JOURNAL OF SCHEDULING
J. Sched. **1**, 89–125 (1998)

THE ODO PROJECT: TOWARD A UNIFIED BASIS FOR CONSTRAINT-DIRECTED SCHEDULING

J. CHRISTOPHER BECK[1], ANDREW J. DAVENPORT[2], EUGENE D. DAVIS[3] AND MARK S. FOX[1,2,*]

[1]*Department of Computer Science, University of Toronto, Toronto,Ontario, Canada M5S 3G9*
[2]*Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, Ontario, Canada M5S 3G9*
[3]*People Soft, Inc., 4305 Hacienda Drive Pleasanton, CA 94588, U.S.A.*

ABSTRACT

The ODO project is an inquiry into constraint-directed scheduling with the primary motivation of the development of a unified foundation for constraint-directed search techniques. Central to this foundation is the exploitation of the knowledge in the constraint representation, the use of commitment assertion and retraction as search operators, a generic model of scheduling strategies, and the use of texture measurements to distill constraint information for search guidance. Each of these components is discussed in-depth. The ODO framework, a commitment-based model of constraint-directed search with which many existing scheduling techniques can be modelled and implemented, is presented along with a selection of past, current, and future research using the framework. © 1998 John Wiley & Sons, Ltd.

KEY WORDS: scheduling; constraints; search; heuristics; texture measurements; propagators; backtracking

1. INTRODUCTION

The ODO project is a theoretical and empirical inquiry into techniques of constraint-directed search with a focus on constraint-directed scheduling. Begun in 1991 at the Enterprise Integration Laboratory, University of Toronto, ODO has, as its chief theoretical motivation, the investigation of the primacy of constraints not only in the representation but also in the search for a solution to scheduling problems. Constraints are not simply a knowledge representation tool that statically represent a scheduling problem. Rather, they have a significant role to play in guiding search techniques toward a solution [1]. The information represented in constraints should be actively exploited to prune the search space and to provide information to heuristic decision-making techniques. In order to investigate the role of constraints in problem solving, we have created a unified framework for constraint-directed scheduling based on the three concepts

* Correspondence to: Mark S. Fox, Department of Mechanical and Industrial Engineering, and Department of Computer Science, University of Toronto, Toronto, Ontario, Canada M5S 3G9. E-mail: msf@ie.utoronto.ca

Contract/grant sponsor: Natural Science and Engineering Research Council of Canada
Contract/grant sponsor: Numetrix Limited
Contract/grant sponsor: IRIS Research Network
Contract/grant sponsor: Materials and Manufacturing, Ontario
Contract/grant sponsor: Digital Equipment of Canada

CCC 1094–6136/98/020089–37$17.50

of the constraint graph, the assertion and retraction of commitments, and the scheduling strategy. In addition, we have identified three components of scheduling strategies: propagators, heuristic commitment techniques, and retraction techniques. The framework is both a cognitive tool and an implementational tool in that it provides a conceptual model of constraint-directed scheduling as well as the object-oriented design of our scheduling system. The primary goal of this paper is the elucidation of the unified framework and its components.

The ODO project and, indeed, the unified framework itself, are also strongly motivated by empirical research questions. The first component of our empirical motivation is a deeper understanding of existing constraint-directed scheduling techniques. With few exceptions (e.g. References 2 and 3), there has been little comparative analysis of the underlying reasons that one scheduling algorithm is better than another in a particular circumstance. In addition, there has been a tendency to view scheduling algorithms as monoliths rather than as containing components to be individually investigated. As noted earlier, we have identified three components of scheduling algorithms. The empirical comparison of instances of each component is critical for a deeper understanding of search behavior [4] and critical as well to our theoretical motivation.

The second empirical motivation for the ODO project is to address new scheduling problems. While most scheduling problems as they exist in the real world have some characteristics that resemble models investigated in the literature, most real problems contain constraints that have not been addressed in the research. This lack is less today than it was at the inception of the ODO project as a number of researchers have begun to push beyond the job-shop model (e.g. References 2, 5–9). However, much of the research still addresses narrowly defined optimization criteria with questionable real-world relevance (e.g. minimization of makespan) [4]. Extending the scope of constraint-directed techniques is also important to our theoretical motivation: we believe the exploitation of the information represented in the constraints becomes far more critical when the information represented is rich and varied, that is, when there are a wide variety of constraints. This, indeed, returns to one of the original premises for applying constraint-directed search techniques to scheduling: in the real-world, a scheduling problem is not simply the meeting of due dates, rather, it is the satisfaction of a plethora of constraints and objectives from many parts of the organization [10].

In this paper we present the ODO project, concentrating on the components of the unified framework and the research issues we have investigated within the framework. In the following section we present an introduction to constraint-directed search and scheduling. We follow this with an overview of the ODO framework and a brief introduction of the components of the framework before devoting a section to an in-depth description of the framework components and examples of scheduling algorithms as they can be modelled with the framework. Finally, we examine the research issues we have investigated within the ODO framework and briefly look at our current and future work.

2. AN OVERVIEW OF CONSTRAINT-DIRECTED SEARCH AND SCHEDULING

Constraint-Directed Search (CDS), 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 first represented, and second, represented in such a way that search techniques can make use of them for guidance.

2.1. The constraint satisfaction problem

The simplest application of constraint directed search is to the *finite constraint satisfaction problem* (CSP) [11, 12] which can be defined as follows:
Given:

(1) A set of n variables $Z = \{x_1, \ldots, x_n\}$ with discrete, finite domains $D = \{D_1, \ldots, D_n\}$.
(2) A set of m constraints $C = \{c_1, \ldots, c_m\}$ which are predicates $c_k(x_i, \ldots, x_j)$ defined on the Cartesian product $D_i \times \cdots \times D_j$. 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*.

Find:

(3) 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 hyper-edge $e \in E$. Other conceptualizations of a CSP exist, including the dual constraint graph and join graph [13].

A *consistent assignment* or *consistent valuation* of a set of CSP variables, S, is the assignment of a value to each variable in S such that all constraints in the subgraph induced by variables in S are satisfied.

In Figure 1 we present a constraint graph of a CSP modelling a small graph colouring problem. Each variable in the CSP represents a node in the graph to be coloured. Each variable (node) has a domain of three values {red, green, blue} and each constraint (edge) expresses a 'not equals' relationship.

The search for a solution to a CSP can be viewed as a traversal of the problem space consisting of all combinations of variable domain subsets. A solution is a state with a single value remaining in the domain of each variable and no unsatisfied constraints. The mechanism for the traversal of the problem space is the modification of the constraint graph by the addition and removal of constraints. The constraint graph, therefore, is an evolving representation of the search state. In the example of Figure 1, we may (heuristically) add a unary constraint that assigns $v_4 = green$. Alternatively, we may have at our disposal an algorithm that is able to infer that v_1 and v_4 must have the same colour, and therefore we introduce a binary 'equals' constraint between those two variables. Figure 2 shows a search tree for the graph colouring problem in Figure 1.

CSPs have been successfully used to model a wide range of problems, from the abstract (e.g. graph colouring [14]) to the concrete (e.g. design [15, 16]). For an excellent review of CSP solution techniques and applications, see References 12 and 17.

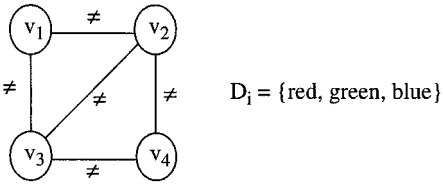


Figure 1. A small graph colouring problem represented as a CSP

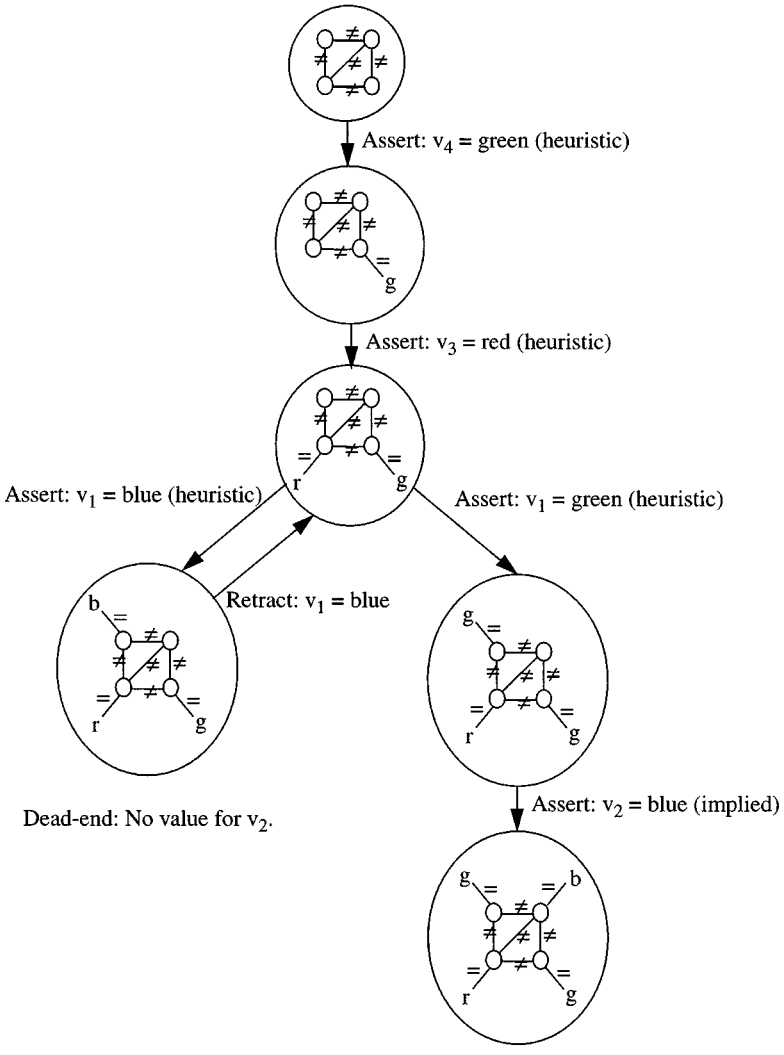


Figure 2. A possible search tree for the problem in Figure 1

A *constraint optimization problem* (COP) [12] is defined as a CSP together with an optimization function f which maps every tuple to a numerical 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 $f: S \rightarrow$ numerical 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 [10, 18–20]. Rather than satisfy all constraints and optimize f, the goal is to minimize the cost by satisfying as many constraints as possible.

