602-1998 / $2.00

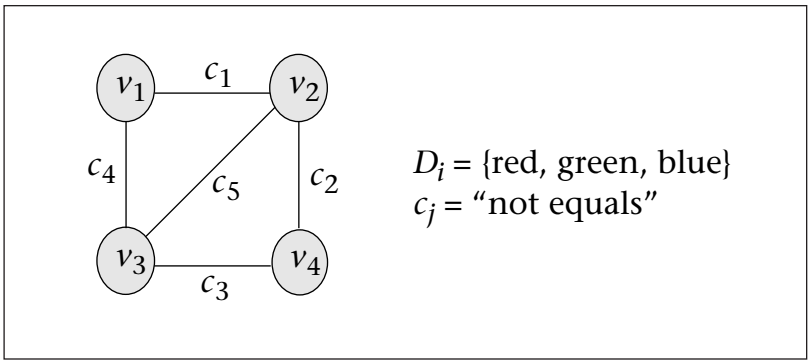

Figure 1. A Small Graph Coloring Problem Represented as a Constraint-Satisfaction Problem.

a CSP representation of a small graph coloring problem. Each variable (node) has a domain of three values {red, green, blue}, and each constraint (edge) expresses a "not equals" relationship. For this particular example, there are a number of solutions, including {v_1 = red, v_2 = green, v_3 = blue, v_4 = red}.

The search for a solution to a CSP can be viewed as modifying the constraint graph by addition and removal of constraints. The constraint graph is an evolving representation of the search state, where a solution is a state with a single value remaining in the domain of each variable, and all constraints are satisfied. In our example in figure 1, we can (heuristically) add a unary constraint that assigns v_2 = green. Alternatively, we may have at our disposal an algorithm that is able to infer that v_1 and v_4 must have the same color, and therefore, we introduce a binary "equals" constraint between these two variables.

CSPs have successfully been used for a wide range of problems from the abstract (for example, graph coloring [Minton et al. 1992]) to the concrete (for example, machine vision [Shapiro and Haralick 1981; Rosenfeld, Hummel, and Zucker 1976; Waltz 1975]). For an excellent review of CSP solution techniques and applications, see Kumar (1992).

A *constraint-optimization problem* (COP) (Tsang 1993) is defined as a CSP together with an optimization function f that maps every solution tuple to a numeric value: (Z, D, C, f), where (Z, D, C) is a CSP, and if S is the set of solution tuples of (Z, D, C), then $f : S \rightarrow$ the numeric value. The task in a COP is to find a solution tuple with the optimal (minimal or maximal) value of f. A common variation of this model is one in which the optimization function is a weighted sum of the constraints violated by a particular valuation of the variables (Zweben et al. 1994; Zweben et al. 1993; Smith et al. 1989; Fox 1983). Rather than satisfy all constraints and optimize f, the goal is to minimize the cost by satisfying as many constraints as possible.

Why Constraints?

Constraint-directed search relies on two interdependent intuitions: The *representational intuition* states that to solve a problem, the relevant knowledge must be represented. The *search intuition* states that search toward a solution should be guided based on the represented knowledge. Underlying these intuitions is the following topological assumption (Fox 1986): Understanding a problem's search space will enable the creation and selection of search techniques that can efficiently navigate the space to a solution. Fortunately, the topological assumption is not too outrageous given that it is underlies much of AI and operations research.

The constraint-directed interpretation of the representational intuition calls for the creation of a rich constraint representation able to express the problem knowledge at a deep level. The search intuition suggests that we look to the constraints for search guidance: Constraints are not passive objects that evaluate a potential solution but form the basis for the search operators.

The representational and search intuitions interact. A constraint representation cannot be created wholly independent of the search techniques. Similarly, a set of efficient search techniques cannot be constructed without knowledge of the constraint representation. It is typical to iterate between representation and search: Not only do the search techniques take into account the details of the knowledge representation, but these very knowledge representation details must be modified and extended based on the details of the search techniques.

A danger with this research approach, especially when applied to solving real-world problems as in scheduling, is that the focus might be on a specific set of problems and not on theoretical issues surrounding search, problem characteristics, complexity, and other aspects of computer science. It is critical for both structured progress toward solving specific problems and incorporation of the work of other researchers that the problem characteristics and search techniques be understood within an overall framework. It is important to make this point because given the current scheduling literature, it is difficult to empirically compare algorithms and assess the characteristics of the algorithms and problems leading to performance differences. (A discussion of the development of an empirical theory of heuristics can be found in Hooker [1996, 1993]).

A Framework for Constraint-Directed Search

Before defining the framework and discussing each of the components of the framework, we define a number of terms used in our description of the framework.

Definitions

Induced subgraph: If e is an edge, let $NODE(e)$ be the set of nodes to which e is directly attached. The *induced subgraph* of a set of nodes, V', in a graph $G = \{V, E\}$ is a graph, $G' = \{V', E'\}$, where $V' \subseteq V$ and $E' \subseteq E$ such that $E' = \{e \mid NODE(e) \subseteq V'\}$.

Consistent valuation-assignment: A *consistent valuation* or *consistent assignment* of a subset of CSP variables, A, is the assignment of a value to each variable in A such that all constraints in the induced subgraph of A are satisfied.

Commitment: A *commitment* is a new constraint (or a set of new constraints) added to the constraint graph. The typical example is assigning a start time to an activity by posting a unary equals constraint. We divide commitments into two classes: (1) implied commitments and (2) heuristic commitments. *Implied commitments* are constraints added to the graph because they are implied by the existing problem state. *Implied constraints* are redundant, however; by explicitly adding them to the graph, we are often able to make use of efficient techniques to derive further implied constraints. A heuristic commitment is a constraint that is added, although it is not necessarily implied. Typically, there is some (heuristic) justification to believe that a heuristic commitment is likely to take part in an overall solution.

Propagator: A *propagator* is an algorithm that examines the existing search state to find implied commitments.

Commitment assertion: A *commitment* can be asserted, that is, added to the constraint graph. An *assertion* is a state-transition operator.

Commitment retraction: *Retraction* is the process of identifying a previously made commitment and removing it from the constraint graph. Like assertion, retraction is a state-transition operator.

Termination criteria: The *termination criteria* is the user-defined condition for ending the search. There can be many criteria: a definition of a solution (for example, all the activities have a start time, and all the constraints are satisfied), determination that a solution does not exist, limits on the search in terms of central processing unit (CPU) time, number of commitments, number of heuristic commitments, number of retractions, and so on.

Dead end: A *dead end* is a point in the search where it is discovered that one or more variables cannot be assigned values without breaking some constraints.

Search strategy: A *search strategy* is an instance of our framework, where each component is specified. We allow null specifications of components because, for example, it might be that a strategy uses no propagators or retraction techniques.

The Framework

In proposing a framework for constraint-directed search, we extend and modify the original ODO scheduling framework (Davis 1994) to reflect recent work. Figure 2 displays a schematic of our framework. Figure 3 presents a pseudocode representation. We are not necessarily proposing this framework at the implementation level. Depending on the choices made for the various components, close interaction among components might be required; therefore, an implementation-level description might split or merge the ones we have identified. Nonetheless, we believe that the existing constraint-directed search work can be understood within this framework, and furthermore, that this understanding allows a new perspective on this work.

The Components of the Framework

As indicated by figures 2 and 3, the framework consists of a number of components. The commitment-assertion component is trivial because it requires adding a constraint to the existing graph. The other components (that is, propagators, heuristic-commitment techniques, and retraction techniques) might require significant search effort. In this subsection, we define each of these components more precisely. A number of examples of each component from the constraint-directed scheduling literature are presented.

Propagators A *propagator* is a function that analyzes the current search state to determine constraints that are implied by, but are not explicitly present in, the constraint graph. By making these constraints explicit, we can use them to prune the number of possibilities to be explored in the search space. For example, these constraints can be used to reduce the start-time domains of some of the activities in the problem. The advantages of propagators stem from the soundness of their commitments (a propagator will never infer a constraint that is not a logical consequence of the

The advantages of propagators stem from the soundness of their commitments …

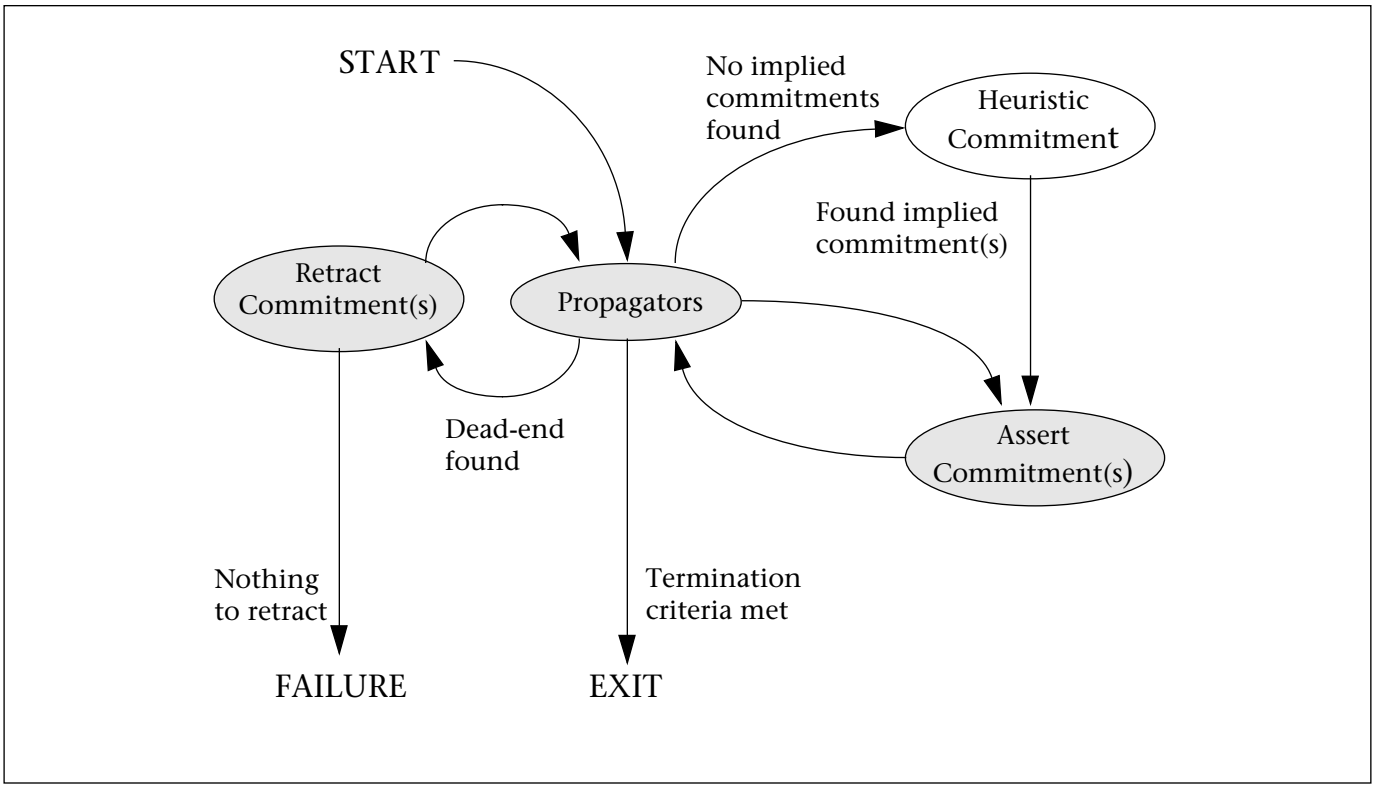

Figure 2. Schematic of the Framework.

```
forever{
    if (termination criteria are met)
        EXIT

    while (untried propagators AND
            no implied commitments found AND
            no dead-end found)
        try next propagator

    if (dead-end found){
        retract commitment(s)
        if (no commitments to retract)
            FAILURE
    }
    else{
        if (no implied commitments)
            make heuristic commitment
        assert commitment
    }
}
```

Figure 3. Pseudocode for a Search Strategy.

current problem state) and the fact that when a constraint is explicitly present in the graph, it not only reduces the search space, but it is also often possible to use additional propagators to further prune the search space.

Examples of propagators from a CSP perspective include the various consistency enforcement algorithms such as arc consistency and *k* consistency (Freuder 1982, 1978; Mackworth 1977). These algorithms are typically viewed as removing values (or tuples of values) from variable domains; however, we treat them as adding implied constraints that, for example, rule out domain values (that is, a unary not-equals constraint).

Many powerful propagation techniques have been developed for constraint-directed scheduling in recent years, for example, many variations of edge finding (Le Pape and Baptiste 1996; Caseau and Laburthe 1994; Nuijten 1994; Carlier and Pinson 1989) and shaving (Carlier and Pinson 1994). It has long been known that search can drastically be reduced by enforcing various degrees of consistency (Freuder 1982). The effort to achieve high degrees of consistency, however, appears to be at least as expensive as more traditional algorithms. The goal for propagator research, then, is to find the trade-off between complex-

ity and the resultant easing of the search effort.

Heuristic-Commitment Techniques Theoretically, we can use propagators by themselves to solve CSPs as used in solution synthesis techniques (Freuder 1978). However, in the worst case, such techniques take time, provably exponential in the number of variables. In practice, to our knowledge, it has not been found to be a worthwhile approach for finding a single solution to a CSP. It is typically necessary to make heuristic commitments—those not necessarily implied by the current problem state.

Traditionally in a CSP, a *heuristic commitment* is the assignment of some value to some variable. With this narrow definition, heuristics focus on variable ordering and value ordering: what is the next variable to assign and to what value will it be assigned. One popular variable-ordering heuristic is the fail-first heuristic that always chooses to instantiate the variable with the fewest number of remaining possible values.

With our more general definition of a heuristic commitment as the addition of a constraint to the graph, we view heuristics as algorithms that, on the basis of measurements of the current search state, suggest new constraints. The traditional variable- and value-ordering heuristics are particular examples of such algorithms.

Retraction Techniques Assume that a search algorithm moves through a sequence of states $S = (s_0, s_1, s_2, \ldots, s_k)$ as a result of the assertion of a number of commitments $A = (a_1, a_2, \ldots, a_k)$. Further, assume that a mistake is made: As a result of one or more of the commitments in A, we have reached a search state s_k that is inconsistent with respect to the constraints in the problem. The state s_k is a dead end in the search. To escape a dead end, we must retract some commitments, $C \in A$. The retraction component of the search strategy must then answer two questions: First, which commitment (or commitments) should be retracted? Second, having selected the to-be-retracted commitment, say, the one made at state s_i, $i < k$, what should be done with the commitments made at the intervening search states, that is, all states s_j, where $i < j < k$? Different retraction techniques supply different answers to these questions. We discuss the various answers applied to constraint-directed scheduling in the next section.

Constraint-Directed Scheduling

Having introduced the framework for constraint-directed search, we now turn to the constraint-directed–scheduling literature. As noted in the introduction, this investigation is to fulfill the dual goals of illustrating the framework components and analyzing the constraint-directed–scheduling research from the perspective of our framework.

Background

Constraint-directed scheduling is the representation of a scheduling problem and the search for a solution to it by focusing on the constraints in the problem. Given that even simple models of scheduling (for example, job-shop scheduling) are NP-hard (Garey and Johnson 1979), the search process often depends on heuristic commitments, propagation of the effects of commitments, and the retraction of commitments in the event of a dead end. Analysis of industrial scheduling problems (Fox 1990, 1983) indicates that the goal is not simply meeting due dates but also satisfying many complex (and interacting) constraints from disparate sources within the organization as a whole. In short, scheduling is a prime application area for constraint-directed search.

… scheduling is a prime application area for constraint-directed search.