2.1.1. Why constraints?

Constraint-directed search relies on two interpendent intuitions. First, the *representational intuition* states that to solve a problem, represent the relevant knowledge. Second, the search *intuition* states that to solve a problem, guide the search with that represented knowledge.

Underlying these intuitions is the *topological assumption* [1]: understanding a problem's search space will enable the creation and selection of search techniques that can efficiently navigate the space to solution.

In the context of CDS, the representational intuition results in the creation of a rich constraint representation that is able to express 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 rather provide an understanding of the structure of the problem which may be used to guide search in the problem space.

2.2. Constraint-directed scheduling

Constraint-directed scheduling is the representation of a scheduling problem and the search for a solution to it by focusing upon the constraints in the problem. Given that even simple models of scheduling (e.g. job-shop scheduling) are NP-hard [21], the search process typically depends on heuristic commitments, propagation of the effects of commitments, and the retraction of commitments. In more complex scheduling models 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 [10, 22]. In short, scheduling is a prime application area for constraint-directed search.

2.2.1. The job-shop scheduling problem

One of the simplest models of scheduling widely studied in the literature is the *job-shop scheduling problem*. The classical $N \times M$ 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 constraints in this problem:

(1) precedence constraints between two activities in the same job stating that if activity A is before activity B in the total order then activity A must execute before activity B;
(2) disjunctive resource constraints specifying that no two activities requiring the same resource may execute at the same time.

Jobs have release dates (the time after which the activities in the job may be executed) and due dates (the time by which all activities in the job must finish). In the classical decision problem, the release date of each job is 0, the global due date is D, and the goal 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 [21]. A recent survey of techniques for solving the job-shop scheduling problem can be found in Reference 23.

An example of a 3×5 job-shop scheduling problem is shown in Figure 3. In this example, there are three jobs (A, B, and C), each job has five activities (e.g. $A_1, \ldots, A_5$) and 5 resources ($R_1, \ldots, R_5$). The release date for all jobs is 0 and the due date for all jobs is D. The resource

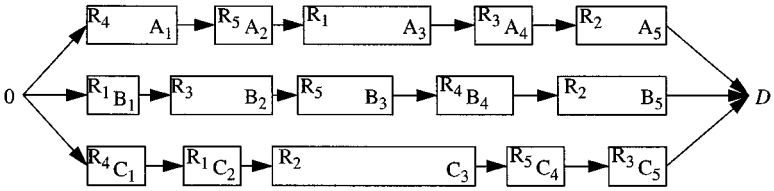


Figure 3. An example 3×5 job-shop scheduling problem

Table I. Notation

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

required by each activity is indicated in the upper-left corner and the duration, though not specified, is represented by the length of each activity. The arrows represent precedence constraints.

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 (i.e. find the schedule with the minimum D) [24], minimization of the average (or maximum) tardiness of activities (i.e. how late after their due-date activities finish), or some combination of other attributes (e.g. minimize work-in-process combined with tardiness) [10, 18, 25]. There has been little work that addresses the many complex and interacting objective functions that typically arise in real-world problems.

For an activity, A_i, and a set of activities, S, we use the notation in Table I through the balance of this paper. We will omit the subscript unless there is the possibility of ambiguity.

2.2.2. Historical perspective

A number of threads of research have contributed to modern constraint-directed scheduling. It is beyond the scope of this document 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 Reference 22 and 26 for more in-depth historical perspectives. There has been cross-fertilization among these threads as they have evolved and some work (e.g. References 27–29) spans more than one category. 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 due to Mark Fox and Steve Smith and their work on the ISS scheduler [10]. They were the first to adopt constraints as a key knowledge representation (KR) and search guidance tool for both schedule construction and revision. In particular, this thread is responsible for the use of constraints to represent scheduling problems in their full generality and for the use of the problem knowledge represented in the constraints as the main basis for heuristic decision making. Systems, directly descended from ISIS (OPIS [18], CORTES [30], MicroBOSS [25], DCHS [31]) and others which have adopted the constraint-directed philosophy (SONIA [32], DAS [33], GERRY [19, 20], MinConflicts [14], DisARM [7]), investigate a wide space of constraint representations and solution techniques.

The constraint programming thread: The constraint programming (CP) community has traditionally stressed representation while using more generic solution techniques: CP languages could typically represent problems far more complex than their solution techniques could handle. The CP thread, developing 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 as the propagation 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 constraints found in scheduling. These recent investigations have corrected the imbalance in solution power and have provided a number of impressive results [5, 6, 8, 34–36].

The operations research thread: The long(er) history of Operations Research (OR) provides a number of techniques for constraint-directed scheduling. From work which pre-dates constraint-directed scheduling itself [37, 38] to techniques which have been adopted and adapted more recently [24, 27–29], a variety of OR methods have significant impact on both the approaches to and performance of modern constraint-directed scheduling systems. Within the OR community, scheduling techniques have been developed based on mathematical programming techniques (integer programming, column generation) and local search (tabu search [39–41], genetic algorithms [42], simulated annealing [43]). Although these techniques can be very useful, one drawback is that they tend to be developed for a specific problem type (e.g. job-shop scheduling), and often cannot represent full, real world problems in all their generality.

3. AN OVERVIEW OF THE ODO FRAMEWORK

The ODO framework is a way to understand constraint-directed scheduling algorithms. As such, we believe that the existing CDS work can be understood within the framework and that this understanding allows a new perspective on that work. We are not necessarily proposing this framework at the implementation level. Depending on the choices made for the various components of the scheduling strategy, close interaction may be required and therefore an implementation may split or merge the components we have identified. That is not to say that the framework cannot also be used at the implementational level: the ODO framework is also the object-oriented architecture of the ODO implementation.

A high-level overview of the ODO framework is shown in Figure 4. At this level the objects of the framework are the *constraint graph*, the *scheduling strategy* or *policy*, and *commitments* which are asserted into and retracted from the constraint graph by the policy. This framework for CDS is an extension of the original ODO framework proposed in References 44 and 45.

The constraint graph contains a representation of the current problem state in the form of variables, constraints, and objects built from variables and constraints. A commitment is a set of constraints, variables, and problem objects that the search strategy adds to and removes from the constraint graph. The assertion and retraction of commitments are the only search operators.

A policy contains the components displayed in Figure 5. A pseudo-code representation of a policy is presented in Figure 6. The commitment assertion is trivial as it requires adding a constraint to the existing graph. The other components may require significant effort.

A *heuristic commitment technique* is a procedure that finds new commitments to be asserted. It can be divided into two components: the first performs some measurement of the constraint graph in order to distill information about the search state and the second uses this distilled information to heuristically choose a commitment to be added to the constraint graph. A *propagator* is a procedure that examines the existing search state to find commitments that are logically implied by the current constraint graph. A *retraction technique* is a procedure for identifying existing commitments to be removed from the constraint graph. The *termination criteria* is a list of user-defined conditions for ending the search. There may be many criteria: a definition of a solution (e.g. 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 CPU time, number of commitments, number of heuristic commitments, number of retractions, etc.

3.1. Why the framework

The ODO framework provides two important advantages to our research effort. The first is a cognitive model of constraint-directed scheduling: the framework represents a way to *think* about constraint-directed scheduling. Using the framework, we can more quickly understand

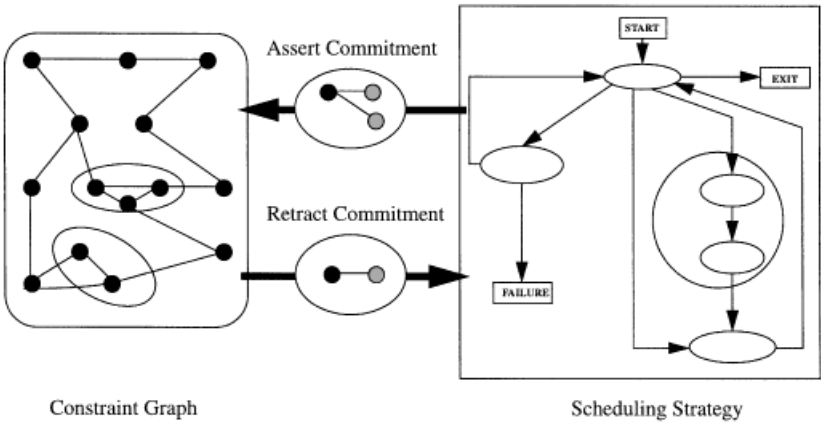


Figure 4. A high-level view of ODO framework

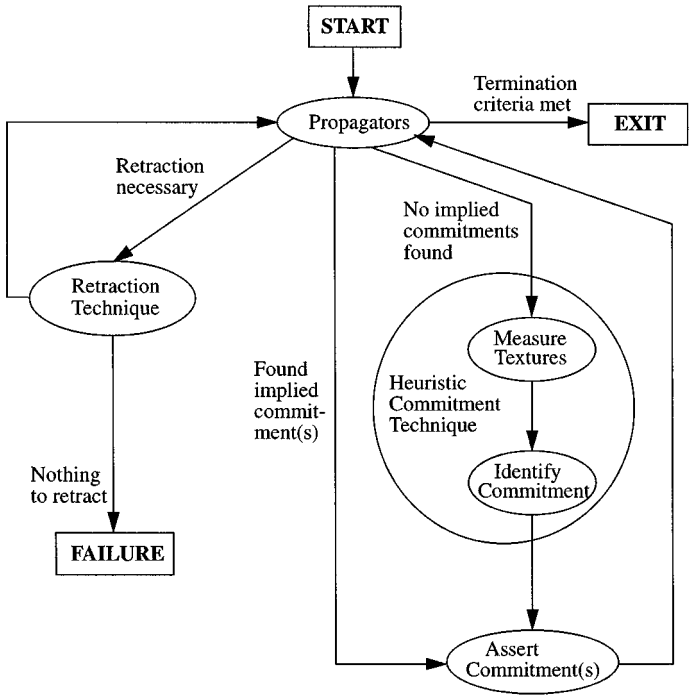


Figure 5. Schematic of a policy

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

   while(untried propagators AND
         no implied commitments found AND
         no retraction necessary)
      try next propagator