The Job-Shop–Scheduling Problem

One of the simplest models of scheduling widely studied in the literature is the job-shop–scheduling problem. The N x M job-shop–scheduling problem is formally defined as follows:

Given are a set of N jobs, each composed of M totally ordered activities and M resources. Each activity A_i requires exclusive use of a single resource R_j for some processing duration dur_i. There are two types of constraint in this problem: Precedence constraints between two activities in the same job state that if activity A is before activity B in the total order, then activity A must execute before activity B. Resource constraints specify that no two activities requiring the same resource can execute at the same time.

Jobs have *release dates* (the time after which the activities in the job can be executed) and *due dates* (the time by which the last activity in the job must finish). In the decision problem, the release date of each job is 0, and a global due date is D. The problem is to determine whether there is an assignment of a start time to each activity such that the constraints are satisfied, and the maximum finish time of all jobs is less than or equal to D. This problem is NP-complete (Garey and Johnson 1979).

An example of a 3 x 5 job-shop–scheduling problem is shown in figure 4. In this example, there are three jobs (A, B, and C); each job has five activities (for example, $A_1, \ldots, A_5$) and five resources ($R_1, \ldots, R_5$). The release date for all

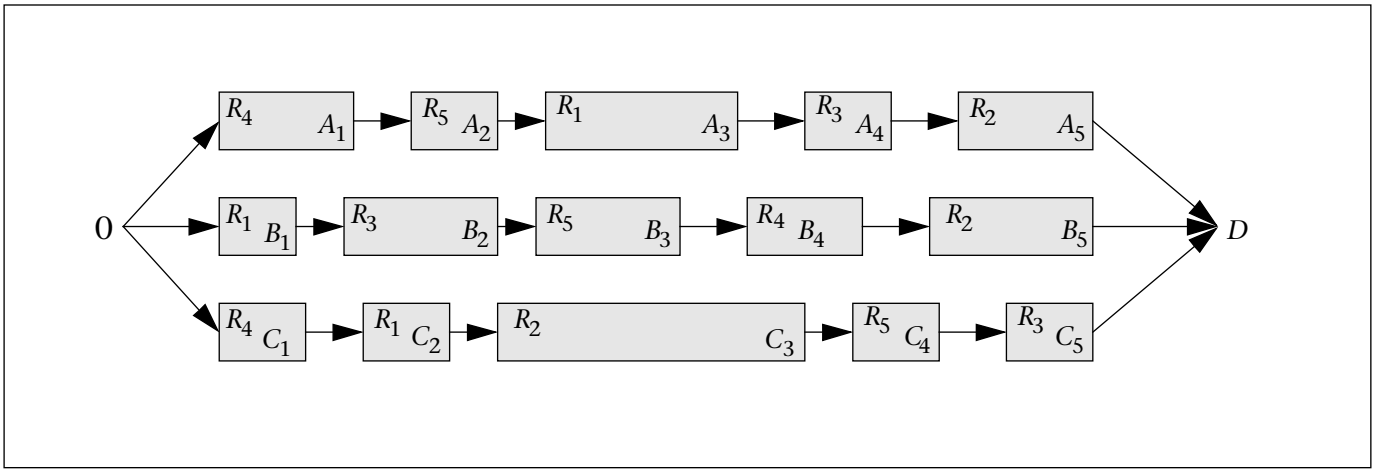

Figure 4. An Example 3 x 5 Job-Shop–Scheduling Problem.

jobs is 0, and the due date for all jobs is *D*. The resource required by each activity is indicated in the upper-left corner, and the duration, although not specified, is represented by the length of each activity. The arrows represent precedence constraints. For clarity, the transitive closure of the precedence relations is not displayed.

Many scheduling problems are not simply CSPs but, rather, COPs. Relatively simple optimization functions have been studied in the literature such as the minimization of makespan (that is, find the schedule with the minimum *D*) (Applegate and Cook 1991), minimization of the average (or maximum) tardiness of activities (that is, how late after their due date activities finish), or some combination of other attributes (for example, minimize work-in-process combined with tardiness) (Sadeh 1991; Smith et al. 1989; Fox 1983). Little work has addressed the many complex and interacting objective functions that typically arise in real-world problems.

Historical Perspective A number of threads of research have contributed to modern constraint-directed scheduling. It is beyond the scope of this article to discuss the contributions of each thread, much less those of each scheduling system. For our purposes, we note the three chief threads and direct interested readers to Fox (1990) and Le Pape (1994a) for more in-depth historical perspectives. There has been cross-fertilization among these threads as they have evolved. This categorization is not meant to indicate completely independent lines of research but, rather, the areas that modern constraint-directed scheduling draws on.

The knowledge representation thread: The original constraint-directed scheduling work is owed to Mark Fox and Steve Smith and their work on the ISIS scheduler (Fox 1983) at Carnegie Mellon University (CMU). Systems, both directly descended from ISIS (OPIS [Smith et al. 1989], CORTES [Sadeh and Fox 1989], MICROBOSS [Sadeh 1991]) and more indirectly related (SONIA [Collinot and Le Pape 1987], DAS [Burke and Prosser 1994], GERRY [Zweben et al. 1994, 1993], MINCONFLICTS [Minton et al. 1992], DCHS [Sycara et al. 1991], DISARM [Neiman et al. 1994]), investigate a wide space of constraint representations and solution techniques. In particular, this thread is responsible for the realization of the use of constraint to represent scheduling problems in their full generality as well as the search approach of focusing on the problem knowledge represented in the constraints as the main basis for heuristic decision making.

The constraint-programming thread: The constraint-programming community has traditionally stressed representation but used more generic solution techniques: Constraint-programming languages could typically represent problems far more complex than their solution techniques could handle. The constraint-programming thread, developed from Prolog, aimed to provide languages for clear, declarative problem representations, with constraint propagation being dealt with by the underlying language. Attempts to solve hard scheduling problems with these languages were often unsuccessful. The propagation found in early versions of constraint programming languages—forward checking and arc consistency—was not sufficiently powerful. More recent work in this field has developed specific propagation techniques for different types of constraint found in scheduling. These recent investigations have corrected the imbalance in solution power and have provided a number of impressive results (Van Hentenryck

1989; Caseau and Laburthe 1995; Caseau and Laburthe 1996; Nuijten 1994; Le Pape 1994b; Le Pape and Baptiste 1996).

The operations research thread: The long(er) history of operations research provides a number of techniques for constraint-directed scheduling. From work that predates constraint-directed scheduling itself (Baker 1974; Little et al. 1963) to techniques that have been adopted and adapted more recently (Applegate and Cook 1991; Glover 1990, 1989; Carlier and Pinson 1989; Erschler, Roubellat, and Vernhes et al. 1980, 1976), a variety of operations research methods have significant impact on both the approaches to, and performance of, modern constraint-directed scheduling systems. Within the operations research community, scheduling techniques have been developed based on mathematical programming techniques (integer programming, column generation) and local search (tabu search). Although these techniques can be useful, one drawback is that they tend to be developed for a specific problem type (for example, job-shop scheduling) and often cannot represent full, real-world problems in all their generality.

Notation For an activity, A_i, and a set of activities, S, we use the notation in table 1 throughout the balance of this article. We omit the subscript unless there is the possibility of ambiguity.

Symbol	Description
ST_i	A CSP variable representing the start time of A_i
STD_i	The discrete domain of possible values for ST_i
est_i	Earliest start time of A_i
lst_i	Latest start time of A_i
dur_i	Duration of A_i
eft_i	Earliest finish time of A_i
lft_i	Latest finish time of A_i
$lft(S)$	The latest finish time of all activities in S
$est(S)$	The earliest start time of all activities in S
$dur(S)$	The sum of the durations of all activities in S

CSP = constraint-satisfaction problem.

Table 1. Notation.

Scheduling Algorithms as Instances of the Framework

To illustrate the applicability of our framework, we now demonstrate how a number of existing scheduling algorithms can be conceptualized within it.

The ORR-FSS Algorithm One of the algorithms implemented in the MICROBOSS scheduler (Sadeh and Fox 1996; Sadeh 1994, 1991) is the operation resource reliance–filtered survivable schedules (ORR-FSS) algorithm. It is a constructive algorithm: Each iteration works on a consistent partial solution (that is, a subset of variables are consistently assigned) and attempts to extend it by assigning a start time to a currently unassigned activity.

Informally, ORR heuristically identifies the most critical activity by finding the activity that relies most on the resource and time for which there is the most contention. (A more formal definition of contention and reliance is provided later). FSS then rates the quality of the possible start times of the critical activity. The start time with the highest quality is assigned. If there is high contention for a resource at a particular time, t, it means that a (relatively) high number of activities are competing to execute at t. Intuitively, it is important for search to focus here because start times must be found for all the competing activities such that only one is executing at t. Focusing elsewhere is likely to result in a reduction of the possible start times of the competing activities to such an extent that a dead end is produced: One of the activities at t will not have any possible start times that do not conflict with times when other activities are executing. Focusing on the activities competing for t solves the subproblem before it becomes a dead end. Although the commitments to solve the subproblem might increase contention elsewhere in the graph because contention elsewhere was previously lower, the likelihood of producing a dead end is smaller than if search did not focus at the time most contended for.

Two propagators are used: (1) temporal propagation and (2) resource propagation. *Temporal propagation* (technically, arc-B-consistency) operates on the precedence constraints. If A_i and A_j are activities in the same job such that A_j is a successor of A_i, temporal propagation enforces $est_j \geq est_i + dur_i$ and $lft_i \leq lft_j - dur_j$. Resource propagation (arc consistency) plays a similar role for resource constraints. For example, if A_i and B_j require the same unary capacity

resource and $lst_j < eft_j$, then for the time interval $[lst_j, eft_j)$, B_j must be the only activity using the resource. (The interval notation $[t_1, t_2)$ represents all time points from t_1 to t_2, including t_1 but not including t_2. We follow the convention that an activity that starts at t_1 and ends at t_2 executes on the interval $[t_1, t_2)$. As a consequence, a successor activity may start at t_2 and, for example, execute the interval $[t_2, t_3)$). Resource propagation will remove the values $(lst_j - dur_i, eft_j)$ from the possible start times of A_i.

The retraction technique is chronological backtracking: The most recent commitment is retracted. The search continues until a solution is found or until a bound on the number of heuristic commitments is reached.

In summary, in terms of our framework, we represent ORR-FSS as follows: The termination criteria are that all activities are assigned a consistent start time, or a user-specified bound on the number of heuristic commitments is reached. The propagators used are temporal propagation and resource propagation. The heuristic-commitment technique is ORR-FSS, and the retraction technique is chronological backtracking.

In SOLVE, the heuristic-commitment technique is simpler, less expensive, and less powerful, but the propagators are more expensive and more powerful.

The SOLVE Algorithm The SOLVE algorithm (Nuijten 1994; Nuijten et al. 1993) is also a constructive algorithm but takes a different approach than ORR-FSS. In ORR-FSS, the main effort (and computational complexity) is in the heuristic-commitment component, but the other components are relatively inexpensive. In SOLVE, the heuristic-commitment technique is simpler, less expensive, and less powerful, but the propagators are more expensive and more powerful.

The heuristic-commitment technique finds the smallest earliest finish time, t, of all unscheduled activities and identifies the set of unscheduled activities that can start before t. One of these activities is randomly selected and assigned to start at its earliest start time. In addition to temporal propagation and resource propagation, SOLVE uses an extensive set of propagators, including edge finding (see later). The retraction technique is bounded chronological backtracking with restart. When a dead end is found, a limited number of chronological retractions are done. If no solution is found within this limit, search is restarted from the beginning. Because of the randomness in the heuristic-commitment technique, restarting is likely to result in the exploration of different search states. The conditions for termination of search are having an assignment for all activities or reaching the bound on the number of restarts.

GERRY GERRY (Zweben et al. 1994, 1993) is a local search algorithm. Working from a total assignment of start times that fails to satisfy some set of constraints, GERRY will reschedule or "repair" some activity to reduce the total cost of the schedule. The cost is a weighted sum of the extent to which each constraint is violated. The precedence constraints are always maintained (using temporal propagation); therefore, when applied to job shop, the only constraints that GERRY repairs are the resource constraints. To repair a resource constraint, GERRY reschedules one of the conflicting activities. If K is the set of activities contributing to a violation, GERRY will try to move each task to the previous and next times at which the resource is available. Each of these moves is evaluated by a linear combination of a number of factors, including the extent to which the size of the activity matches the size of the violation, the number of activities temporally dependent on the activity, and the distance from the current start time of the activity to the new start time. Each move is scored, and the score is used to select the heuristic commitment.

Every l commitments, the overall cost of the schedule is calculated. If the cost is less than the previous schedule (that is, from l iterations ago), it is accepted as the new schedule. If it is of lower cost than all schedules seen so far, it is also stored as the "best so far" schedule. Even if the schedule is of higher cost than the previous schedule, it is accepted by some probability based on a simulated annealing technique (Kirkpatrick, Gelatt, and Vecchi 1983); as the search progresses, a higher cost schedule is increasingly less likely to be accepted. Search continues until a zero-cost schedule is found (no violations), or a bound on the total CPU time is reached.

GERRY (and other local search techniques) can be modeled in our framework by encoding the local search moves as first retracting some commitment and then making a new commitment. At a conceptual level, we can view the retraction component as selecting the commitments to retract; the heuristic-commitment technique then identifies new commitments to be asserted. In reality, these two components are tightly integrated: The commitments that will be retracted are highly dependent on the local moves that the heuristic commitment can perform.

Every l iterations, the retraction step is different because the whole schedule is evaluated. If the new schedule is accepted (because of lower cost or probabilistic acceptance of higher cost), the usual retraction takes place (after replacing the stored solution with the new one). If the new schedule is not accepted, the

previous schedule is put back: The *l* most recent commitments are retracted, and the *l* commitments that must be (re)made to return to the previous schedule are asserted.

In summary, GERRY's termination criteria are the discovery of a zero-cost schedule or the reaching of a limit on CPU time. The only propagator used is temporal propagation. The heuristic-commitment technique and the retraction technique are tightly integrated. Together, they identify a violated constraint and rate, moving each activity taking part in the violation to the next earliest or next latest time at which the resource is available. Based on the rating, the start-time commitment of one of the activities is retracted. Every *l* iterations, the overall schedule is evaluated against the stored solution, and perhaps, the previous *l* commitments are retracted.

Tabu Search In tabu search (Vaessens, Aarts, and Lenstra 1994; Glover 1990, 1989), a heuristic neighborhood function defines a set of states that can be reached by retracting a set of commitments and then asserting a new set. For example, one neighborhood function is swapping the ordering of two adjacent activities (see later), and another is that used by GERRY. After the neighborhood function has (implicitly or explicitly) defined the set of neighboring states, each state is rated. The state that the search moves to depends not only on the rating but also on a (potentially) complex mechanism involving a tabu list; an aspiration criteria; and, perhaps, other caches of search information. A *tabu list* is typically a set of states, substates, state attributes, or moves that are prohibited as the next state. The form of the tabu list and the criteria for adding and removing elements can be varied and complex and, indeed, are an important research issue. A simple example is to have a tabu list of limited length and insert every move on the list; when the tabu list is full, the oldest moves are removed. While the move is on the tabu list, it cannot be performed. The intuition is that search should not revisit states too frequently, to avoid being trapped in a local optimum of the state-rating function. An aspiration criteria is a condition (or set of conditions) for overriding the tabu list. Like the tabu list, the aspiration criteria can be complex, and a full discussion is beyond our scope. As an example, perhaps a state with an attribute that is on the tabu list is selected anyway because it is more highly rated than any state visited to this point in the search.

Based on the combination of the neighborhood function, the state-rating function, the tabu list, and the aspiration criteria, a search state to move to is selected. The search terminates when a state with the optimal rating is found or when a bound on the number of iterations is reached.

With a conceptualization similar to that used with GERRY, we see that the neighborhood function, the tabu list, the aspiration criteria, and the state-rating function of the heuristic-commitment technique are closely integrated with the retraction component. Together the two components work to evaluate and select a neighboring state, retract a set of commitments, and assert a new set of commitments that move the search to the selected neighbor.

In terms of our framework, the search terminates when an optimal state is found or when the maximum number of iterations is reached. No propagators are used. The heuristic-commitment technique and the retraction technique use a (potentially) complex combination of neighborhood function, tabu list, and aspiration criteria to identify the commitments to be retracted and the ones to be asserted to move to the selected neighboring state.

Genetic Algorithms Genetic algorithms (Goldberg 1989; Holland 1975) operate on an evolving population of potential solutions rather than a single solution. An initial population of solutions is created by some (usually randomized) means, and each individual schedule is rated. Based on the rating, a subset of existing solutions are allowed to "reproduce" either by *crossover* (two new schedules are created from two existing schedules by exchanging some commitments) or *mutation* (a random subset of commitments are changed). The crossover and mutation operators are usually random or, at least, have a random component. After reproduction, potential solutions from the previous generation are discarded, although in some genetic algorithms, those that are highly rated remain in subsequent populations. The search repeats until a bound is reached on the number of generations or until the optimal solution is found. Clearly, performance of a genetic algorithm depends highly on the rating function and the crossover and mutation operators.

Genetic algorithms … operate on an evolving population of potential solutions rather than a single solution.

In terms of our framework, the definition of termination criteria and propagators is straightforward. A commitment, however, is now an entire schedule (for example, a set of unary constraints that assign a start time to each activity or, in an alternative formulation, a set of binary precedence constraints that order all activities on each resource). To this point, we have had a single constraint graph to which constraints are added and from which constraints are removed. To represent a popu-

lation of schedules, we need a set of constraint graphs, one for each population member. A single commitment results in a single population member with the addition of a complete set of constraints to its own constraint graph (that is, a separate copy of the problem). Each iteration uses the previous set of commitments (the previous generation) to create a new set of commitments using the reproduction operators. The heuristic-commitment technique embodies the rating of each population member and the use of the crossover and mutation operators to create new schedules (that is, sets of new commitments). The retraction component discards all (or some low-rated subset of) the members of the previous generation.

We define genetic algorithms as follows: The search is terminated when an optimally rated solution is found or when the maximum number of generations is reached. No propagators are used. The heuristic-commitment components rates the current generation of schedules (based on some heuristic rating scheme) and produces a new set of commitments using the crossover and mutation operators. The retraction technique discards the commitments that formed the members of the previous generation.

… to account for local search algorithms in our framework, we had to tightly integrate (if not actually combine) the heuristic-commitment technique and the retraction technique.

Why the Framework?

With these examples, two points, in particular, should be noted: First, to account for local search algorithms in our framework, we had to tightly integrate (if not actually combine) the heuristic-commitment technique and the retraction technique. Second, to fit genetic algorithms into the framework, we need to account for the population of schedules on which genetic algorithms operate.

Given these points, the question arises: "Why should we force these disparate algorithms into the framework?" The answer offered here is that it is not such a leap of faith to conceptualize these algorithms as we have, and even if it were such a leap, the benefits of the unified framework are significant.

Answer 1: The Conceptualization Is Not Forced We believe the conceptualization of the various algorithms is not forced. To account for local search algorithms, we have tightly integrated the retraction and heuristic-commitment techniques. At a conceptual level, ignoring issues of computational efficiency, we could have maintained the separation between the components at the expense of repeated computation. For example, the retraction technique of GERRY might do its analysis, pick a new state, and then simply retract the commitments necessary to move to the new state. The heuristic-commitment component could then repeat some of the analysis to identify which commitments to assert. The point is not that the heuristic-commitment component and retraction component must be completely independent, but rather that in many cases, they might be completely independent; choosing a retraction component does not necessary limit the choice of heuristic-commitment technique. For example, one could imagine replacing the heuristic-commitment technique in GERRY with another (for example, ORR-FSS) that conducts a completely, partially, or probabilistically different analysis than the retraction component.

It was necessary for genetic algorithms to account for the population of schedules by having the commitments that are asserted and retracted correspond to a member of the population. However, our original definition of a commitment noted that it might be a set of constraints. The fact that each iteration retracts and asserts a whole population of these commitments into different constraint graphs does not significantly modify this definition.

Answer 2: The Benefits The benefits from describing these strategies within the framework are significant. One key benefit is the ability to perform empirical comparisons of different instances of the same component. As we demonstrate in the balance of this article, little empirical work compares, for example, two instances of heuristic-commitment components, while all other components are held constant. As a result, it is difficult to compare strategies, attribute performance observations to specific components, and correlate problem features to performance. The framework allows us to experiment with a variety of combinations and generate and test specific hypotheses to explain performance differences.

A second benefit is the ability to create novel strategies and components with combinations of existing ones. For example, another way to account for the relation between the heuristic-commitment technique and the retraction technique of GERRY is to implement a form of communication between the retraction component and the heuristic-commitment component. We could place the intelligence of the local-move calculation in the retraction component and have it communicate the to-be-asserted commitments to the heuristic-commitment component. It might be that a wide range of scheduling strategies can benefit from such a communication between the retraction and heuristic-commitment components. In many of the strategies discussed here, the main focus of intelligence tends to be in only one of

the framework components. With communication, we can put intelligence in all components and have them communicate suggestions about what appear to be good commitments to retract or assert.

By placing genetic algorithms within our framework, we can similarly imagine a number of interesting scenarios. For example, we could modify the genetic algorithm retraction technique to, under certain circumstances (for example, finding a solution significantly better than all other solutions seen), retract all individual schedules and transfer to, for example, a tabu search on this one solution. Rather than moving to tabu search, perhaps the retraction component could not only retract all but the best solution but also remove some subset of the constraints from the best solution. The heuristic-commitment technique could become ORR-FSS, and the retraction component could become chronological backtracking to try to build a complete schedule from the promising partial schedule. Similarly, from a tabu search, perhaps we could, under certain circumstances, spawn a number of schedules from the best schedule it has found and move to a genetic algorithm search.

The point is not whether these scheduling strategies will work (that remains to be seen) but, rather, that simply being able to conceptualize genetic algorithms and other local search algorithms, together with constructive search, provides a new perspective on all these techniques that, in turn, creates a space of novel algorithms to be investigated.

Propagators

In the examples of our framework, we noted temporal propagation, resource propagation, and edge finding as propagators. Because of space limitations, we only discuss two propagators in depth here; one we have already encountered (edge finding), and the other we have not (constraint-based analysis).

Constraint-Based Analysis

Constraint-based analysis (CBA) (Cheng and Smith 1996; Erschler, Roubellat, and Vernhes 1980, 1976; Smith and Cheng 1993) enforces arc-B-consistency on the resource constraints. Arc-B-consistency (Lhomme 1993) (where *B* stands for *bounds*) ensures that for the minimum and maximum values for the any variable, v_1, at least one consistent assignment exists for any other connected variable, v_2. Clearly, arc-B-consistency is limited to variables where there is a total ordering over the values. The start-time variables in scheduling meet this requirement.

CBA analyzes the start and end times of all pairs of activities and, given activities A_i and A_j, identifies the following cases:

Case 1. If $lft_i - est_j < dur_i + dur_j \leq lft_j - est_i$, then A_i must be before A_j.

Case 2. If $dur_i + dur_j > lft_j - est_i$ and $dur_i + dur_j > lft_i - est_j$, then the current state is a dead end.

Case 3. If $dur_i + dur_j \leq lft_j - est_i$ and $dur_i + dur_j \leq lft_i - est_j$, then either sequence is still possible.

If after looking at all pairs of activities on each resource, CBA finds that all pairs are in case 3, it cannot infer any new constraints: All the resource constraints are arc-B-consistent.

The worst-case time complexity for CBA is $O(MN^2)$, where N is the number of activities on one resource, and M is the number of resources.

In the example shown in figure 5, CBA is able to infer that A_1 must be before B_3 because the interval between the earliest start time of B_3 and the latest end time of A_1 is 15, which is smaller for the combined durations of the activities, which is 20.

Empirical evaluation of CBA with the PCP SLACK heuristic (Smith and Cheng 1993) (see later) has demonstrated good performance on one set of job-shop–scheduling benchmarks. The experimental design did not evaluate CBA independently of the heuristic-commitment technique, so it cannot be judged whether the performance was because of CBA, the heuristic, or the combination.

Empirical evaluation of CBA with the PCP SLACK heuristic … has demonstrated good performance on one set of job-shop–scheduling benchmarks.

Edge Finding

Given S, a set of activities executing on the same resource, and activity $A \notin S$ on the same resource as the activities in S, edge finding makes operational the implications 1 and 2.

$$[(lft(S) - est(S) < dur_A + dur(S)) \wedge (lft(S) - est_A < dur_A + dur(S)) \rightarrow est_A \geq est(S) + dur(S). \quad (1)$$

$$(lft(S) - est(S) < dur_A + dur(S)) \wedge (lft_A - est(S) < dur_A + dur(S)) \rightarrow lft_A \leq lft(S) - dur(S). \quad (2)$$

Implication 1 states that if A is scheduled at its earliest start time, and there is not enough room for all the activities in S before the latest finish time of S, then A must occur after all the activities in S have finished. Implication 1 can be used to derive a new earliest start time for A.

Similarly, implication 2 is used to find a new latest end time. If A is scheduled at its latest start time, and there is not enough room for all the activities in S before A and after the earliest time of S, then A must occur before all the activities in S start.

Complete examination of the $O(2^N)$ subsets of activities is not practical. However, by defin-

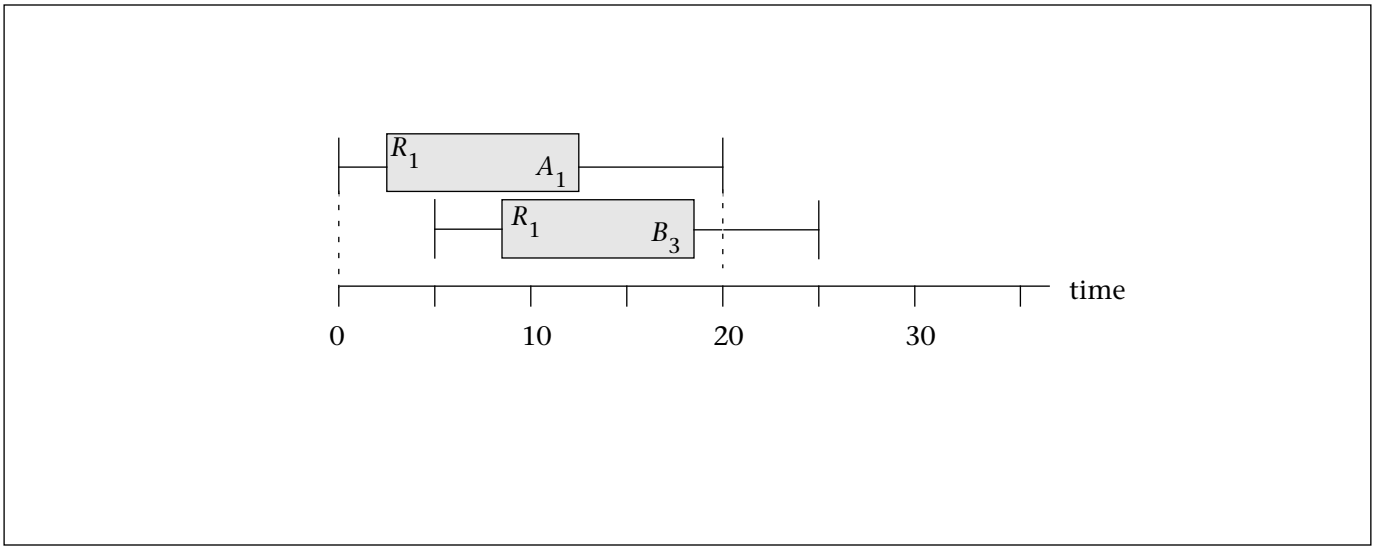

Figure 5. An Example Where Constraint-Based Analysis Can Infer A New Constraint: A_1 *before* B_3.

ing a task interval (Caseau and Laburthe 1996) as follows, it is possible to represent all subsets with only $O(2^N)$ sets:

If activity A_i and A_j are two activities (possibly the same) that use the same resource such that $est_i \leq est_j$ and $lft_i \leq lft_j$, then the task interval $[A_i, A_j]$ is the set of tasks $\{A_k\}$ that use the same resource as A_i and A_j such that $est_i \leq est_k$ and $lft_k \leq lft_j$.

It has been shown (Caseau and Laburthe 1996; Carlier and Pinson 1989) that any constraints derived by edge finding on a task interval are at least as strong as the constraints that could be derived from any subset of the task interval. Clever implementation (Caseau and Laburthe 1996; Nuijten 1994; Carlier and Pinson 1989) is able to reduce the complexity of the edge-finding algorithm to $O(MN^2)$.

Figure 6 presents an example where edge finding is able to infer not only that C_4 must execute after A_1 and B_3 but also that the earliest possible start time of C_4 is 25. Implication 1, where $S = \{A_1, B_3\}$ and $A = C_4$, is applicable and results in a new lower bound on the start time of C_4.

Empirical evaluation of edge finding for job-shop scheduling has concentrated primarily on the operations research library set of benchmarks from the operations research community (Beasley 1990). Results show that a simple randomized heuristic-commitment component plus edge finding and a number of simpler propagators can find makespans approximately 25 percent closer to optimal than the heuristic by itself (Nuijten 1994) and further outperforms ORR-FSS with the same set of propagators.

Other work (Caseau and Laburthe 1996, 1995) has extended edge finding by adding additional valid implications and has demonstrated favorable empirical results against a number of polyhedral (cutting-plane) methods from the operations research literature.

Open Issues

There are a number of open issues in the research concerning the use of propagators and the potential impact of new propagators.

Given the theoretical basis for propagators as consistency techniques, there might be other forms of consistency that can be used in scheduling. Many of the results using propagators have been excellent; therefore, one avenue for research is to characterize existing propagators about the type of consistency they enforce and look for novel propagators based on different types of consistency.

Propagators have primarily been studied in the job-shop model, where they are able to exploit the fact that two activities requiring the same resource cannot execute at the same time. Applications of edge finding to other types of resource (multiple capacitated [Nuijten and Aarts 1997; Nuijten 1994] and cumulative [Caseau and Laburthe 1996]) have resulted in preliminary evidence, indicating that edge finding, although still useful, has less impact where constraints are more complex. There is, therefore, the dual challenge of extending these techniques for more complex constraints and, should they be found wanting, identifying new propagators.

Currently, it appears that propagators are applied across an entire problem. Although

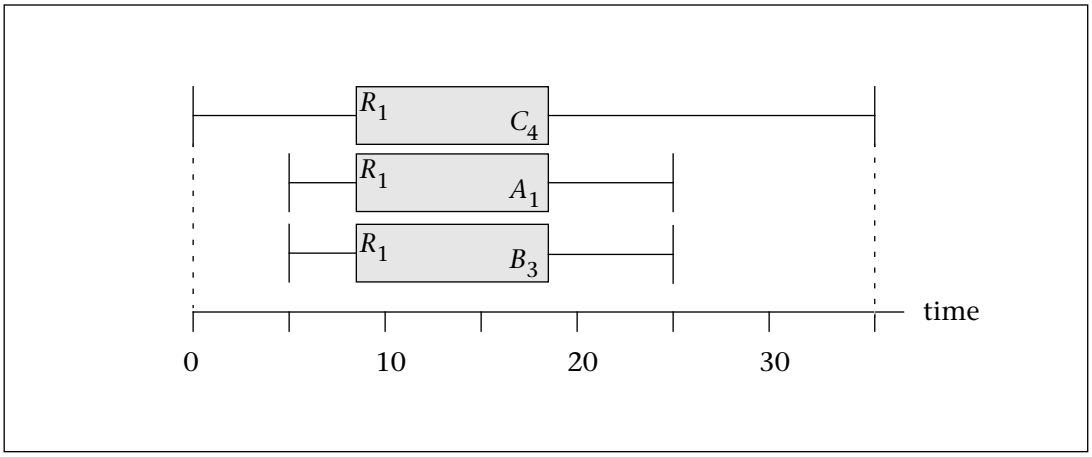

Figure 6. An Example Where Edge Finding Can Infer a New Constraint: $ST_C \geq 25$.