   if (retraction necessary){
      retract commitment(s)
      if (no commitments to retract)
         FAILURE
   }
   else{
      if (no implied commitments)
         measure textures
         make heuristic commitment
      assert commitment
   }
}
```

Figure 6. Pseudo-code for a policy

(and create) novel scheduling strategies and components, understand the structural similarities and differences among existing strategies, and approach new problem types.

The second advantage of the ODO framework is as an implementational model of constraint-direct scheduling. The high-level concepts of constraint graph, commitment, and scheduling strategy as well as the strategy components all have corresponding objects in our C++ implementation of the ODO scheduling shell. New propagators for example, can be created by inheriting an interface from an abstract `Propagator` class and then implementing the details of the new propagator. At run-time, then, we can specify that the new propagator (perhaps as one of a set of propagators) is used in a scheduling strategy. The architecture supports rigorous empirical comparison of the components of scheduling algorithms, allowing us to compare, for example, different heuristic commitment techniques while keeping the propagators and retraction techniques constant. It provides the ability to compose novel scheduling strategies simply from the component instances previously implemented. It also allows us to address novel problems with extensions to the constraint graph and commitment representations.

4. THE COMPONENTS OF ODO: THE CONSTRAINT GRAPH AND COMMITMENT MODEL

In this section we discuss two components of the ODO framework: the constraint graph and the commitment model. The scheduling strategy, together with its components is discussed in the following section.

4.1. The constraint graph representation

The constraint graph is the evolving representation of the search state. It is composed of components at the constraint/variable level as well as of components at the higher scheduling level. These higher level components are themselves composed of sets of variables and constraints. Examples of the lower level components include interval variables which can be assigned to a (possibly non-continuous) interval of integer values and constraints expressing various mathematical relationships (e.g., less-than, equal) among interval variables. At the higher level, the constraint graph represents, among other scheduling components, activities, Allen's 13 temporal relations [46], and resources and inventories with minimum and maximum constraints. The components of the constraint graph are not an innovation of the ODO model as most constraint-directed scheduling systems represent these concepts in some way (e.g. References 34, 36, 47 and 48).

Neither the implementation-level details of the constraint graph nor the scope of the objects represented are part of the ODO framework. Such prescriptive details and scope would unnecessarily limit the applicability of ODO to scheduling problems with well-understood constraint-based representations. One of the key aspects of constraint-directed search is the extensibility and flexibility of the representation, and therefore the ODO constraint graph is continuously evolving as our research explores new areas of application of constraint-directed scheduling.

4.2. The commitment model

In traditional, constructive, constraint-directed search, the forward movement through the search space is achieved by the assignment of a value to a variable. Given the heuristic nature of

the assignment step, it is likely (in difficult problems) to encounter a dead-end, that is, a state in which at least one variable has no values to which it can be assigned without breaking one or more constraints. At this point some form of backtracking is done: some previously made assignments are undone. Forward search then continues.

An instance of a CSP or COP can also be addressed with a local search procedure (such as tabu search and simulated annealing). Local search techniques work on sets of search states S, which may be partial or complete assignments of values to variables. A local search procedure uses a neighborhood function f to generate new search states from the current search state : $f(S) \rightarrow S'$. From S', one or more states are selected for exploration. Some analysis is typically done in each new state to decide whether it is acceptable. If so, the neighborhood function is applied to the new state and search continues. If it is not accepted, the previous state is returned to and a different neighbour is chosen.

4.2.1. Commitments, assertion, and retraction

A *commitment* is a variable, a constraint, or a set of variables and constraints, added to the constraint graph representation of the problem during search. *Assertion* of a commitment is the process of adding the problem objects in the commitment to the constraint graph. Thus commitment assertion is a state transition operator. *Retraction* of a commitment is the process of removing a commitment from the constraint graph. Like assertion, retraction is a state transition operator, resulting in a new constraint graph.

Examples of commitments in scheduling include:

(a) Assigning a start time to an activity by posting a unary 'equals' constraint (e.g. the start time of activity A is equal to 100).
(b) Posting a precedence constraint between activities (e.g. activity A executes before activity B) [49, 50].
(c) Instantiating a process plan, in response to a demand for some amount of an inventory. A process plan is a set of activities and constraints that together produce some inventory. Assertion of a process plan commitment, like the assertion of any commitment, adds these new objects to the constraint graph.
(d) Adding a new resource to the problem. It may be that part of the scheduling problem is to determine (and perhaps minimize) the number of resources used. Such a problem arises in transportation applications where it is desired to use as few trucks as possible to meet the shipment requirements. A resource, like an activity, is composed of variables and constraints that, with an assertion, are added to the constraint graph.

4.2.2. Commitments as a unification for constructive and local search

Both constructive and local search techniques can be modelled as the assertion and retraction of commitments. This is straightforward in constructive search where the assertion and the retraction are explicit. In local search, the assertion and retraction takes place in a single step. Changing the value of a variable can be modelled as an assertion (add the new commitment that assigns the variable to its new value) followed by a retraction (un-assign the variable, that is remove the commitment that assigned it to its current value). A step in the search space, for local search, therefore, is one or more assertions and retractions. Figure 7 shows a schematic example of both styles of search based on assertion and retraction of commitments.

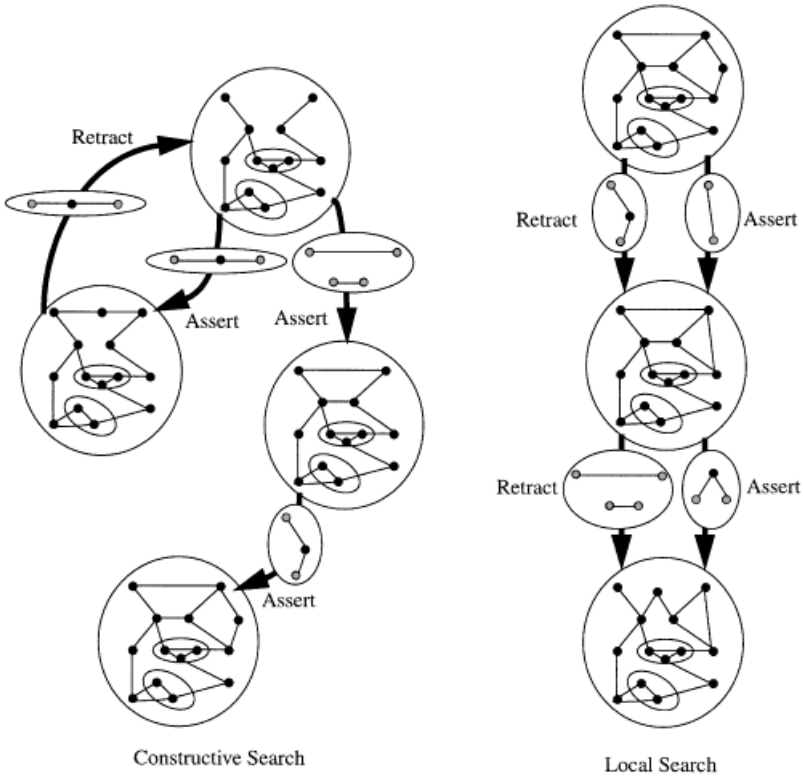


Figure 7. Constructive and local search in the commitment model

The definition and use of commitments makes no assumptions either about the form of a commitment (other than being a set of variables and constraints) or about why particular commitments are asserted or retracted. The former point is critical in modelling the variety of commitments that are used in constraint-directed scheduling while the latter point is important for the wide applicability of the commitment model. One of the major differences between local search and constructive search styles is in the reasoning that identifies the commitments to be asserted and retracted. By simply modelling the assertion and retraction of commitments without specifying the ways in which the commitments are identified, we are able to account for both constructive and local search techniques.

4.2.3. Research issues

The use of the assertion and retraction of commitments as the sole search operators raises of a number of research issues:

1. Constructive and local search are generally viewed as having different strengths and weaknesses. While constructive search, in recent years at least, makes use of both sophisticated propagators and heuristic commitment techniques and is typically systematic and

complete, local search generally uses simpler heuristics, appears to scale better in some domains (e.g. hard random 3-SAT [51]), and is often easier to understand. An overarching issue is the investigation of hybrid techniques that can combine the strengths of each. For example, can we use propagators with local search techniques? (Some work has recently been done on this question in the context of vehicle routing [52]). Can we apply the heuristics that are more typical of local search to constructive search or *vice versa*?

2. Are there structural characteristics of a problem (and of a sub-problem) that indicate it is better to apply constructive rather than local search techniques or *vice versa*? Can we move back and forth between constructive and local search while solving a single problem? Can we identify search states where we should change our style from constructive to local or *vice versa*?
3. Given that the same problem may be solved using different commitments (e.g. job-shop scheduling can be solved by assigning start times or by sequencing activities), is there any information that can tell us which types of commitments should be used? Can finding the right commitment type make the search for a solution much easier?
4. Commitments can be of different granularities. 'Macro commitments' are large moves in the search space where, for example, all the activities on a unary resource are sequenced (or resequenced) [18, 53]. In contrast, 'micro commitments' make small steps such as assigning (or reassigning) the start time of a single activity [25]. An interesting question therefore is the distinction among commitments of different granularities. Are they all equivalent in some sense (i.e. is a macro commitment simply an agglomeration of micro commitments)? Are there structures in the constraint graph or conditions of the search (e.g. significant backtracking) that suggest that a macro commitment may be of more use? If macro commitments are based on the same information as a micro commitment are they more cost effective? Is there a trade-off? (For research that has examined some of these issue see References 3, 25 and 54.)