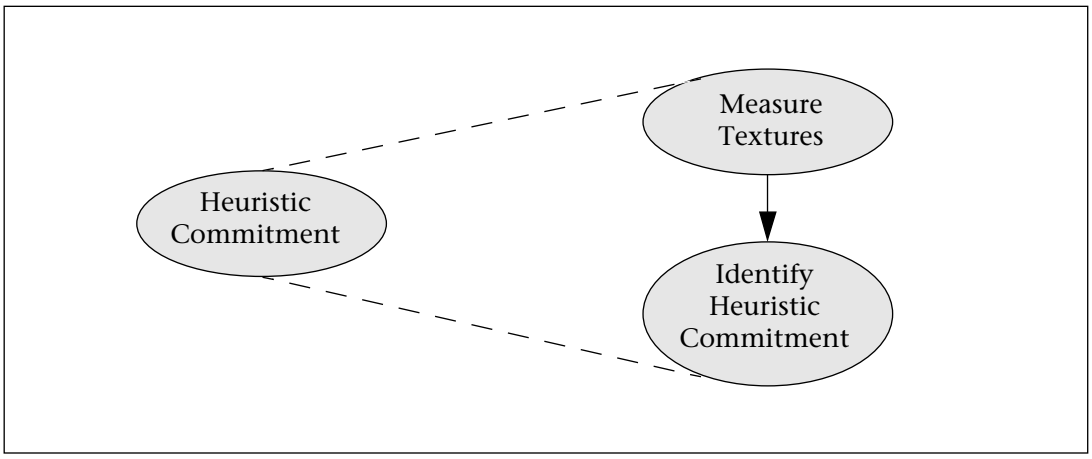

Figure 7. The Subcomponents of the Heuristic-Commitment Component.

anecdotal evidence suggests that on some problems, the constraints inferred by edge finding often involve a small number of activities, there has been no work that we are aware of that attempts to identify problem substructures that (heuristically) indicate that a particular propagator might have a great impact. If structures within the scheduling problem where a propagator is likely to be useful can be identified cheaply, it might be possible to avoid the use of propagators where they are unlikely to contribute. (An example of a selective application of a propagator based on information from a commitment-retraction technique is the dynamic consistency enforcement technique [Sadeh et al. 1995] discussed later). For example, because propagators find implied commitments, it is reasonable to expect that they will perform better in more constrained problems. It might be useful, therefore, to look for a correlation between constraint tightness (and other problem characteristics) and propagator performance. Identification of such a correlation will not only aid in understanding propagators but might speak to the other research issues.

Heuristic-Commitment Techniques

The heuristic-commitment component of our framework can be further subdivided into two components, as shown in figure 7.

In the first subcomponent, texture measurements distill information from the constraint graph representation. Based on the distilled information, in the second subcomponent, a commitment is found that will (heuristically) move toward a solution state.

Before discussing texture measurements in more depth, it is instructive to note that one of the key differentiating factors among existing heuristic-commitment techniques is the tradeoff between computational complexity and the accuracy of the heuristic information. At one end of the spectrum is the approach that can

be characterized as "do the right thing, once." A significant amount of computation effort is expended at each search state to distill the texture information and make the right commitment based on the information. The hope is that this investment of (polynomial) effort at each state will pay off by avoiding the (exponential) effort of frequent retraction of commitments. The other end of the trade-off spectrum is the approach that is characterized as "do something, often." In this approach, many commitments are made and retracted with the belief that a solution will eventually be found. Little computational effort is spent at each search state measuring textures; however, many search states are visited. The extreme of this approach is to not spend any effort in measuring textures, and indeed, this extreme approach is the basis of randomized heuristic-commitment techniques such as those used in genetic algorithms. The crux of this trade-off stems from the quality of the information that the heuristic decisions are based on. If the information does not appreciably increase the likelihood of making a correct decision at a search state, then the extra computation is wasted. It is better, then, to move quickly through the search space.

In the balance of this section, we discuss textures in depth before examining a number of heuristic-commitment techniques. For each technique, we provide insight into the decisions that were made with respect to the previous three issues: (1) what information is being estimated, (2) what the (relative) complexity of the estimation algorithm is, and (3) what heuristic commitments are made.

Texture Measurements

Although a relatively small number of texture measurements have explicitly been identified (Sadeh 1991; Fox et al. 1989), we take the broad view of a texture measurement as any analysis of the constraint graph producing information on which heuristic commitments can be based. A texture measurement is not a heuristic; rather, it is a technique for distilling the knowledge in the constraint graph into a form that heuristics can use. A texture measurement might, for example, label some structures in the constraint graph (for example, constraints, variables, subgraphs) with information condensed from the surrounding graph. On the basis of this condensed information, heuristic commitments can be made.

Example 1: The Contention Texture

The contention texture (Sadeh 1991) is the extent to which variables related by a disequality compete to be assigned to the same value. In the ORR-FSS scheduling heuristic (see later), contention for a resource is estimated by finding a probabilistic estimate of each activity's individual demand for the resource and then summing these individual demands to form an aggregate demand for the resource.

Example 2: The Fail-First Heuristic

Recent work on the familiar fail-first heuristic for CSPs (Haralick and Elliot 1980) helps to illustrate the differences among texture measurements, estimations of texture measurements, and heuristics based on the estimations. The heuristic states that in a CSP, a good choice for the next variable to assign is the one most likely to *fail*, that is, most likely to result in a dead end after it is assigned a value. The texture measurement, therefore, is the probability of failure of each variable, and this probability, in the traditional fail-first heuristic, is estimated by the domain size of each variable: the smaller the domain size, the higher the probability of failure. In practice, the variable ordering achieved by choosing the smallest domain size (and variations of it) is robust and often outperforms other more complicated orderings (Gent et al. 1996).

This understanding of the fail-first heuristic has been challenged by using better estimations of the probability of failure texture measurement (Smith and Grant 1997). If the smallest domain-size heuristic works because of its basis on the estimation of the probability of failure texture, better estimations of the texture should lead to better performance (when the cost of the estimation is not taken into account). Smith and Grant (1997) create four increasingly accurate estimations of the probability of failure texture. Contrary to predictions, the heuristics based on these better estimations perform worse than the smallest domain-first heuristic. The success of the smallest domain-first heuristic is because the domain size is an estimate for some other texture measurement, not the probability of failure. The identity of this other texture measurement remains an open question.

Basic Texture Measurement Questions

The previous examples illustrate three basic questions surrounding texture measurements:

First, what is the information that is to be distilled? If we are to understand the heuristic-commitment technique, we need to have a clear model of the information on which our heuristic technique operates. We might not be able to precisely calculate this information in any practical algorithm; however, a firm theoretical basis shows that if we had this information, we could use it to find a solution that allows us to then look to the practical aspects

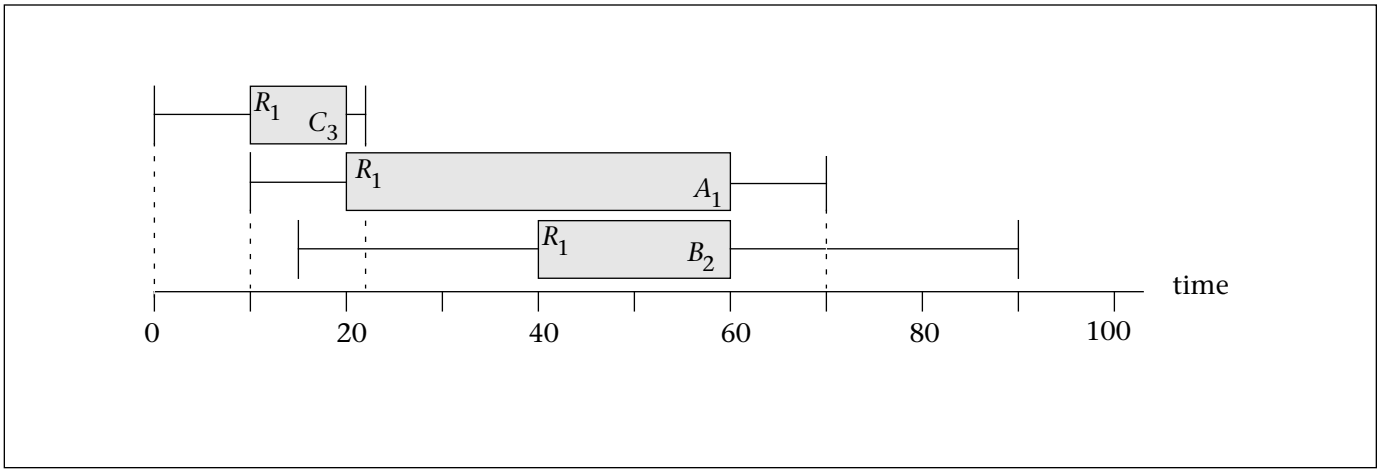

Figure 8. Individual Demand Curves for A_1, B_2, *and* C_3.

of forming an estimate of this information.

Second, given the impracticality of precisely calculating texture information, can we construct an algorithm that estimates the desired information? It is likely that we can construct a number of algorithms producing estimates of increasing accuracy at the cost of increasing computational complexity. Where is the tradeoff in terms of impact on the overall scheduling algorithm?

Third, what are the heuristic commitments that are being made on the basis of the information distilled by the texture measurements? After the texture measurements have estimated search information, it is still necessary to use this information to make a heuristic commitment. The type of commitment and the heuristic for finding the instance of the commitment, based on the distilled information, have a significant impact on the search.

ORR-FSS Heuristic

The ORR-FSS heuristic (Sadeh 1991) is based on two texture measurements: (1) contention and (2) reliance.

Contention is defined as the extent to which variables linked by a disequality constraint compete for the same value. In the context of job-shop scheduling, contention is the extent to which activities compete for the same resource over the same time interval. In ORR-FSS, contention is estimated by finding a probabilistic estimate of an activity's individual demand for the resource and then summing the individual demands for a resource to form an aggregate demand (see later).

Reliance is the extent to which a variable must be assigned to a particular value to form an overall solution. In scheduling, one illustration of reliance arises with alternative resources. If activity A_1 requires resources R_1, R_2, R_3, or R_4, and activity A_2 requires resources R_2 or R_5, clearly A_2 has a higher reliance on R_2 than A_1. If A_1 is not assigned to R_2, it has three other resource alternatives; however, A_2 only has one. Reliance can also be formulated in the context of an activity relying on being assigned to a particular start time on a particular resource. In ORR-FSS, reliance is a probabilistic estimate of the activity's time preferences.

To calculate contention for a resource, we first determine the individual demand that each activity has for the resource. If an activity does not require a resource, it has no demand for it; so, its individual demand is 0. Otherwise, to calculate an activity's individual demand, a uniform probability distribution over the possible start times is assumed: Each start time has a probability of $1 / |STD|$. (Recall that *STD* is the domain of the activity's start-time variable. A uniform probability distribution is the "low-knowledge" default. It might be possible to use some local propagation in the constraint graph to find a better estimate of the individual demand [Muscettola 1992; Sadeh 1991].) The individual demand, $ID(A, R, t)$, is the probabilistic amount of resource R, required by activity A, at time t. It is calculated as follows for all $est_A \leq t < lft_A$:

$$ID(A, R, t) = min(t, lst_A) - max(t - dur_A + 1, est_A) / |STD|. \quad (3)$$

Figure 8 shows the individual demand curves for the three activities in figure 9.

Contention is estimated by aggregate curves found for each resource by summing the individual demand curves. The aggregate demand curves and a time interval equal to the average activity duration are then used to identify the {resource, time interval} with the greatest area. By definition, the unassigned activity that contributes the most area to the critical time inter-

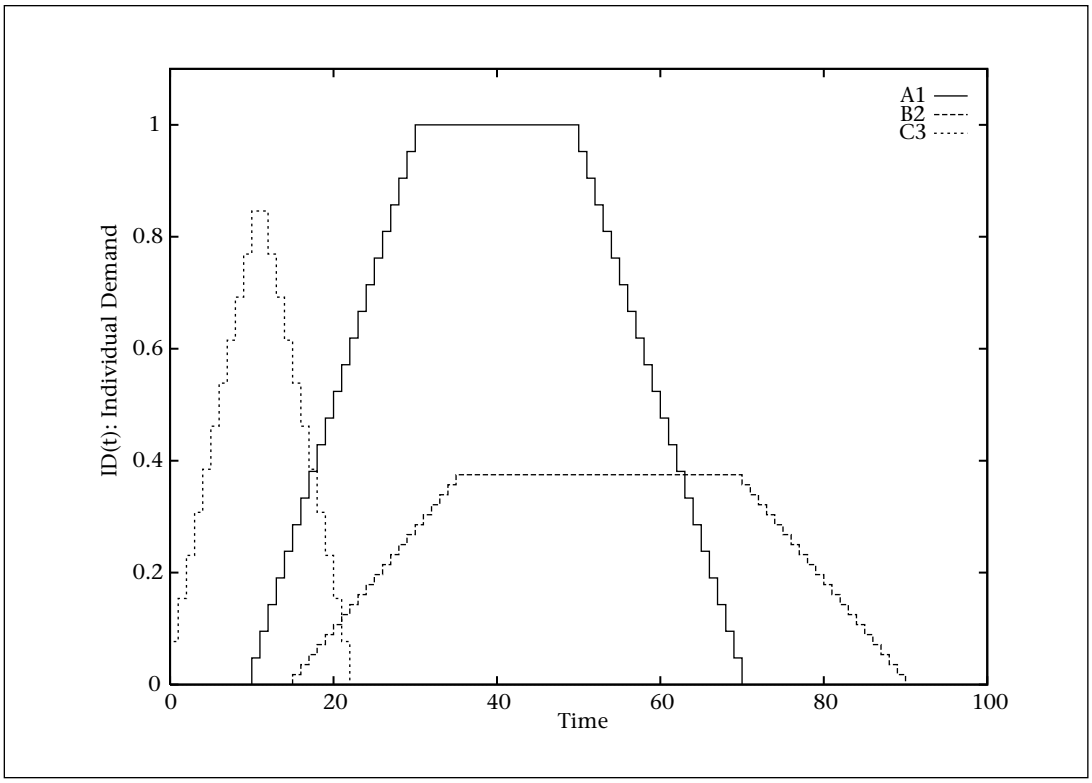

Figure 9. Activities A_1, B_2, *and* C_3.

val is the most critical activity: It is the most reliant activity on the most contended-for {resource, time interval} pair.

Once the critical activity, *A*, is identified, FSS is used to rate each of its possible start times by using the demand curves. This rating takes into account the effect an assignment to *A* will have both on activities competing directly with *A* and on those temporally connected to *A*. A more detailed description of the start-time–assignment heuristic is beyond the scope of this document; interested readers are referred to Sadeh (1991).

Sadeh presents empirical evidence showing that the MICROBOSS scheduler, using ORR-FSS, dominates all the tested dispatch rules on a set of 60 problems. Subsequent work showing the competitive performance of simpler heuristics (for example, the PCP SLACK heuristic later) (Smith and Cheng 1993) and significantly better performance using edge finding (Nuijten 1994) have called into question the efficacy of ORR-FSS. The difficulty is that it is not clear that it is the heuristics, rather than, for example, the retraction method, that lead to the poor results. No work that we are aware of has compared only the heuristic-commitment techniques and held all other components constant. Without such an experiment, meaningful comparisons of heuristic-commitment techniques cannot be made.

Task-Interval–Entropy Heuristic

Caseau and Laburthe (1995) use task intervals as a basis for the estimation of resource contention. For task interval *I* and resource *R*, the following are defined: U_R is the set of all activities using *R*, *First*(*I*) is the set of activities in *I* that might execute the first of the activities in *I*, and *Last*(*I*) is the set of activities in *I* that might execute the last of the activities in *I*.

The *task-interval–entropy (TIE) heuristic* identifies the resource with the highest contention as the one that minimizes the product in expression 4.

$$IntervalSlack(I^*) \text{ x } ResourceSlack(r) \text{ x } min(First(I^*), Last(I^*), par). \quad (4)$$

$$IntervalSlack(I) = lft(I) - est(I) - dur(I). \quad (5)$$

$$ResourceSlack(R) = lft(U_R) - est(U_R) - dur(U_R). \quad (6)$$

Where I^* is the task interval of *R* with the minimum *IntervalSlack,* and *par* is a parameter set empirically to about 3. I^*_R is then defined to be I^* on the critical resource. TIE chooses the smallest set from $First(I^*_R)$ and $Last(I^*R)$ and picks two activities, *A* and *B*, to sequence. If $|First(I^*_R)|$ is smaller, *A* is the activity with minimal *est,* and *B* is chosen to minimize the sum of the slack resulting from sequencing *A* before *B* and sequencing *B* before *A*. Similarly,

if $|Last(I^*_R)|$ is smaller, B is the activity with maximal *lft,* and A is chosen to minimize the sum of the slack resulting from sequencing A before B and B before A. In either case, A before B is posted.

The intuition for choosing the activities that minimize the sum of the slack is that a search state is a branch point, and it is desirable to simplify the problem as much as possible on both branches. Minimizing the slack is likely to increase the number of implied commitments that the propagators can find and, therefore, simplify the problem the most. Experiments are carried out with this heuristic, sophisticated propagators (including edge finding and some extensions), and chronological backtracking on the operations research library problems. Although there are no direct comparisons to other constraint-directed work (comparisons are done with a number of polyhedral [cutting-plane] methods), comparison of reported results with those found in Vaessens, Aarts, and Lenstra (1994) seems to indicate that TIE is competitive or better than the other approaches.

Interestingly, the heuristic attempts to minimize the resulting slack on both branches (A before B and B before A) because then the propagators will infer more constraints. In contrast, the PCP SLACK heuristic (later) attempts to maximize the slack remaining after a commitment.

Resource Slack, First-Last Heuristic

Baptiste, Le Pape, and Nuijten (1995) use the minimum RESOURCE SLACK to identify the resource with the highest average contention. Based on a comparison of the time windows available for each unsequenced activity on the critical resource, one is selected to execute first (or last). All activities on the critical resource are sequenced before identifying the next critical resource. Three heuristics are examined for activity selection:

First is *choose first.* An activity is chosen to execute first among the unsequenced activities. A set of propagators are used to identify activities that cannot execute first; however, it is unclear precisely which are used. Once the set of activities is identified, one activity is selected to execute first by the *EST-LST rule:* the activity with the smallest *est* breaking ties with the smallest *lst.*

Second is *choose last.* An activity is chosen to execute last among the unsequenced activities. Analogously, identify the set of activities that can execute last from the set of activities, and use an *LFT-EFT* (latest *lft* with ties broken by latest *eft*) to pick the activity to schedule last.

Third is *choose dynamically. A*n activity is heuristically chosen to execute either first or last based on the size of the sets of activities that are able to execute first or last. The smaller set of activities is chosen, and then the appropriate rule (EST-LST or LFT-EFT) is used to select an activity from this set. If the set sizes are equal, the tie is broken by looking at the difference between the top two activities to schedule first and the top two activities to schedule last. If the difference is greater between the activities to execute first, then this set is used; otherwise, the activities that can execute last are used.

Once the critical resource is identified, all activities on it are scheduled before moving to another resource, which is different from most modern techniques that have a more opportunistic nature in that after each commitment the critical resource is recalculated. Such resource-centered approaches were previously investigated in ISIS (Fox 1983) and OPIS (Smith et al. 1989).

Empirical evidence shows little difference among the activity-selection heuristics; however, they are not directly compared with any other heuristics. Reported results of these heuristics with edge finding and chronological backtracking seem to be a significant improvement over many techniques, although they lag behind results using the TIE heuristic. Again, head-to-head comparison is necessary for firmer conclusions to be drawn.

PCP Slack Heuristic

The precedence constraint-posting slack (PCP SLACK) (Cheng and Smith 1996; Smith and Cheng 1993) makes use of the CBA propagator, as described previously; when CBA is unable to find an implied commitment (that is, in case 3 previously), a heuristic commitment is made.

BSLACK, as defined in equation 7, is calculated for all pairs of activities, and the pair with the smallest BSLACK value is identified as the most critical. BSLACK is an estimate of the two-way contention among activities. The smaller BSLACK is, the greater the contention there is between the two activities. The sequence that preserves the most slack is the one chosen. The intuition here is that a pair with a smaller BSLACK is closer to being implied than one with a larger value. Once we identify this pair, it is important to leave as much temporal slack as possible in sequencing them to increase the likelihood of not having to backtrack.

$$bslack(i \rightarrow j) = slack(i \rightarrow j) / \sqrt{S} \quad (7)$$

$$slack(i \rightarrow j) = lft_j - est_i - (dur_i + dur_j). \quad (8)$$

$$S + min(slack(i \rightarrow j), slack(j \rightarrow i)) / max(slack(i \rightarrow j), slack(j \rightarrow i)). \quad (9)$$

Empirical evidence (Smith and Cheng 1993)

shows that the PCP SLACK heuristic combined with CBA, competes well with ORR-FSS, although different types of commitment are made (that is, precedence constraints rather than start-time assignments).

The Randomized Left-Justified Heuristic

The heuristics reviewed previously all estimate the contention-texture measurement by increasingly simple algorithms: ORR-FSS and TIE use sophisticated techniques to estimate the contention based on all activities, but PCP SLACK simply uses only the two-way interactions. In the randomized left-justified heuristic (Nuijten 1994; Nuijten et al. 1993), contention is not calculated, but rather, a small set of reliance estimates are made. The set of activities that can execute before the minimum earliest finish time of all unscheduled activities is identified. It is assumed (under this estimation) that each activity in this set has equivalent reliance on its earliest start time and, further, that it relies more on its earliest start time than any other of its possible start times. With this information, one of the activities is randomly selected and scheduled at its earliest start time.

Experiments compare SOLVE (using sophisticated propagators and bounded chronological backtracking with restart) with the ORR-FSS heuristic (using chronological backtracking, temporal propagation, and resource propagation) and with ORR-FSS augmented with the propagators used in SOLVE (still with chronological backtracking). SOLVE strongly outperforms the augmented ORR-FSS, which, in turn, strongly outperforms unaugmented ORR-FSS.

It is not clear from this study whether chronological backtracking or the ORR-FSS heuristics are to blame for the relatively poor performance of ORR-FSS. However, these results raise further questions about sophisticated heuristic techniques such as ORR-FSS.

Local Search Heuristics

Finally, there are a number of heuristic-commitment techniques often used in local search algorithms. Many such techniques fall into the "do something, often" extreme noted previously, although not all do. For example, in tabu search, the heuristic-commitment technique involves not only a neighborhood function but also the tabu list; aspiration criteria; and, perhaps, other caches of search information. Indeed, the complexity of the neighborhood function, although often small, is not prescribed by the tabu search method. With this caveat in mind, we examine a number of inexpensive heuristic-commitment techniques often used in local search.

The basic MINCONFLICTS algorithm (Minton et al. 1992) evaluates contention in the existing state by counting the number of resource violations for each activity. The activity with the most violations (that is, an activity that relies on the most-contended-for resource and time) is chosen, and for each of its possible start times, the number of resulting resource violations is assessed. The start time with the lowest number of violations is chosen as the new commitment. The heuristic commitment used in GERRY (Zweben et al. 1994, 1993) is a variation of MINCONFLICTS.

MINCONFLICTS has performed well on some large CSPs (for example, N-queens with N = 1000000) and on a particular set of space shuttle scheduling problems (Zweben et al. 1994, 1993). Performance is weaker on other job-shop–scheduling benchmarks (Davis 1994).

Vaessens, Aarts, and Lenstra (1994) provide an in-depth comparison of many of the neighborhood functions that have been applied to scheduling. Based on a search state where all the activities on a resource are completely sequenced, the neighborhood algorithms span the swapping of two adjacent activities, *jumps* that remove an activity from its current position and insert it before or after another (non-adjacent) activity, the reordering of all activities on a single machine, and the modification of orderings across multiple machines. Examples of the three single-machine moves are shown in figure 10.

Neighborhood functions do not estimate texture measurements but, rather, look ahead to the states that would occur if particular commitments are made. Based on the evaluation of the resulting states (which can include estimation of texture measurements), a move is selected.

Unfortunately, the empirical comparisons embed the neighborhood functions in different overall strategies (for example, one heuristic with tabu compared to another heuristic with simulated annealing). Although they are able to compare the overall search strategies, it is not clear how these results speak to a comparison based solely on the neighborhood functions.

Open Issues

As with propagators, there are a number of interesting research issues concerning heuristic-commitment techniques.

We have presented heuristic-commitment techniques from the perspective of texture measurements even though, with the exception of

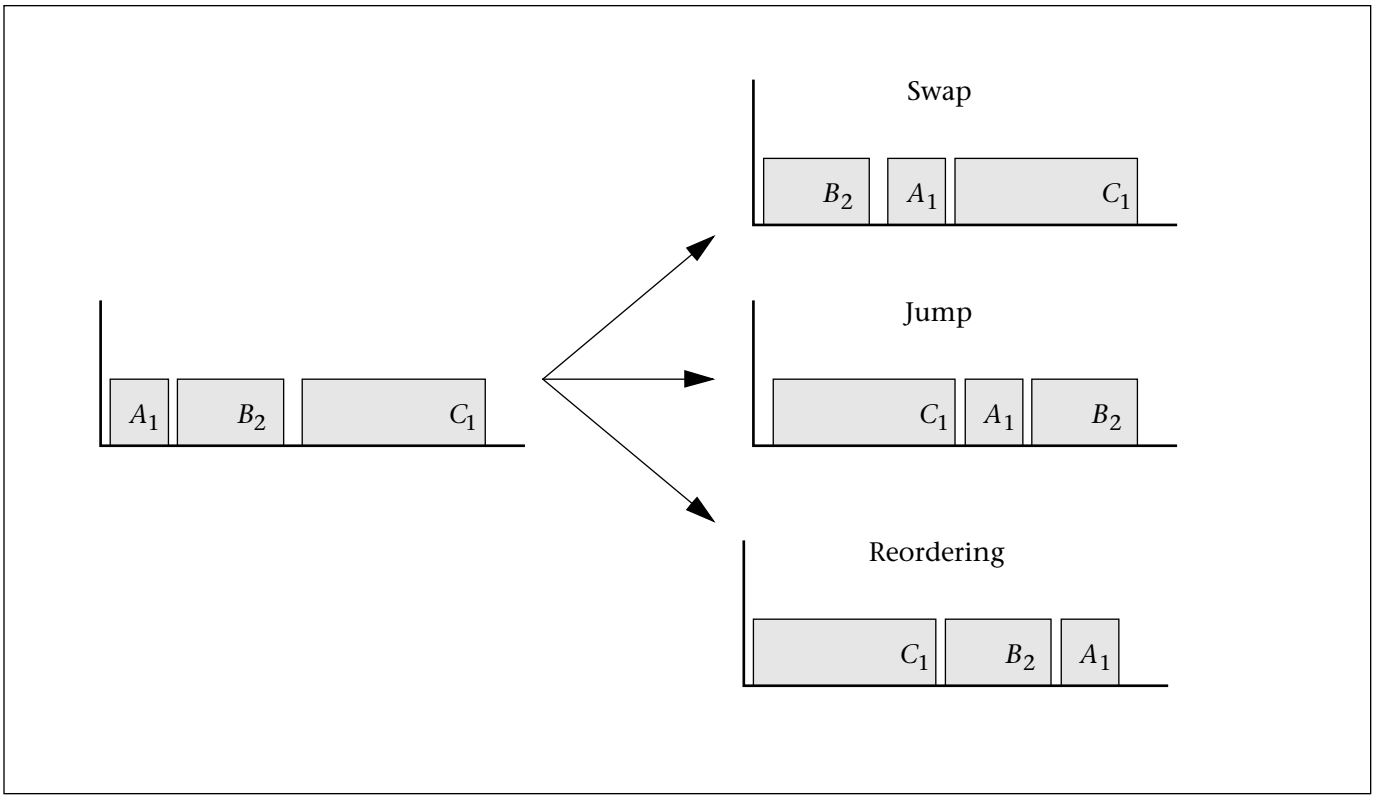

Figure 10. Examples of Three Local Search Moves.

ORR-FSS, they were not originally conceived in this way. This characterization is an effort toward an underlying, domain-independent foundation for heuristics used in constraint-directed search. Such a foundation can lead to a deeper understanding of the existing heuristics as well as a firm basis for the creation of new heuristic-commitment techniques. However, the foundation must be justified both theoretically and empirically. Theoretically, texture measurements require a mathematical basis for both their exact calculation and their estimates. The ability to characterize various estimation techniques based on the expected error from the true measurements is a significant open question. From an empirical perspective, a basic starting point is to demonstrate that a better estimate of texture measurement (while holding the type of commitments made constant) leads to a better heuristic commitment.

There is a major need for a head-to-head comparison of the techniques for heuristic commitments. It is not at all clear how to evaluate the techniques described previously because empirical studies often embed the techniques in larger search strategies. It would seem straightforward to compare a number of these techniques (ORR-FSS, task-interval entropy, RESOURCE SLACK first-last, PCP SLACK, and randomized left justified) and hold all other components of the search strategy constant. It is more difficult, however, to compare, say, ORR-FSS meaningfully with the various local search heuristics because their formulations use different retraction strategies. This comparison is an important issue. (Recent work in the operations research literature [Hooker 1996] raises this issue, among many others, concerning the empirical study of heuristics.)

Given the two extremes in the approaches to heuristic-commitment techniques, is there a way to identify problems where one is likely to perform better than the other? Can we quickly analyze a problem and discover a structure indicating that it will be particularly amenable to a simple heuristic (in a local search strategy) or, conversely, a sophisticated heuristic in a constructive strategy?

Propagators versus heuristics: It seems possible that if a heuristic technique is good enough, there is no need for propagators: The heuristics will make the same decisions that the propagators will make. If propagators are more expensive than the heuristic technique, a trade-off is suggested between spending effort in finding and making sound decisions and just making heuristic commitments. Such a trade-off has not been observed.

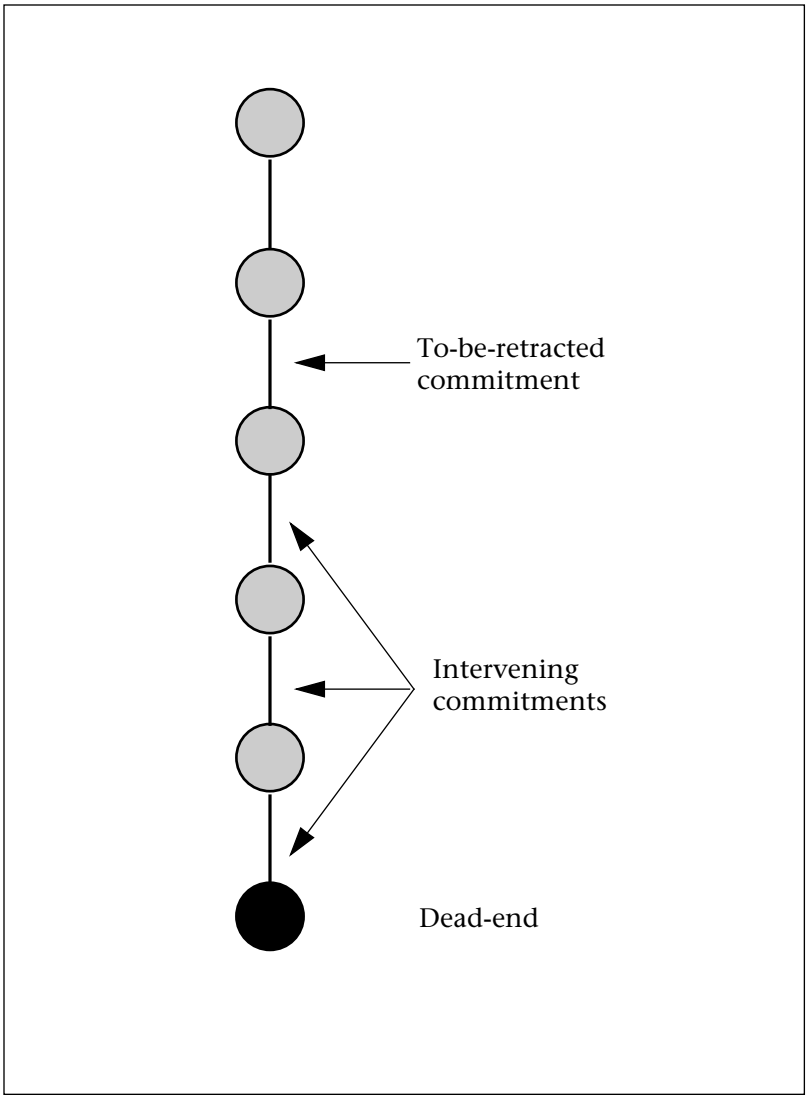

Figure 11. A Search at a Dead End.