5. THE COMPONENTS OF ODO: THE SCHEDULING STRATEGY

In this section, we discuss the three main components of a scheduling strategy: heuristic commitment techniques, propagators, and retraction techniques.

To model a scheduling algorithm, instances of each component are specified. Since it is the case that some scheduling algorithms may not use some components of a strategy, say a propagator, while others may use multiple propagators, it is possible to specify none, one, or many instances of each component for a single scheduling strategy.

5.1. Heuristic commitment techniques

Traditionally in CSP techniques, a commitment is the assignment of a value to a variable. With this narrow definition, heuristics focus upon variable and value orderings: what is the next variable to assign and to what value will it be assigned. With our more general definition of a commitment, we view heuristics as procedures which, on the basis of measurements of the current search state, suggest new commitments. The traditional variable and value ordering heuristics, as well as heuristics that make different search steps (e.g. References 34, 49 and 50), are particular examples of such procedures.

We divide the heuristic commitment technique into two sub-components: measurement of textures and the identification of the heuristic commitment based on the information in the textures. As with the larger search strategy, it is possible to specify a null-component in a heuristic commitment technique. We do not require heuristic commitment techniques to be based on texture measurements, however as we argue below, there are few heuristics that do not use textures for search guidance.

5.1.1. Texture measurements

A *texture measurement* is a technique for distilling information embedded in the constraint graph into a form that heuristics can use. A texture measurement is not a heuristic itself. For example, a texture measurement may label some structures in the constraint graph (e.g. constraints, variables, sub-graphs) with information condensed from the surrounding graph. On the basis of this condensed information, heuristic commitments can be made. A relatively small number of texture measurements have been explicitly identified [25, 55], however, we take the broad view of a texture measurement as any analysis of the constraint graph producing information upon which heuristic commitments are based.

The concept of texture measurements is directly related to the representational and search intuitions. If we are to intelligently search for a solution to a problem and if the problem information is represented in a rich constraint representation, we need to look to the constraints for guidance. We need to distill information from the underlying constraint graph representation of a problem state and base our search commitments on this information.

It is not the case that any possible heuristic decision that can be made in constraint-directed search is necessarily based on texture measurements. However, any heuristic commitment technique that makes use of information in the constraint graph is, at least partially, texture-based. For example, we would not classify a 'heuristic' that randomly selects an activity and randomly assigns it a start time as texture-based as it is doing no measurement of the constraint graph.

In general, a texture measurement may be prohibitively expensive (e.g. NP-hard or worse) to compute. Making practical use of texture measurements, then, often requires a polynomial estimation algorithm. For example, the *value goodness* texture is defined to be the probability that a variable, V, will be assigned to a particular value, v_a, in a solution [55]. To exactly calculate the value goodness we need the ratio of the number of solutions to the problem where $V = v_a$ to the total number of complete valuations. This is clearly impractical. In practice, therefore we might estimate the goodness of v_a by examining the proportion of values of connected variables that are consistent with $V = v_a$. We may then base a heuristic commitment on the (estimated) value goodness by choosing to assign value with greatest (or least) goodness. What information a texture distills, how that information is practically estimated, and what commitment is made on the basis of the estimated information form the basic issues surrounding texture measurements.

Example. The *contention* texture [25] is the extent to which variables related via a disequality constraint compete to be assigned to the same value. To exactly calculate contention for disequality constraint, C, we require all the solutions to the problem with constraint C removed. Contention is the ratio of the number of these solutions that are inconsistent with C to the total number of solutions. As with value goodness, exact calculation is impractical and therefore an estimation is required. For a unary resource in scheduling there is the added complication of

the time dimension: contending activities require the same resource over overlapping time intervals.

The Operation Resource Reliance/Filtered Survivable Schedules (ORR/FSS) heuristic [25] estimates contention by aggregating a probabilistic estimate of each activity's individual demand for the resource over time. If an activity does not require a resource its individual demand is 0. Otherwise, a uniform probability distribution over the possible start times of the activity is assumed: each start time has a probability of $1/|\mathrm{STD}|$. (Recall that STD is the domain of the activity's start time variable. A uniform probability distribution is the 'low knowledge' default. It may be possible to use some local propagation in the constraint graph to find a better estimate of the individual demand [25, 56].) 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 $\mathrm{est}_A \leqslant t < \mathrm{lft}_A$:

$$\mathrm{ID}(A, R, t) = \frac{\min(t, \mathrm{lst}_A) - \max(t - \mathrm{dur}_A + 1, \mathrm{est}_A)}{|\mathrm{STD}|} \tag{1}$$

Using equation (1) an individual demand curve for all activities is constructed. Figure 8 displays an example of three activities, A_1, B_2, and C_3, competing for a single resource, R_1. The individual demand curves of each activity are shown in Figure 9.

Each individual curve is summed to form an aggregate resource demand curve which is used as an estimation of resource contention. Figure 9 also displays the aggregate curve resulting from the sum of the three individual demand curves.

5.1.2. Identifying the heuristic commitment

Once the texture information has been distilled, it is necessary to make a heuristic decision to identify the commitment that will be asserted. There are a large number of ways to identify commitments that are likely to contribute to the search toward a solution. We illustrate this process with a portion of the ORR/FSS heuristic in the context of job-shop scheduling.

Once contention has been estimated, the aggregate demand curves and a time interval equal to the average activity duration are used to identify the {resource, time interval} pair with the greatest contention. By definition, the unassigned activity, A, that contributes the most individual demand to the critical time interval is the most critical activity. Once A is identified, the value selection heuristic is used to rate each of its possible start times by again examining the contention information. 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. The highest rated start time is then assigned to A.

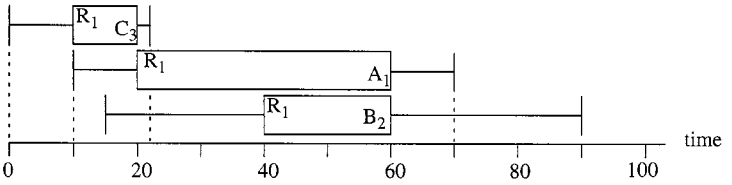


Figure 8. Activities A_1, B_2, and C_3

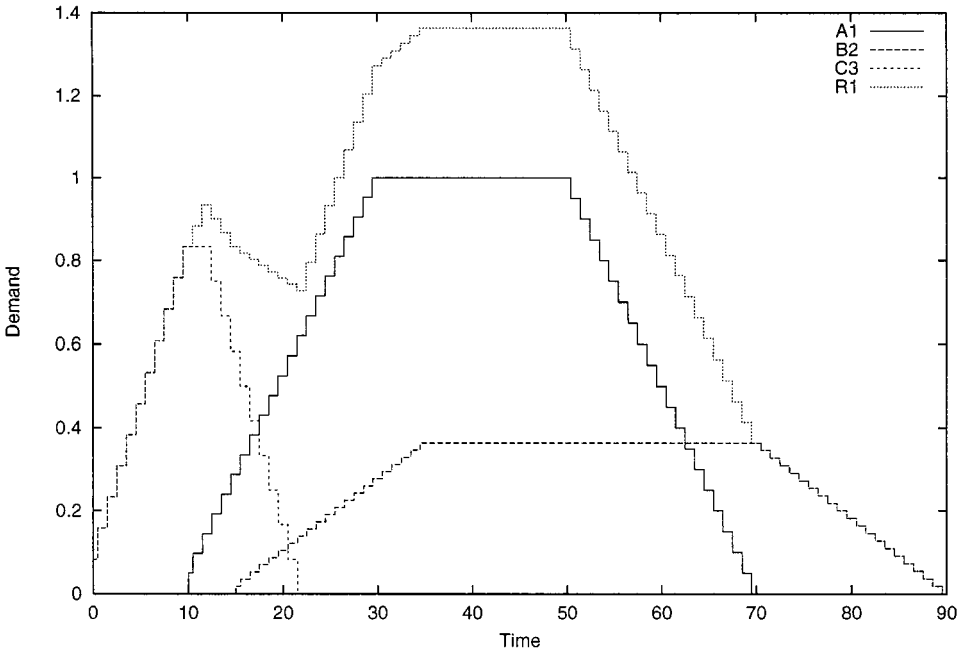


Figure 9. Individual demand curves (A_1, B_2, C_3) and their aggregate demand curve (R_1)

5.1.3. Research issues

The texture measurement concept produces a number of research issues.

1. What is the information that is to be distilled by a texture? We may not be able to precisely calculate this information in any practical algorithm, however a firm theoretical basis showing that, if we had this information, we could use it to find a solution allows us to then look to the practical aspects of forming an estimate of this information.
2. 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 trade-off in terms of impact on the overall scheduling algorithm? Can we characterize different estimation techniques on the basis of expected error from the true measurements?
3. 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 been estimated, it is necessary to use that 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, may have a significant impact on the search.
4. The *texture hypothesis* is one focus of our research in constraint-directed scheduling. It states that spending a significant, but polynomial, computational effort in measuring textures and

making commitments on the basis of the texture information pays off in the reduced need for backtracking and hence results in greater search efficiency. We believe this hypothesis to be true of scheduling though there appear to be CSP domains where it does not hold (e.g. hard random 3-SAT [51]).

5. Scheduling problems often require a variety of commitment types: the assignment of activities to resources, the instantiation of process plans, the addition of new resources, etc. Can textures provide a basis for integrating a variety of commitment types in a single search? Based on the constraint graph, can we (heuristically) decide that it is better to make a resource assignment at same point than to sequence activities?

5.2. Propagators

A propagator is a function that analyses 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. 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 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 often possible to further prune the search space.

Examples of propagators in CSP include the various consistency enforcement algorithms such as forward-checking, arc-consistency [11], and k-consistency [11, 57, 58]. 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 (i.e. a unary 'not equals' constraints).

5.2.1. Example: constraint-based analysis

Constraint-Based Analysis (CBA) [28, 29, 49, 50] enforces arc-B-consistency on unary resource constraints. Arc-B-consistency [59] (where 'B' stands for 'bounds') ensures that for the minimum and maximum values of any variable, v_1, there exists at least one consistent assignment for any other connected variable, v_2 (when considered independently of all other variables). 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 analyses the start and end times of all pairs of activities executing on the same unary capacity resource. Given activities, A_i and A_j, that compete for the same resource, CBA identifies the following cases:

1. If $\text{lft}_i - \text{est}_j < \text{dur}_i + \text{dur}_j \leqslant \text{lft}_j - \text{est}_i$ then A_i must be before A_j.
2. If $\text{dur}_i + \text{dur}_j > \text{lft}_j - \text{est}_i$ and $\text{dur}_i + \text{dur}_j > \text{lft}_i - \text{est}_j$ then the current state is a dead-end.
3. If $\text{dur}_i + \text{dur}_j \leqslant \text{lft}_j - \text{est}_i$ and $\text{dur}_i + \text{dur}_j \leqslant \text{lft}_i - \text{est}_j$ 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.

5.2.2. Example: *edge-finding*

Given, S, a non-empty sub-set of activities executing on the same resource, and activity $A \notin S$, on the same resource as the activities in S, edge-finding operationalizes implications (2) and (3).

$$\left[\begin{array}{l}(\mathrm{lft}(S) - \mathrm{est}(S) < \mathrm{dur}_A + \mathrm{dur}(S)) \\ \wedge (\mathrm{lft}(S) - \mathrm{est}_A < \mathrm{dur}_A + \mathrm{dur}(S))\end{array}\right] \rightarrow \mathrm{est}_A \geqslant \mathrm{est}(S) + \mathrm{dur}(S) \tag{2}$$

$$\left[\begin{array}{l}(\mathrm{lft}(S) - \mathrm{est}(S) < \mathrm{dur}_A + \mathrm{dur}(S)) \\ \wedge (\mathrm{lft}_A - \mathrm{est}(S) < \mathrm{dur}_A + \mathrm{dur}(S))\end{array}\right] \rightarrow \mathrm{lft}_A \leqslant \mathrm{lft}(S) - \mathrm{dur}(S) \tag{3}$$

Implication (2) 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 of A. Similarly, Implication (3) is used to find a new latest end time.

Complete examination of the 2^N subsets of activities on a resource is not practical. However, it is possible to deduce the same consequences by examining only N^2 subsets of activities on each resource [27, 35, 48]. Algorithms to do this with time-complexity of $O(N^2)$ [8] and $O(N \log N)$ [60] for each resource have been presented.

Many powerful propagation techniques have been developed for constraint directed scheduling in recent years [6, 8, 27, 48, 60, 61]. It has long been known that search can be drastically reduced by enforcing various degrees of consistency however the effort to achieve high degrees of consistency appears to be at least as expensive as search itself [58]. The goal for propagator research, then, is to find the trade-off between complexity of the algorithm and the resultant easing of the search effort.

5.3. Retraction techniques

Assume that a search moves through a sequence of states $S = (s_0, s_1, s_2, \ldots, s_k)$ via the assertion and retraction of commitments. Further assume that in state s_k it is determined that one or more of the currently asserted commitments must be retracted. Such a state arises in a constructive search because a mistake has been made: as a result of one or more of the asserted commitments, s_k is inconsistent with respect to the constraints in the problem. In a local search context, s_k is simply any state since, typically, all moves have some retraction component.

In such a state, the retraction component of the search strategy must then answer two questions:

1. Which commitments should be retracted?
2. In retracting a commitment that was made, say at state s_i, where $i < k$, what should be done with the intervening commitments, that is those made in all states s_j, where $i < j < k$?

Techniques for identifying the set of C of commitments to be retracted fall into two general categories:

1. *Provable*: Provably, 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 may also need to be retracted. After the proof, the subspace is not visited again.

2. *Heuristic*: Based on a heuristic evaluation, it is determined that C is likely to be the cause of the need for retraction.

There is a distinction between provability and completeness. It is possible to have a retraction technique (such as Limited Discrepancy Search (see below)) that is complete but not provable, that is, that retracts commitments even though there may be a solution in the sub-space. To maintain completeness, the sub-space is revisited later in the search if no solution has been found elsewhere.

There are three ways to treat the intervening commitments: retract them all, retract none, or retract some. Typically, provable retraction retracts all the intervening commitments though there are examples (e.g. dynamic backtracking [62]) where some intervening commitments are not retracted if it can be shown that they are not dependent upon the retracted commitment.

5.3.1. Example: chronological backtracking

At one end of the retraction spectrum is chronological backtracking, which consists of retracting the most recently made commitment. Clearly, since 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 may 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 sub-space below that wrong commitment before finding and retracting it. The question of intervening commitments is moot as there are none if the most recent commitment is retracted.

If chronological retraction is to be improved upon, while still maintaining provability, it is necessary to exploit the situation where chronological retraction does too much work: states where chronological retraction will exhaustively and fruitlessly search a sub-space. All provable retraction techniques rely on some mechanism to identify the most recent state at which it is possible to escape the dead-end (e.g. backjumping [63], conflict-directed backjumping [64], graph-based backjumping [65], dynamic backtracking [62]). Most of these techniques (with the exception of dynamic backtracking) retract all intervening commitments. While many of these techniques show good average time performance on constraint satisfaction problems, few have been applied to scheduling where the most common method of provable retraction is chronological retraction.

5.3.2. Example: limited discrepancy search

Limited Discrepancy Search (LDS) [60, 61] is based on the intuition that a good heuristic will only make a few mistakes in an unsuccessful search for a solution. Therefore, after failing to find a solution while following the heuristic, a good way to continue search is to examine all those paths in the search tree that differ from the heuristic path by at most one step, that is with a discrepancy level of one. If search still fails, then examine all those paths in the search tree with a discrepancy level of at most two and so on. LDS examines those nodes with a limited number of discrepancies from the heuristic path, increasing that limit as time allows and while no solution is found.

LDS is not a provable retraction technique as it retracts commitments without proving that no solution exist in the sub-tree below the commitment. However LDS is complete, as by revisiting

states, it will eventually (at higher cost than chronological retraction [68]) traverse the entire tree in a systematic fashion.

In terms of our retraction questions, LDS is limited by the discrepancy level and existing search tree in choosing the commitments it retracts. Once a commitment is chosen all intervening commitments are also retracted.

5.3.3. Example: retraction for local search

Retraction for local search is typically completely guided by the heuristics of the local search: whatever commitments that the heuristic identifies are retracted and intervening commitments are not. Completeness of search is abandoned for the advantages of free movement in the search space. This can be seen as the other extreme from chronological backtracking which maintains completeness at the cost of freedom to move. LDS and its variations (e.g. Improved Limited Discrepancy Search (ILDS) [68], Depth-bounded Discrepancy Search (DDS) [69] make compromises between the ability to heuristically move through the search space and the additional effort for the maintenance of completeness [70]. LDS is able to retract a commitment that, based on an *a priori* analysis [66], is believed to be more likely to be incorrect than the most recently made commitment. The trade-off is that to do this and maintain completeness, LDS must spend polynomial more effort than chronological retraction [68].

Other variations and combinations of retraction techniques have been proposed [8, 62, 71], however each variation depends on answering our two questions (identifying the to-be-retracted commitment(s) and dealing with the intervening commitments) in different ways. Interested readers are referred to Reference 66 for a recent examination of retraction techniques.

6. SCHEDULING ALGORITHMS AS INSTANCES OF THE FRAMEWORK

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

6.1. The ORR/FSS algorithm

One of the algorithms implemented in the MicroBOSS Scheduler is the Operation Resource Reliance/Filtered Survivable Schedules (ORR/FSS) algorithm [25, 72, 73]. It is a constructive algorithm that uses the ORR/FS heuristic. The texture measurements estimated are the reliance of each activity and the contention for each resource. The heuristic commitment is found by identifying the most critical activity and rating its start times in terms of survivability. Survivability is the likelihood that an assignment of the start time will 'survive' to participate in a full solution and is calculated based on the reliance and contention measures. The commitment made is to assign the most survivable start-time to the activity with the highest reliance on the resource and time interval with highest contention.

Two propagators are used: temporal propagation and 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 that: $\text{est}_j \geqslant \text{est}_i + \text{dur}_i$ and $\text{lft}_i \leqslant \text{lft}_j - \text{dur}_j$. Resource propagation (arc-consistency) plays a similar role for unary capacity resource constraints. For example, if A_i and B_j require the same unary resource and $\text{lst}_j < \text{eft}_j$, then for the time interval $[\text{lst}_j, \text{eft}_j)$, B_j must be the only activity using the

resource. Resource propagation will remove the values $[\text{lst}_j - \text{dur}_i + 1, \text{eft}_j - 1]$ from the possible start times of A_i. Both propagators remove values from the start-time domain of activities by asserting commitments composed of unary disequality constraints.

The retraction technique is chronological backtracking.

6.2. *The SOLVE algorithm*