The fact that PCP SLACK maximizes remaining slack but TIE minimizes it raises an interesting question about the intuition behind the heuristics. The slack-maximizing algorithm preserves slack in a least-commitment approach: By leaving more slack, it is more likely that a solution will be found because there is a higher probability that the to-be-scheduled activities can be scheduled. The slack-minimizing algorithm attempts to minimize the slack resulting from either sequence at a branch point. The intuition is that both branches will terminate more quickly (either in solution or failure). Recent work on branching rules for satisfiability problems (Hooker and Vinay 1995) shows that a popular heuristic works not because it increases the probability that the chosen branch contains a solution (as was conjectured) but because it simplifies the problem the most (and allows a propagator to infer more implied commitments). We interpret this work as supporting the slack-minimizing intuition.

Few of the heuristic techniques discussed here are directly applicable to more complex constraints outside the job-shop–scheduling model. If research is to address real-world problems, the more complex constraints need to be addressed.

Retraction Techniques

Commitment-retraction techniques must determine the commitment to be retracted and the fate of the intervening commitments, as illustrated in figure 11. In this section, we look at the ways various retraction techniques address these issues.

Choosing Commitments to Retract

Techniques for identifying *C*, the commitment to be retracted, fall into two general categories: (1) provable and (2) heuristic. *Provable techniques* guarantee that no solution exists in the area of the search space defined by *C* and the commitments made prior to *C*. The proof is typically necessary but not sufficient: At least *C* must be retracted; however, prior commitments might also need to be retracted to escape the dead end. In contrast, *heuristic techniques* are based on a heuristic evaluation: It is determined that *C* is likely to be the cause of the dead end.

It should be noted that there is a distinction between provability and completeness. It is possible to have a retraction technique, such as limited discrepancy search (see discussion later), that is complete but not provable, that is, that retracts commitments even though there might be a solution in the subspace. To maintain completeness, the subspace is revisited later in the search if no solution has been found elsewhere.

Provable Retraction Provable retraction appears reasonable: Until it can be proved that there is no solution given *C* and the commitments preceding *C*, the search should concentrate on the states in that area.

The simplest scheme is *chronological retraction:* The most recent commitment is retracted. Clearly, because the search space under the most recent commitment contains one state and it is a dead end, we can retract the most recent commitment without missing a solution. However, it might be that a commitment made much earlier in the search is responsible for the dead end that has only been discovered now. Chronological retraction will exhaustively search the subspace below this wrong com-

mitment before finding and retracting it.

If chronological retraction is to be improved on and provability still maintained, it is necessary to exploit the situation where chronological retraction does too much work. For example, assume some set of commitments C_1 leads to a state s_1, and then a second set of commitments C_2 leads from state s_1 to state s_2. Further, suppose that in s_2, it is discovered that none of the values for variable V are consistent with the current search state. It might be the case that had we decided to make commitments involving V at state s_1, we would have discovered precisely the same dead end. In other words, the commitments in C_2 do not contribute to the dead end. Chronological retraction will revisit all the search states between s_1 and s_2 and exhaustively try all the possible commitments before returning to a state prior to s_1, where, finally, it might be possible to escape the dead end. All provable retraction techniques rely on some mechanism to identify the most recent state at which it is possible to escape the dead end.

Full dependency-directed backtracking (Stallman and Sussman 1977) is the most extreme of these methods because it ensures that the correct state will be identified. The time and space complexity of maintaining the dependency information is in the same order (exponential) as the search itself. Other methods (for example, backjumping [Gaschnig 1978], conflict-directed backjumping [Prosser 1993], graph-based backjumping [Dechter 1990], dynamic backtracking [Ginsberg 1993]) use various techniques to prove necessity rather than sufficiency: It is necessary to jump back at least as far as the identified state, but it might be that subsequent search shows that the jump was not sufficient to escape the dead end. The intervening commitments are not causes of the dead end, so no solutions will be missed. Empirically, these techniques are able, in the average case, to make significant improvements over chronological retraction.

Although many of these techniques show good average time performance on CSPs, few have been applied to scheduling where the most common method of provable retraction is chronological.

Heuristic Retraction The major difficulty with provable retraction is that search can get stuck in a subspace that takes so long to prove that it does not contain a solution that the problem cannot be solved in a reasonable amount of time. It might be better to be more opportunistic in the selection of commitments to be retracted and, therefore, move more freely in the search space. The problem with heuristic retraction is that it can abandon completeness (that is, there is no guarantee that the strategy will find a solution or prove that none exists) and that multiple search states can be rediscovered a large number of times.

The simplest heuristic-retraction technique is to restart the search whenever a dead end is found. This technique is called *iterative sampling*, or restart. Provided the heuristic-commitment component has some randomness, restarting the search is likely to explore new states. Unless the search space is fairly dense in terms of solutions, however, incompleteness can be damaging. Nonetheless, there has been successful scheduling done using simple iterative sampling on a set of job-shop–scheduling problems (Crawford and Baker 1994).

It is easy to adapt iterative sampling to produce bounded chronological retraction with restart (Nuijten 1994). When a dead end is found, a bounded number of chronological retractions are performed to investigate the neighborhood of the dead end. If no solution is found, it is heuristically concluded that there is no solution in the area, so the search is restarted. Good results with this technique as part of a full strategy were discussed previously. A similar retraction technique, the incomplete backjumping heuristic (IBH), is proposed in Sadeh, Sycara, and Xiong (1995). After a bounded number of chronological retractions, the retraction returns to the root commitment. Sadeh's heuristic (ORR-FSS) does not have a random component; so, rather than restart, the search selects the next-best commitment in the initial search state. Empirical results indicate that IBH is particularly effective where a dead end is the result of a complex interaction among activities on more than one resource.

Limited discrepancy search (LDS) (Harvey 1995; Harvey and Ginsberg 1995) maintains completeness at the cost of significant effort spent in the rediscovery of search states that have already been visited. If the search exhausts the entire search space, LDS visits polynomially more search states than chronological retraction (Korf 1996). LDS is based on three ideas: First, with good heuristics, the solution is expected in states where the heuristic is wrong a limited number of times. Second, heuristics are more likely to be wrong earlier in the search. Third, a wrong commitment early in the search has more impact (the "early mistake" problem).

Based on these intuitions, LDS follows the heuristic during its first probe into the search tree until arriving at a dead end. The search is restarted, and the probes investigate all paths in the search tree with as few as one search state where the heuristic is ignored: all paths

The major difficulty with provable retraction is that search can get stuck in a subspace that takes so long to prove that it does not contain a solution that the problem cannot be solved in a reasonable amount of time.

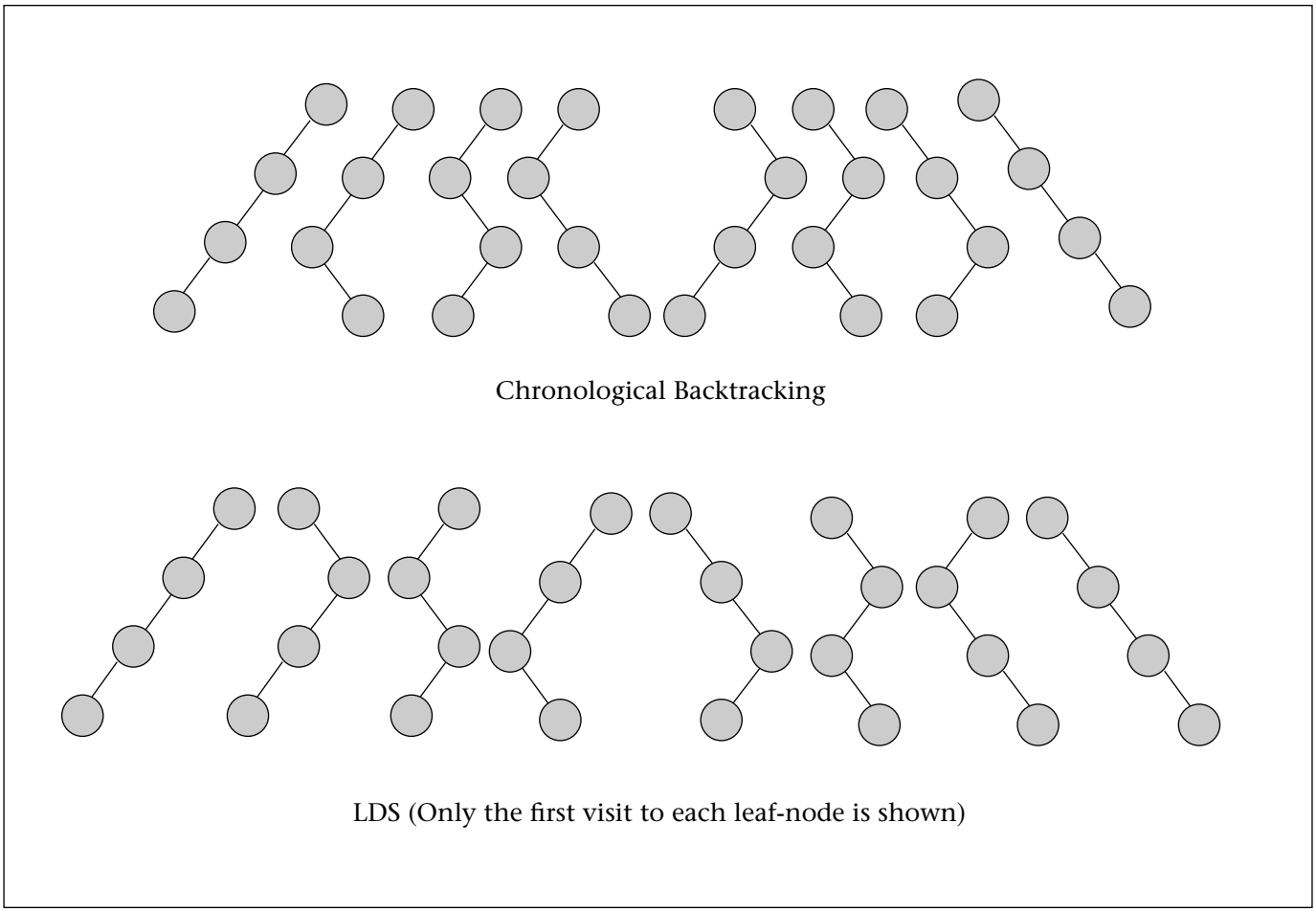

Figure 12. A Comparison of Traversals of the Search Space for a Binary Tree of Depth 4.

with discrepancy level 1. Discrepancies early in the search are explored first. In the second phase of the search, the first path (the one where there were no discrepancies) will be rediscovered. The discrepancy level is incremented every time all the search paths to the current level are exhausted. Each iteration with a particular discrepancy level will rediscover all the search states found at all previous levels.

Figure 12 displays the order in which chronological backtracking and LDS traverse the same search tree. We use the convention that the commitment represented by the left branch at each node represents the commitment chosen by the heuristic. As noted, at iteration level x, LDS rediscovers all the search states that were visited in previous iterations to $x - 1$.

The weakness of rediscovering search states is overcome in improved LDS (ILDS) (Korf 1996). ILDS generates more states than chronological retraction, but at worst, the number of ILDS states is bounded by a linear factor of the number of chronologically generated states.

Empirical evaluation of LDS on job-shop–scheduling problems (with a partially randomized heuristic-commitment technique) shows significant gain over chronological retraction and iterative sampling and comparable results to bounded backtracking with restart. The best performance was attained with a combination of LDS with bounded backtracking (Harvey 1995).

At the extreme of heuristically guided retraction are the local search methods that move in the search space with few limitations. The retraction is completely driven by the heuristics. If the search is free to follow the gradient defined by the neighborhood function, however, it is likely to discover a local optima from which it cannot move: All neighboring states are worse than the current one, yet the current one is not a global solution. A number of techniques to escape local optima have been suggested, such as simulated annealing (Zweben et al. 1994, 1993; Kirkpatrick et al. 1983), tabu search (see earlier), genetic algorithms (see earlier), and the shuffle technique (Baptiste, Le

Pape, and Nuijten 1995). In the shuffle technique, each commitment is retracted with some probability. The nonretracted commitments then form a new partial solution from which forward search can continue.

In general, local search with such escape techniques has been shown to perform well. However, as we discuss later, these techniques are often treated as abstract frameworks for algorithms rather than as algorithms themselves; so, comparison between local search algorithms specifically crafted for a benchmark set and more general strategies is questionable.

Dealing with Intervening Commitments

When the to-be-retracted commitment has been identified, there are three encompassing ways to deal with commitments made in states between the dead end and the state where the to-be-retracted commitment was made: (1) retract all, (2) retract some, or (3) retract none.

Retract All The most principled way to deal with intervening commitments is to retract them all. The intuition for this approach is that each commitment depends on all commitments made previously. If the search is jumping back to retract a commitment, all the commitments that were made subsequently are retracted because their justification no longer exists. Iterative sampling or restart, dependency-directed backtracking, the host of provable retraction techniques (except dynamic backtracking), and LDS all follow this method.

Bounded chronological backtracking with restart is a special case because for some backtracks (that is, when performing chronological backtracking), there are no intervening commitments to retract, but for other backtracks (that is, restart), all the intervening commitments are retracted.

Retract Some Assuming that the justification for a commitment no longer exists simply because a previous commitment is retracted might be too conservative. The retracted commitment might have had no bearing on some of the intervening commitments: By retracting an intervening commitment, the search is discarding information that must be rediscovered later when precisely the same commitment is made again. The advantages of not having to rediscover search information are strongly argued in Ginsberg (1993) and used as motivation for dynamic backtracking.

In dynamic backtracking, limited dependency information (used to choose the to-be-retracted commitment in backjumping) is cached so that it is not necessary to make the assumption that all intervening commitments depend on the retracted commitment. In fact, only those intervening constraints that might have depended (based on the cached information) on the retracted commitment are retracted. Empirical evidence (Ginsberg 1993) shows significant improvement over backjumping on the CSPs tested. Interestingly, further evidence (Baker 1994) shows that in some situations, dynamic backtracking can be exponentially worse than chronological backtracking. The intuition is that dynamic backtracking changes the heuristic ordering of commitments as the search proceeds. This ordering has significant impact on the search, and therefore, modifying it can lead to highly inefficient behavior. From another perspective, this intuition can be restated as follows: Although the intervening commitments do not directly depend on the to-be-retracted commitment, the state created by the to-be-retracted commitment leads by way of the heuristic to the intervening components. Given the retraction, the heuristic might well guide the search in a completely different direction that would not result in the intervening commitments being made at all. Although the intervening commitments are not directly dependent on the retracted commitment, they are dependent by way of the heuristic component.