The SOLVE algorithm [8, 74] 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 while the other components are relatively inexpensive. In SOLVE, the heuristic commitment technique is a simpler, less expensive, less powerful technique while the propagators are more expensive and more powerful.

The heuristic commitment technique is the Left-Justified Randomized heuristic. The set of activities that can execute before the minimum earliest finish time of all unscheduled activities is identified. One activity from this set is randomly selected and scheduled at its earliest start time. The heuristic commitment that is made is a unary equals constraint, assigning the start-time of the selected activity. The heuristic commitment technique randomly searches the space of 'left-justified' schedules which have been shown to form a dominance class in terms of makespan minimization [37]. While it is clearly a measurement of the constraint graph (based on earliest start times and earliest finish times of activities), it is unclear what, if any, underlying constraint graph properties are being estimated.

In addition to temporal propagation and resource propagation, SOLVE uses an extensive set of propagators including edge-finding. Edge-finding adds unary greater-than constraints on activity start-time variables and unary less-than constraints on the activity end-time variables.

The retraction technique is bounded chronological backtracking with restart: chronological backtracking is done until the user-specified bound on the number of backtracks is reached. At that point all commitments are retracted and the search starts over. Clearly, at the root node a different commitment than previously must be made to prevent cycling. With a randomized heuristic commitment technique the random nature of the heuristic performs this function.

6.3. *GERRY*

GERRY [19, 20] is a local search algorithm based on the MinConflicts heuristic [14]. Working from a total assignment of start times that fails to satisfy some set of constraints, GERRY will reschedule some activity in order 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 do this, GERRY reschedules one of the conflicting activities in a MinConflicts fashion, that is, by assigning it to a new start time that will minimize its conflicts with other activities. GERRY examines moving each conflicting activity to the previous and next time at which the resource is available. Each of these moves is evaluated by a linear combination 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.

Table II. Summary of the ODO policy model of three example scheduling algorithms

	Heuristic Commitment Technique			
Algorithm	Texture measurements	Commitment identification	Propagators	Retraction technique
ORR/FSS	Contention and reliance.	Assign the most survivable start-time to the activity with the highest reliance on the {resource, time interval} with highest contention.	Temporal and resource propagation.	Chronological backtracking.
SOLVE	Unknown.	Randomly pick an activity from the identified set and assign it to its earliest start-time.	Edge-finding, temporal and resource propagation.	Bounded chronological backtracking with restart.
GERRY	Contention and over-all schedule cost.	Assign activities with highest contention to a start time that minimizes contention.	Temporal propagation.	Retract the previous start-time assignment for the activity assigned in the heuristic technique.

In terms of textures, the activity rating procedure estimates contention by calculating the weighted sum of violations for each activity and then estimates the change in contention from moving activities to other start times. Every l commitments, a different texture, the overall cost of the schedule, is calculated. If the cost is less than the previous schedule (i.e. 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 at some probability based on a simulated annealing technique [75].

GERRY (and other local search techniques) can be modelled in our framework by encoding the local search moves as first asserting a new commitment (in this case, start-time assignment) and then retracting an existing commitment. Every l iterations the retraction is different because the whole schedule is evaluated. If the new schedule is accepted, 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.

6.4. Summary

Table II displays a summary of how each of the example scheduling algorithms can be modelled with an ODO policy.

7. INVESTIGATIONS IN THE ODO FRAMEWORK

Using the ODO framework both as a cognitive and implementational tool, we have addressed a number of research issues over the past seven years. In this section, we select four examples, as well as providing brief descriptions of our current and future work.

7.1. Combining constructive and local search

One of the consequences of our commitment mechanism is the ability to switch between different styles of search. Given the differing strengths of constructive and local search techniques, an interesting question is the utility of switching from one to the other within a single problem.

The MinConflicts hill-climbing technique [14] has been successfully applied to a variety of scheduling problems [20, 21, 76]. It has been observed that MinConflicts performs better when given a good initial state from which to start [14]. Given ODO's ability to switch between search styles, perhaps we could use a powerful constructive technique to find an initial state, switch to MinConflicts and quickly find a solution. The overall search (constructive plus MinConflicts) may find a solution more quickly than continuing with constructive search or than starting MinConflicts from a random initial state.

Using the ORR/FSS algorithm for the constructive portion of the search, our overall strategy was the following: run ORR/FSS until it either solves the problem or reaches a bound on the number of heuristic commitments. If the bound is reached, use the existing partial solution as an initial state for the MinConflicts repair policy. Experiments proved disappointing: we found solutions more often by simply continuing to use ORR/FSS alone than when we switched to MinConflicts.

Analysis revealed that similarities in texture information and commitment granularity between the two search styles that contributed to the negative result. ORR/FSS uses contention and reliance to identify the unassigned activities with the highest demand on a highly contended-for resource while MinConflicts uses the activities that had the most resource conflicts with other activities. These are two measures of the same underlying phenomenon: the competition among activities for a resource reservation. Given the similarity of this information, it is not surprising that an area of the search space where ORR/FSS is making many commitments and backtracks is also an area of the search space from where MinConflicts has difficulty escaping. Secondly, both ORR/FSS and MinConflicts make small granularity, 'micro' commitments: 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 can not quickly move to a solution. When reaching the commitment-bound, the constructive search is making and retracting micro commitments without much success. In changing to local search, we hoped that MinConflicts would be able to quickly escape the sub-space and proceed to a solution. However, 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 to make a micro commitment that immediately resulted in a lower cost state. What was necessary was a macro commitment (e.g. a set of activity reassignments) that changed a number of the start times in a single step. Such a commitment may be able to move to a lower cost state in a single step. At present it is an open question as to what information to gather to decide when to make macro commitments and also to decide which ones should be made. (Recent work in the integration of local search with linear programming techniques shows promise in 'large-scale, coherent moves' in the search space [54]).

This is the first work of which we are aware that attempts to combine constructive and local search in this way. While the results are negative, in that we were not able to improve upon scheduling performance by combining different techniques, the experiment revealed unexpected intricacies arising from similarities in the commitment types and the texture measurements. A more in-depth presentation of this work can be found in Reference 44.

7.2. Investigating the texture hypothesis

The texture hypothesis states that spending significant, but polynomial effort, at each search state, in the estimation of texture information and basing heuristic commitments on that information will pay-off in terms of higher quality decisions and greater search efficiency. To investigate this hypothesis and address criticisms of it [8, 49], we conducted an experiment to evaluate a variety of heuristic commitment techniques while holding the propagators and retraction techniques constant in each experimental condition.
The heuristic commitment techniques used in the experiments are as follows:

(a) SumHeight—a variation of the ORR/FSS heuristic. The key differences are an event-based representation of the texture measurement curves and the actual commitment that is made. Rather than assigning start times with FSS, SumHeight identifies the two activities with highest reliance on the most critical resource and event-point and heuristically posts a precedence constraint between them.
(b) CBASlack—the heuristic portion of the Precedence Constraint Posting algorithm [49, 50] where a pair of activities with the minimal (pairwise) slack are identified and sequenced to preserve the maximum amount of slack.
(c) FirstCommit—a heuristic that identifies a resource with minimal slack and then completely sequences that resource before moving to the resource with the next smallest slack [77]. The activity sequencing is done by repeatedly choosing an activity (based on heuristic criteria) and scheduling it before all the unscheduled activities on the resource.
(d) LJRand—the Left-Justified Randomized heuristic described earlier.

SumHeight and, to a lesser extent, CBASlack are representatives of the texture hypothesis as they spend more computational effort (as compared to FirstCommit and LJRand) in distilling information on which to base heuristic commitments.

Three consistency techniques (edge-finding, CBA, and temporal propagation) are used after each new heuristic commitment is made. Two backtracking techniques are used in two experimental conditions: chronological backtracking and Limited Discrepancy Search (LDS) [67].