Retract None Finally, in typical local search algorithms such as hill climbing, tabu, simulated annealing, and shuffle, none of the intervening commitments are retracted. The intuition here is that if they actually do need to be retracted, subsequent iterations will determine the retraction that must be done.

The most principled way to deal with intervening commitments is to retract them all.

Open Issues

A number of retraction techniques can be characterized by the cache of information that is used to select the commitment to retract or determine the fate of the intervening commitments. Dynamic backtracking and tabu search begin to look similar when it is realized that the contents and size of the cache are the main differences, so, too, for dependency-directed backtracking (with an exponential size cache) compared with restart and LDS (with caches of zero size). Can we use the size and content of the cache to construct new retraction techniques with varying trade-offs between the completeness and heuristic responsiveness (Havens 1997)?

There would seem to be a middle ground between local search algorithms and, say, LDS, in terms of the ability to respond heuristically. For example, it would be interesting to record a confidence with which all commitments are made and, when a dead end is reached, use the

confidences to identify the commitment to retract.

Three examples of retraction techniques discussed previously are actually combinations of other retraction techniques: (1) bounded chronological backtracking with restart, (2) IBH, and (3) bounded LDS. Furthermore, as we see later, IBH has successfully been combined with other retraction techniques. Are there other retraction techniques that can be combined to improve performance over the individual techniques by themselves?

Is the completeness of the retraction technique relevant? In a large problem, it is practically impossible to explore more than a small percentage of the search space, so why should we care that eventually our search will cover the entire space?

Is the completeness of the retraction technique relevant? In a large problem, it is practically impossible to explore more than a small percentage of the search space, so why should we care that eventually our search will cover the entire space?

Interactions among Components

In the previous sections, we concentrated individually on each component. The underlying assumption in much of the analysis is that instances of a component can be compared with each other while the other components are held constant. Although we believe isolated component comparison to be possible and useful, we also expect that there will be significant interactions among the components. A particular heuristic that works well with, for example, a tabu search retraction component might perform poorly with LDS. Conversely, a heuristic might perform well with LDS and poorly with tabu search.

Given that the components have not previously been defined as such in the literature, it is not to be expected that much work has addressed interactions of scheduling components. Nonetheless, it is possible to examine some of the existing work for insight.

Heuristic-Commitment Techniques and Propagators

To our knowledge, the only empirical work that speaks to the interaction between heuristic-commitment techniques and propagators is that done by Nuijten (1994). As noted previously, it was found that the SOLVE strategy significantly outperformed the ORR-FSS heuristics with edge finding and chronological backtracking. Given the different backtracking techniques, however, this result should not be interpreted to indicate that the sophisticated heuristics and propagators are not complementary. Indeed, Nuijten indicates that it might be the restart retraction that leads to the disparate results.

Our intuition is that heuristics and propagators can be complementary because the propagators find sound commitments that improve the information on which the heuristics are calculated. This intuition remains to be tested, although there has been work that recognizes the possibility and formulates heuristics specifically to take advantage of the power of the propagators. In Baptiste, Le Pape, and Nuijten (1995), it is noted that if the heuristic-commitment component schedules activities in a particular order (say earliest to latest), the propagator will make more inferences than if no such order is followed. This technique is extended in the TIE heuristic (Caseau and Laburthe 1995): One justification for the minimizing slack is that a smaller slack allows the propagators to find more implied commitments. Unfortunately, no work was done to isolate components and test the extent to which it is the combination that leads to the performance results.

Heuristic-Commitment Techniques and Retraction

Given a particular retraction technique, is it better to use a more or less computationally expensive heuristic-commitment technique? Given a heuristic-commitment technique, is it better to use a more systematic retraction technique or one more like those used in local search? It has been shown, using GERRY, that in tightly constrained, smaller problems, a more informed (that is, more complex and more powerful) heuristic performs better, but in larger, more loosely constrained problems, a simpler heuristic performs better (Zweben et al. 1993). This work should be followed up and extended for different variations of heuristic-commitment and retraction techniques.

The learning ordering from failure (LOFF) (Sadeh, Sycara, and Xiong 1995) retraction technique proposes an explicit interaction between the retraction and heuristic-commitment components. LOFF is a variation on chronological retraction that overrides the default activity-ordering heuristic: The most recent commitment is retracted, and the activities that were without any possible value are placed in a stack. The heuristic-commitment technique then assigns the activities on the stack before returning to the default activity-ordering heuristic. The order in which activities are assigned is modified based on information from retraction. It is not clear how LOFF performs on its own because it is evaluated as combined with either dynamic consistency enforcement (see later) or dynamic consistency enforcement and IBH.

Propagators and Retraction

A propagator finds commitments that are implied by the search state. It is contradictory to retract a commitment found by a propagator unless the search state in which the commitment was made no longer exists. In other words, it is incorrect to ever select an implied commitment as the commitment to retract from a dead-end state.

More interesting is what to do with implied commitments when retracting a prior commitment. Existing work with propagators has simply retracted all the implied constraints (and, indeed, all the intervening constraints whether implied or not). However, given that propagators infer commitments based on specific criteria, one could imagine caching the context of an implied commitment. On retracting a prior commitment, an implied commitment only need be retracted if its context no longer exists. This technique is the same that is used in dynamic backtracking; however, it might be that the decision to retract or not will be of higher quality because there is a specific context in which the commitment was made. Of course, the question of the content and size of the cache might well have bearing on the usefulness of this idea.

Sadeh et al. (1995) present dynamic consistency enforcement (DCE) where, when a dead end is detected, a subset of "dangerous" activities that are believed to cause the dead end (because of their competition for a resource) are identified. Commitments are retracted in chronological order while a *k*-consistency propagator (Freuder 1982) is run on the dangerous activities (in the experiments reported, $k = 4$). Commitment retraction continues with newly unscheduled activities being inserted into the dangerous group if appropriate until such time as *k*-consistency is established on each group. The intuition behind DCE is that dead ends arise because of the lack of consistency; however, it is too expensive to use the propagator on the entire graph. Rather, consistency should be enforced selectively among those activities that are identified as the likely causes of the dead end. DCE can be combined with LOFF by placing the dangerous activities on the stack to be assigned first after the propagator is run. Similarly, DCE and LOFF can be further combined with IBH by allowing a bounded number of DCE and LOFF retractions before backjumping to the initial commitment and selecting the next best assignment. In this way, DCE, LOFF, and IBH embody interaction both between the commitment-retraction technique and the heuristic-commitment technique and between the propagator and the commitment retraction. Empirical results show significant performance gains over chronological backtracking as well as a form of dependency-directed backtracking (second-order deep learning).

Open Issues

We expect significant interactions among the various components of a scheduling strategy. For example, we noted previously the conjecture that a particular form of heuristic commitment would increase the pruning abilities of the propagator. Similarly, DCE and LOFF depend on such interactions. The identification of such interactions in existing algorithms, as well as the creation of new components that depend on such interactions, remains to be investigated.

A further issue with respect to interactions is the robustness of formulating an entire strategy to specifically take advantage of a particular propagator. The performance of the strategy then depends significantly on the propagators: If there is a class of problems where the propagators do not perform well, it is unlikely that the strategy will be of any use.

Comparing Scheduling Strategies

The comparison of individual components is crucial for developing an understanding of both problems and solution techniques. We should not lose sight of the fact, however, that it is the performance of the overall strategy that solves (or fails to solve) the problem. We now look at comparing scheduling strategies as a whole. First, we comment on the difference that has traditionally been seen between constructive and local search algorithms and then turn to more empirical questions: direct comparison of strategies and the possibility of combining them.

Constructive versus Local Search

Traditionally, constructive search and local search algorithms have been viewed as quite different. *Local search* deals with an assigned, inconsistent state, and the heuristic-commitment techniques focus on repair. In contrast, *constructive algorithms* examine the unassigned activities to make new assignments. We believe that the key difference between constructive and local search algorithms, however, is not the heuristic-commitment techniques but the retraction techniques: how the to-be-retracted commitments are identified and how the intervening commitments are handled.

Although some modification will be necessary, there does not appear to be any principled reason why heuristic-commitment techniques typically used in constructive search cannot be

Traditionally, constructive search and local search algorithms have been viewed as quite different.

used in repair. For example, one could imagine formulating an alternative calculation of the ORR-FSS heuristic that can be used on an inconsistent, completely assigned search state. ORR-FSS can then guide, for example, the neighborhood function in a tabu search. Similarly, the lookahead used in GERRY could be modified and applied to unassigned activities in a constructive search.

We are not claiming that all traditionally constructive heuristic-commitment techniques can be used in a local search, or vice versa. Rather, we claim that the fundamental difference in the approaches is in the retraction technique and that viewing heuristic-commitment techniques as constructive or local search techniques is unnecessarily limiting.

Is there any advantage to be gained from combining different scheduling strategies? For example, can we run a constructive algorithm for a bounded time (or number of backtracks) and then switch to a local search algorithm to attempt to improve the situation?

Head-to-Head Comparisons of Strategies

Little work has been done in the comparison of scheduling strategies, and that which has been done is far from convincing because of at least two problems: (1) the scheduling problems used as a basis for comparison and (2) the extent to which algorithms have been tuned for specific problem sets. (Both of these problems [and others] have been discussed in the context of the empirical study of heuristics [Hooker 1996, 1993]).

The first problem results from the fact that different researchers have their own sets of problems, and in the past, there was little cross-validation of scheduling techniques. For example, GERRY performed on a set of space shuttle scheduling problems, but MICROBOSS had its own set of job-shop problems. Recently, problem benchmarks have arisen: First, the MICROBOSS problems were informally accepted and, more recently, the operations research library problems (Beasley 1990). (Although the operations research library problems have existed for a number of decades as a benchmark set in the operations research community, they have only recently been used in constraint-directed scheduling work.) If the goal of the research is to develop a scheduler that can be applied broadly, it is difficult to extrapolate from results on these benchmarks. Indeed, with the exception of the MICROBOSS problems, it is unclear that any work has been done in examining the characteristics of scheduling problems.

The second, and perhaps more critical, issue is the tuning of algorithms. Although a tabu algorithm currently displays close to the best performance of the operations research library benchmark problems (Vaessens, Aarts, and Lenstra 1994), the inventors point out that they have constructed it specifically for this problem set. It is questionable what general understanding can be gained from comparing tuned tabu algorithms with, for example, the SOLVE algorithm on precisely those problems that the tabu is tuned on.

That being said, a large study of local search techniques (Vaessens, Aarts, and Lenstra 1994), together with SOLVE (although without recent task-interval work from Caseau and Laburthe [1995]) and Baptiste, LePape, and Nuijten [1995]), concludes that an instantiation of tabu search is the choice on the operations research library set of benchmark problems. Although this study is likely to be outdated, it does provide an interesting snapshot comparison.

Combining Strategies

Is there any advantage to be gained from combining different scheduling strategies? For example, can we run a constructive algorithm for a bounded time (or number of backtracks) and then switch to a local search algorithm to attempt to improve the situation?

Davis (1994) looked at running the ORR-FSS heuristic until a bound on the number of backtracks was reached. The scheduler then switched to the MINCONFLICTS strategy. Three intuitions formed the basis for this test. First, a diversity is introduced to the search: Different strategies will explore different regions. Second, if the first strategy reaches the backtrack bound, it is unlikely to find a solution in a reasonable amount of time. Third, it has been demonstrated that MINCONFLICTS requires a relatively good starting point to be successful. ORR-FSS is likely to find such an initial solution. Empirical evidence, however, showed that solutions were found more often by continuing to use ORR-FSS than by switching to MINCONFLICTS. The explanation for these results stems from underlying similarities between the two search style. Similar texture measurements were used to guide both ORR-FSS and MINCONFLICTS. ORR-FSS uses contention and reliance to identify the unassigned activities with the highest demand on a highly contended-for resource, but MINCONFLICTS uses the activities that have the most resource conflicts with other activities. These are two measures of the same underlying phenomenon: the competition among activities for a resource reservation. In addition, both ORR-FSS and MINCONFLICTS make small-granularity microcommitments: the assignment of a start time to an activity. Because of this similar granularity, a state from which ORR-FSS is not able to quickly move to a solution tends to correspond to a state from which MINCONFLICTS cannot quickly move to a

solution. When reaching the commitment bound, the constructive search is making and retracting microcommitments without much success. In changing to local search, MINCONFLICTS made the same granularity of commitment as ORR-FSS and was limited to accepting new states of lower (or equal) cost. In the starting state for MINCONFLICTS, there was often no possibility of making a microcommitment that resulted in a lower-cost state. What was necessary was a macrocommitment (for example, a set of activity reassignments) that changed a number of the commitments in a single step.

Another attempt at combining strategies to minimize makespan was recently done in Baptiste, LePape, and Nuijten (1995). The first strategy was a heuristic one that used edge finding, the RESOURCE SLACK first-last heuristic, and the shuffle procedure as a retraction technique. After a number of iterations of this strategy, the best solution (smallest makespan) was used as the initial upper bound in a branch-and-bound strategy (using edge finding, RESOURCE SLACK first-last, and chronological retraction). The results showed a significant speedup in finding the optimal solution over simply using branch and bound to find an initial upper bound on the schedule makespan.

Open Issues

If we are trying to discover algorithms that can be applied without tuning across a large class of problems, the concentration on a set of benchmark problems that are simply hard, rather than characteristic of classes of problems, is misguided. There is a need to investigate and classify scheduling problems to test and compare algorithms on a representative set.

The existence of interactions among scheduling strategies is also an open question. It seems reasonable that a problem (or problem class) for which a particular strategy does poorly might be amenable to a different strategy. Conversely, when a particular strategy becomes trapped, another strategy is not necessarily more likely to escape. However, one can conceive of situations where exactly the opposite is true: No strategy performs well, but one strategy starting from a state where another failed results in better performance than either algorithm alone. Empirical work is needed to assess these intuitions.

Without a framework such as the one we are proposing, it is difficult to develop an understanding of the performance of scheduling algorithms. Insight into the reasons for performance results cannot easily be developed if the algorithms are unrelated black boxes. By providing structure to a scheduling strategy, the framework enables the attribution of performance results to specific components, allowing the beginnings of an empirical understanding of the algorithms.

Conclusions

In this article, we introduced a constraint-directed–search framework. The framework rests on assertion and retraction of commitments as primary search operators and defines three nontrivial components: (1) propagators, (2) heuristic-commitment techniques, and (3) commitment-retraction techniques.

Rather than illustrating the framework with general constraint-directed–search research, we concentrated on constraint-directed scheduling and showed how a variety of scheduling algorithms can be conceptualized within the framework. The analysis of instances of scheduling strategies and components of the framework found in the literature led to the identification of a number of research issues. Many of the issues raised surround the need for a firm empirical foundation to understand both the problems and solution techniques.

Our principal conclusion from the application of the framework to constraint-directed scheduling is the need for an empirical foundation for scheduling research.

Our principal conclusion from the application of the framework to constraint-directed scheduling is the need for an empirical foundation for scheduling research. Little is known about how particular instances of scheduling components compare over a wide range of problem characteristics. There is an even smaller body of knowledge concerning why algorithms and their components perform as they do on particular problem sets. Our framework is offered as a structure to allow comparisons and evaluations of the components of scheduling algorithms as well as the generation of hypotheses concerning the underlying reasons for algorithm behavior. The conceptualization of different scheduling strategies within our framework allows a first (nonempirical) comparison and then experimentation through systematic variation of scheduling components. Not only will this process lead to a better understanding of the strategies but also the creation of novel strategies from simple-minded combination and recombination of components; the need to create new strategies to test hypotheses about the performance of existing strategies; and, once a better understanding has been achieved, more principled creation of strategies from the understanding itself.

Acknowledgments

This research was funded in part by the Natural Science and Engineering Research Council of Canada, Numetrix Limited, the Institute for

Robotics and Intelligent Systems Research Network, the Manufacturing Research Corporation of Ontario, and Digital Equipment of Canada. Thanks to Scott Hadley, Ioan Popescu, Rob Morenz, and Andrew Davenport for comments on, and discussion of, earlier drafts of this article.