Using a set of 21 job-shop scheduling problems from the Operations Research library of bench-mark problems [78], the algorithm (using each heuristic commitment technique) is run on a number of instances of each problem with a range of makespans. Specifically, for each problem, the optimal (or best-known upper bound) makespan is used initially. If a solution cannot be found within the time limit, we increase this makespan by 0·005 times the optimal makespan and re-run the algorithm. Lengthening of the makespan continues until a solution is found. Only the results from the final, successful search are used and therefore algorithms are compared on two criteria: mean relative error (how close to the optimal they were able to solve the problems) and number of commitments (the effort expanded in finding a solution).

Figures 10 and 11 plot mean relative error from the optimal makespan for each heuristic against the mean number of heuristic commitments made, for chronological backtracking and LDS, respectively. The graph layout means that the better algorithms will be closer to the lower left-hand corner. Note the differing scales of the two graphs.

With chronological backtracking, our experiments show that SumHeight finds significantly better schedules (i.e. lower mean relative error) than all the other heuristics. CBASlack finds better schedules than FirstCommit, which in turns finds better schedules than LJRand. The only

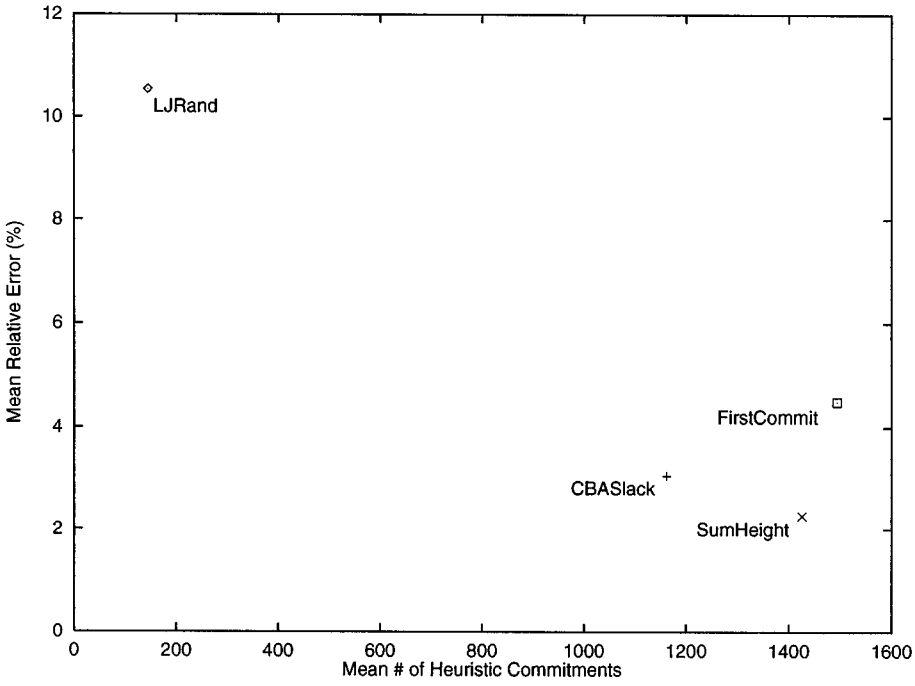


Figure 10. Mean relative error vs. mean # of heuristic commitments using chronological backtracking

significant difference in the number of heuristic commitments is that LJRand makes fewer than any of the other heuristics. The LDS experiments show no significant difference between SumHeight and CBASlack in terms of mean relative error though both are significantly better than FirstCommit which is itself significantly better than LJRand. In terms of the number of heuristic commitments the only significant result is that FirstCommit uses significantly fewer heuristic commitments than any of the other heuristics.

Overall, these results support the texture hypothesis and fail to corroborate the claims of the inferiority of sophisticated heuristics. A more detailed description of this work, with more experimentation, can be found in Reference 79.

7.3. Generalizing texture measurement estimation

Though texture measurements are a domain independent concept [55], they have only been applied, in scheduling, to the unary capacity resource constraints that exist in job-shop scheduling [25]. To apply textures to different constraints (e.g. inventory minimums, inventory maximums, multi-capacity resources) it may be necessary to revisit at least two of the basic texture measurements research questions: what is the information that is to be distilled from the graph and what is the algorithm to calculate or estimate this information.

In examining the work on the contention texture [25, 79], it is clear that contention is directly applicable to disequality constraints: where two or more variables are competing for the same

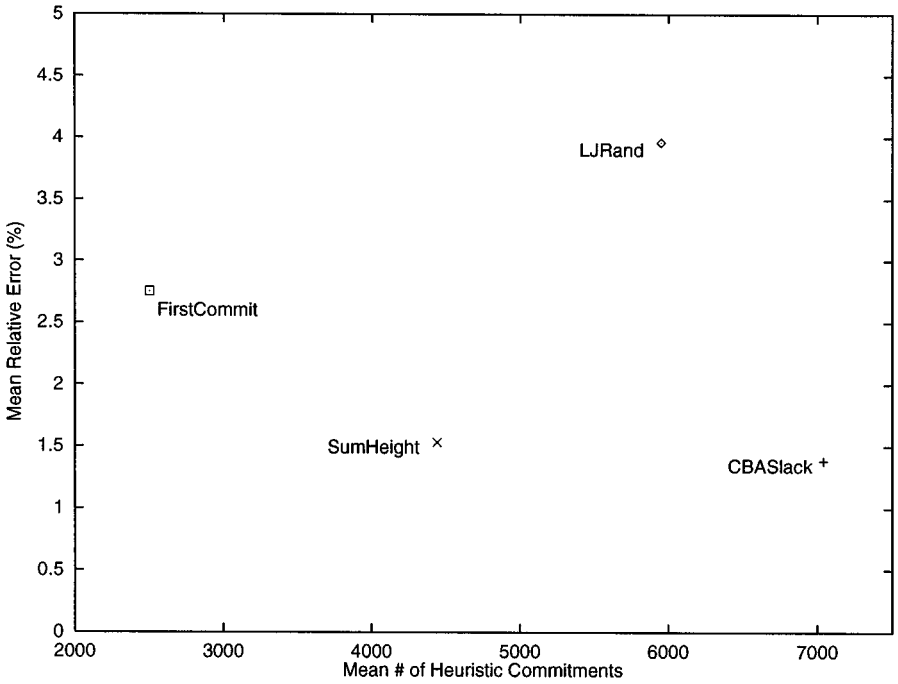


Figure 11. Mean relative error vs. mean # of heuristic commitments using LDS

value and are constrained not to have the same value. To extend contention to constraints generally, we introduce the concept of the *probability of breakage* of a constraint. For a disequality constraint, the probability of breakage is the extent to which its variables compete for the same value (i.e. contention) because a disequality constraint expresses that the variables cannot be assigned to the same values. For an equality constraint, for example, the relationship is reversed: the more the variables 'compete' for the same value the less likely the constraint is to be broken. The probability of breakage measures the extent to which variables compete for value tuples that the constraint defines as incompatible and so it is applicable to constraints in general, not just to disequality constraints. A critical advantage of the estimating probability of breakage texture is that we can compare that probability among different types of constraints (e.g. we can compare the criticality of a unary maximum resource capacity constraint with that of an minimum inventory constraint).

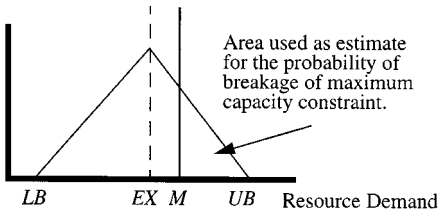
In order to test the usefulness of this generalized texture, we formulated three estimation algorithms for unary capacity resources. The goal of the experiment is to compare heuristics based on probability of breakage texture to existing heuristics. The three estimations of the probability of breakage of unary resource constraints developed are JointHeight, TriangleHeight, and VarHeight. All three are based, as is contention, on an activity's probabilistic individual demand for a resource over time. Each estimation uses the event-based individual activity demand formulation developed in Reference 79. The differences among the methods of estimation surround how the individual demand is aggregated to form an estimate of the probability of breakage of a constraint over time.

JointHeight is the most computationally expensive, and, we believe, most accurate estimation. It calculates (based on the individual demand) the joint probability of any two activities executing at each event-point. Examples of the TriangleHeight and VarHeight for a resource maximum constraint are shown in Figure 12. TriangleHeight estimates the probability of breakage based on the expected value and a distribution around the expected value. The simple sum of individual demands is the expected value at each event point. In addition, we can find the upper bound and lower bound on the demand. By assuming a triangular distribution around the expected value, the probability of breakage at event point t is estimated by a portion of the area under the curve. Given the maximum capacity constraint, M, the area under the curve to the right of the line representing M is interpreted as the probability that the maximum capacity constraint will be broken. For VarHeight, the same expected value is used, but the distribution is assumed to be a normal distribution formed based on the sum of the variances of the individual demands. Further discussion of these estimation techniques and their underlying assumptions is beyond the scope of this document. Interested readers are referred to Reference 80.

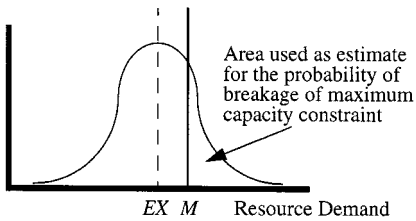
The same heuristic commitment is used for each texture estimation: the two most critical activities are identified based on the aggregate resource curve and they are sequenced based, again, on the aggregate resource curve.

Our experiments are based on the model described in the previous section. In particular, we use the same propagators, retraction techniques, and problems. In order to compare our results against existing techniques we used the two best heuristics from the previous experiment: SumHeight and CBASlack.

With chronological backtracking (Figure 13) our results indicate that, in terms of mean relative error, VarHeight outperforms TriangleHeight, CBASlack,and JointHeight while SumHeight



The TriangleHeight Estimation



The VarHeight Estimation

Figure 12. The TriangleHeight and VarHeight estimations of probability of breakage of a maximum resource constraint, M

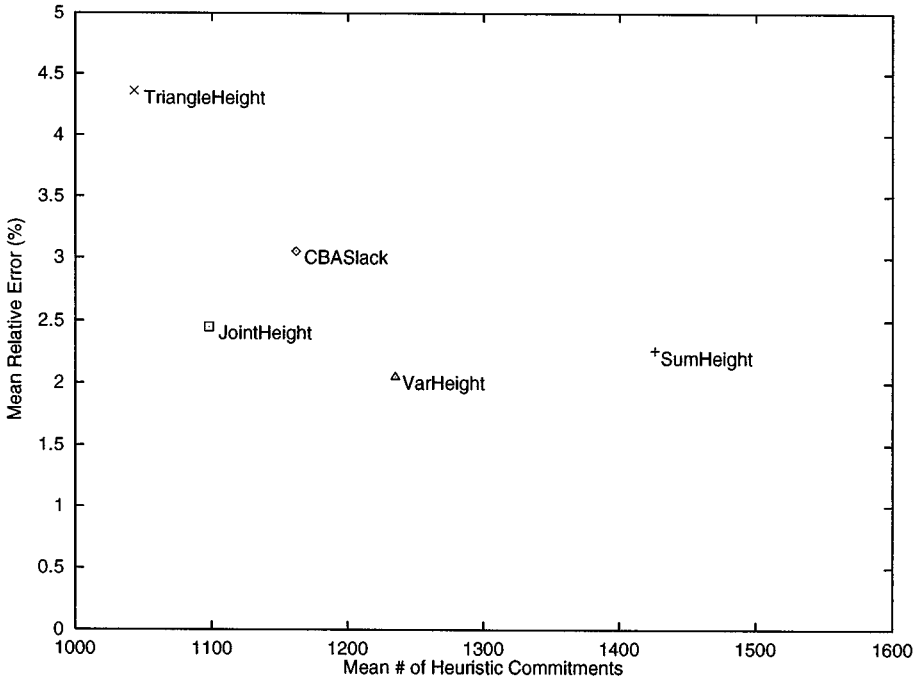


Figure 13. Mean relative error vs. mean number of heuristic commitments using chronological backtracking

outperforms CBASlack and TriangleHeight. There are no significant differences among VarHeight and SumHeight and between SumHeight and JointHeight. There are no significant differences in the number of heuristic commitments. In the LDS results (Figure 14) we find no significant differences in mean relative error among CBASlack, VarHeight, and SumHeight. JointHeight is significantly worse than each of these heuristics and TriangleHeight is significantly worse than all other heuristics. In terms of the number of heuristic commitments, VarHeight uses significantly fewer heuristic commitments than CBASlack. The only other significant differences are that JointHeight uses fewer heuristic commitments than SumHeight an CBASlack and that TriangleHeight uses fewer heuristic commitments than CBASlack.

The experiment indicates that we have been successful in formulating new texture measurement estimation techniques based on the probability of constraint breakage that can form the basis of the heuristics that are as good or better (or job-shop scheduling problems) than existing heuristics. In addition, the basis of VarHeight on the more general texture measurement of probability of breakage indicates that it may be applied to constraints other that disequality constraints. The application remains to be explored. The lack of performance of JointHeight is troubling (and interesting) given our expectation that VarHeight and TriangleHeight are poorer estimates of the probability of breakage than JointHeight. We are unable to explain this anomaly and plan to further investigate it in future work.

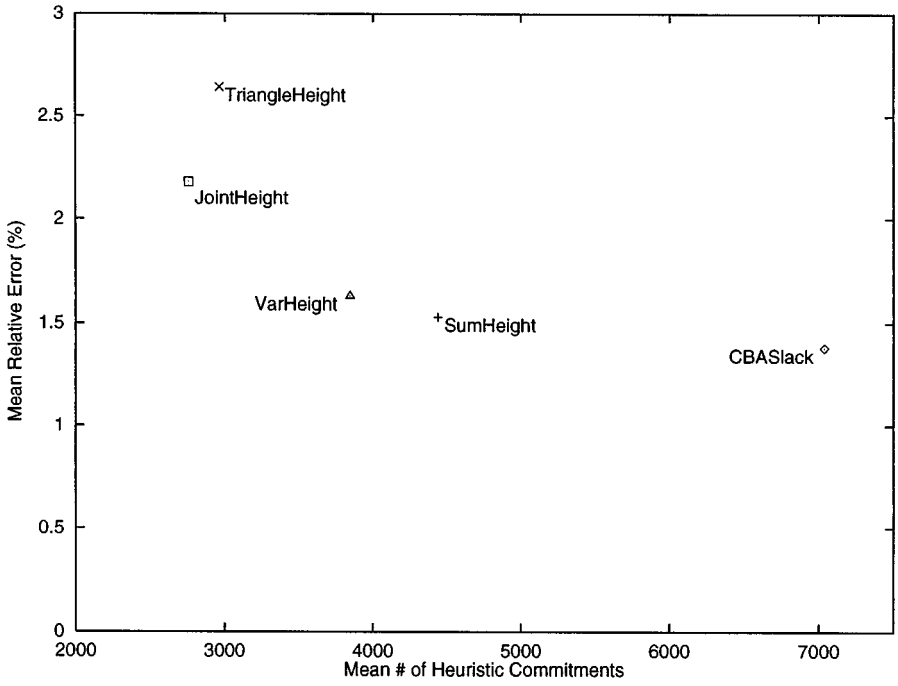


Figure 14. Mean relative error vs. mean number of heuristic commitments using LDS

7.4. Scheduling with uncertainty

In a real-world environment the probability of a precomputed schedule being executed exactly as planned is low: machines malfunction, raw material deliveries are delayed, resources are not available when required. A disrupted schedule incurs higher costs due to missed customer delivery dates, higher work-in-process inventory, and idling of people and/or machines [81].

Unknown uncertainties may be best addressed a purely reactive scheduling system. Alternatively, if the uncertainty in the scheduling environment can be mathematically modelled, then one can investigate building predictive, robust schedules. A robust schedule is one that is likely to remain valid under a wide variety of disturbances [82]. Attempts to build robustness into a schedule try to protect the schedule from being interrupted by stochastic events and therefore to reduce the implementation cost in the face of uncertainty.

One way of achieving robustness is through the use of temporal protection, first proposed by Chiang and Fox [81]. Temporal protection adds slack time to an activity's duration. This slack time will be used in the event of schedule disruption.