References

Applegate, D., and Cook, W. 1991. A Computational Study of the Job-Shop Scheduling Problem. *ORSA Journal on Computing* 3:149–156.

Baker, A. 1994. The Hazards of Fancy Backtracking. In Proceedings of the Twelfth National Conference on Artificial Intelligence, 288–293. Menlo Park, Calif.: American Association for Artificial Intelligence.

Baker, K. 1974. *Introduction to Sequencing and Scheduling*. New York: Wiley.

Baptiste, P.; Le Pape, C.; and Nuijten, W. 1995. Constraint-Based Optimization and Approximation for Job-Shop Scheduling. Paper presented at the IJCAI-95 Workshop on Intelligent Manufacturing Systems, 22–25 August, Montreal, Canada.

Beasley, J. E. 1990. OR-Library: Distributing Test Problems by Electronic Mail. *Journal of the Operational Research Society* 41(11): 1069—1072.

Burke, P., and Prosser, P. 1994. The Distributed Asynchronous Scheduler. In *Intelligent Scheduling,* eds. M. Zweben and M. Fox, 309—339. San Francisco: Morgan Kaufmann.

Carlier, J., and Pinson, E. 1994. Adjustment of Heads and Tails for the Job-Shop Problem. *European Journal of Operational Research* 78:146–161.

Carlier, J., and Pinson, E. 1989. An Algorithm for Solving the Job-Shop Problem. *Management Science* 35(2): 164–176.

Caseau, Y., and Laburthe, F. 1996. Cumulative Scheduling with Task Intervals. In *Proceedings of the Joint International Conference and Symposium on Logic Programming*. Cambridge, Mass.: MIT Press.

Caseau, Y., and Laburthe, F. 1995. Improving Branch and Bound for Job-Shop Scheduling with Constraint Propagation. In Proceedings of the Eighth Franco-Japanese Conference CCS'95, Brest, France.

Caseau, Y., and Laburthe, F. 1994. Improved CLP Scheduling with Task Intervals. In *Proceedings of the Eleventh International Conference on Logic Programming,* 369–383. Cambridge, Mass.: MIT Press.

Cheng, C., and Smith, S. F. 1997. Applying Constraint-Satisfaction Techniques to Job-Shop Scheduling. *Annals of Operations Research* (Special Volume on Scheduling: Theory and Practice) 70:327–378.

Collinot, A., and Le Pape, C. 1987. Controlling Constraint Propagation. In Proceedings of the Tenth International Joint Conference on Artificial Intelligence, 1032–1034. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.

Crawford, J. M., and Baker, A. B. 1994. Experimental Results on the Application of Satisfiability Algorithms to Scheduling Problems. In Proceedings of the Twelfth National Conference on Artificial Intelligence, 1092–1097. Menlo Park, Calif.: American Association for Artificial Intelligence.

Davis, E. D. 1994. ODO: A Constraint-Based Scheduler Founded on a Unified Problem-Solving Model. Master's thesis, Enterprise Integration Laboratory, Department of Industrial Engineering, University of Toronto.

Dechter, R. 1990. Enhancement Schemes for Constraint Processing: Backjumping, Learning, and Cutset Decomposition. *Artificial Intelligence* 41:273–312.

Dechter, R.; Dechter, A.; and Pearl, J. 1990. Optimization in Constraint Networks. In *Influence Diagrams, Belief Nets, and Decision Analysis,* eds. R. Oliver and J. Smith, 411–425. Chichester, U.K.: Wiley.

Erschler, J.; Roubellat, F.; and Vernhes, J. P. 1980. Characterizing the Set of Feasible Sequences for *n* Jobs to Be Carried Out on a Single Machine. *European Journal of Operational Research* 4:189–194.

Erschler, J.; Roubellat, F.; and Vernhes, J. P. 1976. Finding Some Essential Characteristics of the Feasible Solutions for a Scheduling Problem. *Operations Research* 24:772–782.

Fox, M. S. 1990. Constraint-Guided Scheduling—A Short History of Research at CMU. *Computers in Industry* 14:79–88.

Fox, M. 1986. Observations on the Role of Constraints in Problem Solving. In Proceedings of the Sixth Canadian Conference on Artificial Intelligence, May, Montreal, Canada.

Fox, M. S. 1983. Constraint-Directed Search: A Case Study of Job-Shop Scheduling. Ph.D. thesis, Intelligent Systems Laboratory, The Robotics Institute, Carnegie Mellon University.

Fox, M. S.; Sadeh, N.; and Baykan, C. 1989. Constrained Heuristic Search. In Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, 309–316. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.

Freuder, E. C. 1982. A Sufficient Condition for Backtrack-Free Search. *Journal of the Association for Computing Machinery* 29(1): 24–32.

Freuder, E. C. 1978. Synthesizing Constraint Expressions. *Communications of the Association for Computing Machinery* 21(11): 958–966.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: Freeman.

Gaschnig, J. 1979. Performance Measurement and Analysis of Certain Search Algorithms. Technical report, CMU-CS-79-124, Department of Computer Science, Carnegie Mellon University.

Gent, I.; MacIntyre, E.; Prosser, P.; Smith, B.; and Walsh, T. 1996. An Empirical Study of Dynamic Variable-Ordering Heuristics for the Constraint-Satisfaction Problem. In *Proceedings of the Second International Conference on Principles and Practice of Constraint Programming (CP96),* ed. E. Freuder, 179–193. New York: Springer-Verlag.

Ginsberg, M. 1993. Dynamic Backtracking. *Journal of Artificial Intelligence Research* 1:25–46.

Glover, F. 1990. Tabu Search Part II. *Operations*

Research Society of America (ORSA) Journal on Computing 2(1):4–32.

Glover, F. 1989. Tabu Search Part I. *Operations Research Society of America (ORSA) Journal on Computing* 1(3): 109–206.

Goldberg, D. 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, Mass.: Addison-Wesley.

Haralick, R. M., and Elliot, G. L. 1980. Increasing Tree Search Efficiency for Constraint-Satisfaction Problems. *Artificial Intelligence* 14:263–314.

Harvey, W. D. 1995. Nonsystematic Backtracking Search. Ph.D. thesis, Department of Computer Science, Stanford University.

Harvey, W. D., and Ginsberg, M. L. 1995. Limited Discrepancy Search. In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, 607–613. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.

Havens, W. 1997. Nogood Caching for Multiagent Backtrack Search. Paper presented at the AAAI-97 Constraints and Agents Workshop, 26 July, Providence, Rhode Island.

Holland, J. 1975. *Adaptation in Natural Systems*. Ann Arbor, Mich.: University of Michigan Press.

Hooker, J. N. 1996. Testing Heuristics: We Have It All Wrong. *Journal of Heuristics* 1:33–42.

Hooker, J. N. 1993. Needed: An Empirical Science of Algorithms. Technical report, Graduate School of Industrial Administration, Carnegie Mellon University.

Hooker, J. N., and Vinay, V. 1995. Branching Rules for Satisfiability. *Journal of Automated Reasoning* 15:359–383.

Kirkpatrick, S.; Gelatt, C. D.; and Vecchi, M. P. 1983. Optimization by Simulated Annealing. *Science* 220:671–680.

Korf, R. 1996. Improved Limited Discrepancy Search. In Proceedings of the Thirteenth National Conference on Artificial Intelligence. Menlo Park, Calif.: American Association for Artificial Intelligence.

Kumar, V. 1992. Algorithms for Constraint-Satisfaction Problems: A Survey. *AI Magazine* 13(1): 32–44.

Le Pape, C. 1994a. Constraint-Based Programming for Scheduling: An Historical Perspective. Paper presented at the Operations Research Seminar on Constraint-Handling Techniques, London, United Kingdom.

Le Pape, C. 1994b. Using a Constraint-Based Scheduling Library to Solve a Specific Scheduling Problem. In Proceedings of the AAAI-SIGMAN Workshop on Artificial Intelligence Approaches to Modeling and Scheduling Manufacturing Processes, New Orleans, Louisiana.

Le Pape, C., and Baptiste, P. 1996. Constraint-Propagation Techniques for Disjunctive Scheduling: The Preemptive Case. Paper presented at the Twelfth European Conference on Artificial Intelligence, 11–16 August, Budapest, Hungary.

Lhomme, O. 1993. Consistency Techniques for Numeric CSPs. In Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence, 232–238. Menlo Park, Calif.: International Joint Conference on Artificial Intelligence.

Little, J.; Murty, K.; Sweeney, D.; and Karel, C. 1963. An Algorithm for the Traveling Salesman Problem. *Operations Research* 11(6): 972–989.

Mackworth, A. K. 1977. Consistency in Networks of Relations. *Artificial Intelligence* 8:99–118.

Minton, S.; Johnston, M.; Philips, A.; and Laird, P. 1992. Minimizing Conflicts: A Heuristic Repair Method for Constraint-Satisfaction and Scheduling Problems. *Artificial Intelligence* 58:161–205.

Muscettola, N. 1992. Scheduling by Iterative Partition of Bottleneck Conflicts. Technical Report, CMU-RI-TR-92-05, The Robotics Institute, Carnegie Mellon University.

Neiman, D.; Hildum, D.; Lesser, V.; and Sandholm, T. 1994. Exploiting Metalevel Information in a Distributed Scheduling System. In Proceedings of the Twelfth National Conference on Artificial Intelligence, 394–400. Menlo Park, Calif.: American Association for Artificial Intelligence.

Nuijten, W. P. M. 1994. Time- and Resource-Constrained Scheduling: A Constraint-Satisfaction Approach. Ph.D. thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology.

Nuijten, W., and Aarts, E. 1996. A Computational Study of Constraint Satisfaction for Multiple Capacitated Job-Shop Scheduling. *European Journal of Operational Research* 90: 269–284.

Nuijten, W. P. M.; Aarts, E. H. L.; van Arp, T.; Kip, D. A. A.; and van Hee, K. M. 1993. Randomized Constraint Satisfaction for Job-Shop Scheduling. Paper presented at the IJCAI'93 Workshop on Knowledge-Based Production, Scheduling, and Control, 29 August, Chambery, France.

Prosser, P. 1993. Hybrid Algorithms for the Constraint-Satisfaction Problem. *Computational Intelligence* 9(3): 268–299.

Rosenfeld, A.; Hummel, R. A.; and Zucker, S. W. 1976. Scene Labeling by Relaxation Operations. *IEEE Transactions on Systems, Man, and Cybernetics* 6:420–433.

Sadeh, N. 1994. Microopportunistic Scheduling. In *Intelligent Scheduling*, eds. M. Zweben and M. Fox, 99–138. San Francisco: Morgan Kaufmann.

Sadeh, N. 1991. Lookahead Techniques for Microopportunistic Job-Shop Scheduling. Ph.D. thesis, Computer Science Department, CMU-CS-91-102, Carnegie-Mellon University.

Sadeh, N., and Fox, M. S. 1996. Variable and Value-Ordering Heuristics for the Job-Shop–Scheduling Constraint-Satisfaction Problem. *Artificial Intelligence Journal* 86(1): 1–41.

Sadeh, N., and Fox, M. 1989. Focus of Attention in an Activity-Based Scheduler. Paper presented at the NASA Conference on Space Telerobotics, January, Pasadena, California.

Sadeh, N.; Sycara, K.; and Xiong, Y. 1995. Backtracking Techniques for the Job-Shop–Scheduling Constraint Satisfaction. *Artificial Intelligence* 76:455–480.

Shapiro, L. G., and Haralick, R. M. 1981. Structural Descriptions and Inexact Matching. *IEEE Transac-*

tions on Pattern Analysis and Machine Intelligence 3(11): 504–519.

Smith, B., and Grant, S. 1997. Trying Harder to Fail First. Technical Report, 97.45, School of Computer Science, University of Leeds.

Smith, S. F.; Ow, P.; Matthys, D.; and Potvin, J. 1989. OPIS: An Opportunistic Factory-Scheduling System. Paper presented at the International Symposium for Computer Scientists, August, Beijing, China.

Smith, S. F., and Cheng, C. C. 1993. Slack-Based Heuristics for Constraint-Satisfaction Scheduling. In Proceedings of the Eleventh National Conference on Artificial Intelligence, 139–144. Menlo Park, Calif.: American Association for Artificial Intelligence.

Stallman, R., and Sussman, G. 1977. Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis. *Artificial Intelligence* 9:135–196.

Sycara, K.; Roth, S.; Sadeh, N.; and Fox, M. 1991. Distributed Constrained Heuristic Search. *IEEE Transactions on Systems, Man, and Cybernetics* SMC-21(6): 1446–1461.

Tsang, E. P. K. 1993. *Foundations of Constraint Satisfaction*. San Diego, Calif.: Academic.

Vaessens, R. J. M.; Aarts, E. H. L.; and Lenstra, J. K. 1994. Job-Shop Scheduling by Local Search. Technical Report, COSOR Memorandum 94-05, Eindhoven University of Technology.

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

Waltz, D. 1975. Understanding Line Drawings of Scenes with Shadows. In *The Psychology of Computer Vision,* ed. P. Winston, 19–91. New York: McGraw-Hill.

Zweben, M.; Daun, B.; Davis, E.; and Deale, M. 1994. Scheduling and Rescheduling with Iterative Repair. In *Intelligent Scheduling,* eds. M. Zweben and M. Fox, 241–256. San Francisco: Morgan Kaufmann.

Zweben, M.; Davis, E.; Daun, B.; and Deale, M. 1993. Informedness versus Computational Cost of Heuristics in Iterative Repair Scheduling. In Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence, 1416–1422. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.

J. Christopher Beck is a Ph.D. candidate in the Department of Computer Science, University of Toronto, and project manager of the Intelligent Scheduling Group, Enterprise Integration Laboratory, Department of Mechanical and Industrial Engineering, University of Toronto. He received his B.Sc. (mathematics and computing sciences) from St. Francis Xavier University and his M.Sc. (computer science) from the University of Toronto. His doctoral dissertation concerns the extension of constraint-directed–scheduling techniques to address a variety of real-world constraints, such as the production, consumption, and storage of inventory. His research interests include constraint-directed reasoning and search, scheduling, and distributed AI. His e-mail address is chris@cs.utoronto.ca.

Mark S. Fox is a professor of mechanical and industrial engineering with cross-appointments in the Department of Computer Science and the Faculty of Management Science at the University of Toronto. He is head of the Enterprise Integration Laboratory and director of the Graduate Program in Integrated Manufacturing and the holder of the National Sciences and Engineering Research Council Industrial Research Chair in Enterprise Integration. Prior to his return to Toronto, he was an associate professor of computer science and robotics and director of the Center for Integrated Manufacturing Systems of The Robotics Institute at Carnegie Mellon University. He is also a cofounder and past president of Carnegie Group Inc., a knowledge-based software company that focuses on engineering, manufacturing, and telecommunications applications. He received a B.Sc. in computer science from the University of Toronto in 1975 and his Ph.D. in computer science from Carnegie Mellon University in 1983. Fox was elected a fellow of the American Association for Artificial Intelligence (AAAI) in 1991, was a AAAI councilor, and is a cofounder of the AAAI Special Interest Group in Manufacturing. He was also elected a joint fellow of the Canadian Institute for Advanced Research and the Precompetitive Advanced Research Network. He is also a member of the Association of Computing Machinery, the Canadian Society for Computational Studies of Intelligence, the Institute of Electrical and Electronics Engineering, the Institute of Industrial Engineers, and the Society of Manufacturing Engineers. Fox's current research focuses on theories of enterprise engineering (that is, information technology for business-process engineering), constrained-directed reasoning, a unified theory of scheduling, enterprise modeling, and coordination theory. These theories are being applied to the problems of concurrent engineering, supply-chain management, and business-process reengineering. His e-mail address is msf@ie.utoronto.ca.