In our extension of [81] a temporally protected activity is composed of an inner interval P_{inner} and an outer interval P_{outer} with *lower-slack* being the difference and coming at the beginning of P_{outer} as shown in Figure 15. P_{inner} defines the start time and estimated duration of the activity (i.e. the time during which we expect the activity to be performed). Critical resources

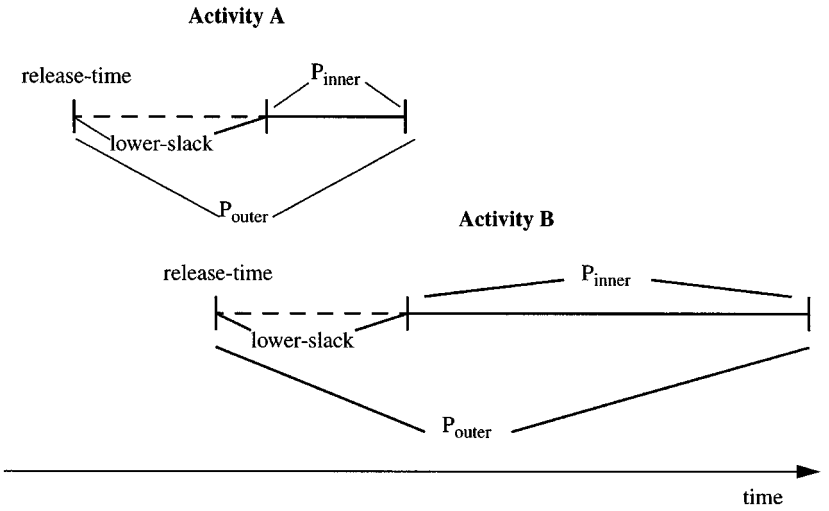


Figure 15. Temporal protection, illustrating overlapping activities A and B, where A must execute before B

are allocated for the period of P_{inner}. P_{outer} defines the earliest time we would expect the activity to start. Non-critical resources for the same activity are allocated from the beginning of P_{outer}. Both P_{inner} and P_{outer} are functions of the original activity duration and knowledge about resource failures. The overlapping slack intervals between activities provide release flexibility during schedule execution. When executing a schedule, if the previous activity takes less time than the protected duration, the critical resource is released earlier than specified and the activity can start as soon as the critical resource is available. Chiang and Fox [81] also defined an *upper-slack* parameter between the end of P_{inner} and the end of P_{outer}. We use an *upper-slack* of 0 due to the expectation that it will cause increased tardiness without providing advantages for dealing with uncertainty.

To determine how much temporal slack to add to an activity, we make use of the following parameters: P, the original processing time of an activity, F, a random variable representing time between machine failure, and D, a random variable representing the duration of a failure.

P/F gives the expected number of breakdowns during the processing of an activity. The extended duration P_{ext} with machine breakdown is

$$P_{\text{ext}} = P + \frac{P}{F} \times D \tag{4}$$

If the mean of D and F are known, as $\bar{D}$ and $\bar{F}$, then we can calculate the mean of P_{ext} as follows:

$$P_{\text{mean}} = P + \frac{P}{\bar{F}} \times \bar{D} \tag{5}$$

Instead of being random variables of known distribution, the failure duration and time between failures may only be known to be bounded approximately. Let the bounds be $(D_{\text{lb}}, D_{\text{ub}})$ for D and

(F_{lb}, F_{ub}) for F. We can then calculate the lower and upper bounds on P_{ext} as follows:

$$P_{extlb} = P + \frac{P}{F_{ub}} \times D_{lb}$$
$$P_{extub} = P + \frac{P}{F_{lb}} \times D_{ub} \tag{6}$$

Equations (4)–(6) are all based on the interpretation of $\bar{F}$ as the mean number of failures per unit of processing time. If $\bar{F}$ is actually the mean number of failures per unit of time (be it process-time or down-time) a slightly different calculation is used [83].

Experiments were run with ODO on Sadeh's job-shop scheduling problem set [25], using four methods for the calculation of temporal protection for an activity. Schedule execution was then simulated, with machine breakdowns generated randomly according to a variety of uncertainty scenarios. Schedule robustness was evaluated according to two criteria: the sum of the tardiness cost and the work-in-process cost and the deviation between the makespan of the predictive schedule and the executed schedule.

The most successful temporal protection parameters were as follows:

$$P_{inner} = P_{mean}$$
$$P_{outer} = P_{extub} \tag{7}$$

In addition, the following observations were made:

1. For highly temporally constrained problems, temporal protection reduces work-in-process cost, but increases the tardiness cost of executing a schedule.
2. For protected schedules under our uncertainty models, the makespan deviation is very small. For unprotected schedules, the execution makespan is much longer than the predicted schedule makespan, since machine failures are not taken into account.
3. When tardiness and work-in-process costs are equally weighted, the work-in-process cost is the majority of the total cost (80 per cent +). Temporal protection reduces work-in-process costs by releasing the activity around the time it can be worked on. Therefore, although tardiness cost grows when the schedule is protected, this is more than compensated for by savings in work-in-process costs.
4. Costs grow when machine failure uncertainty grows. Longer and more frequent failures increase both work-in-process costs and tardiness.
5. Cost savings by temporal protection are more significant when the bottleneck machine is subject to failure.

Overall the use of temporal protection of activities shows promise. We plan to extend this work as part of a more general investigation of predictive and reactive techniques of dealing with uncertainty in schedule execution. This work is presented fully in Reference 83.

7.5. Current and future work

Research in the ODO project is ongoing with the main focus being the exploration of incorporation of real-world constraints into the constraint-directed scheduling paradigm. The exploration includes both inquiries into methods of modelling these constraints as well as

investigation of propagators and texture measurements to enable efficient reasoning and high-quality scheduling results. Here we briefly note a number of our current and planned research activities. This is a far from complete discussion due to the on-going nature of the work.

7.5.1. Transportation scheduling

Many systems exist which can address certain subsets of the transportation problem (e.g. vehicle routing and vehicle routing with time windows) often using local search techniques [84]. Such systems, however, often cannot deal with the side constraints and optimization issues important in the real world such as the minimization of inventory holding costs at each retailer. Our approach is to model transportation problems as scheduling problems, with activities and constraints between these activities. By using a constraint-directed approach, we aim to be able to model all the side constraints found in real-world problems. This approach is also being investigated by a number of other researchers [52, 85].

The constraint representation we are using models vehicles as resources and trips as activities. Each vehicle is represented by two resources: a unary resource and a multi-capacity resource. The unary resource deals with constraints on the possible journeys the vehicle can make. For instance, time window constraints on the vehicle may preclude it from making certain deliveries. The multi-capacity resource deals with the vehicle capacity constraints, for instance, limitations on how much the vehicle can carry, and constraints on what type of goods can be mixed in a single load.

7.5.2. Scheduling with inventory

In the manufacturing context, raw material inventory is transformed by a series of activities to work-in-process inventory and eventually finished-goods inventory. Each activity may produce and/or consume quantities of inventory (at varying rates) that may need to be stored before and after the activity executes. There can be many constraints on the inventory including multiple storage resources, each with independent minimum and maximum storage levels, timing constraints concerning how long an inventory may (or must) remain in some state (e.g. cooling or curing time, spoilage time), and resource dependencies (e.g. if one activity is performed on a particular machine then only a subset of the downstream machines can be used for subsequent activities).

Our initial work addresses a simplified problem with batch-activities which consume inventory instantaneously at their start times and produce inventory instantaneously at their end times. We also limit ourselves to a single storage resource for each type of inventory. To solve this problem we need to sequence the activities on resources such that the resource capacity and the inventory storage constraints are respected. Our approach has lead to the application of the TriangleHeight and VarHeight texture estimators to inventory constraints as well as the creation of a new propagator based on the inventory constraints and the calculation of upper and lower bounds on the inventory levels. This work will be reported in full in Reference 86.

7.5.3. Alternate resources and alternate process plans

The importance of addressing alternate resources in constraint-directed scheduling cannot be over-stated. Nearly all real-world production scheduling problems have a choice of resources for

at least some of their activities. Alternate resources are found in transportation scheduling, resource allocation, and mixed production planning and scheduling problems. While early constraint-directed scheduling systems (e.g. ISIS [10] and OPIS [18]) represented and reasoned about alternative resources and process plans, there has been surprisingly little recent work in this area (though there are some exceptions, notably [9, 8]).

An approach we are studying is the notion of *probability of existence* (PEX) which first appeared in the context of the KBLPS system [9]. An activity that exists in some stage of the search may not exist in the final solution, even though no backtracking has occurred. For example, if we have the goal to create n units of widget x, we may do this by either taking the widgets from inventory, buying them from a third party supplier, or by manufacturing them ourselves. We want to be able to reason about all of these alternatives within the context of the other constraints in the problem because the dynamic state of the factory (e.g. the utilization levels of resources) has an impact on the best alternative. If machines for manufacturing widget x are already highly constrained, there may be more utility in buying the widgets from an outside supplier rather than further increasing machine utilization. The PEX approach instantiates an activity for each alternative and, in this case, links them by a PEX constraint specifying that the three PEX values must sum to 1. The PEX values can be taken into account by texture measurements and propagation techniques as well as being modified by commitments during search.

7.5.4. Sequence-dependent changeovers

Given two activities A and B, a changeover occurs when there is a need to execute an activity between the end of A and the start of B, when A immediately precedes B. Changeovers typically occur in factories where tools must be configured or resources must be cleaned. Characteristics of changeover activities (i.e. duration, resource requirements) are often asymmetric: the cost for $A \rightarrow B$ is different to that for $B \rightarrow A$. In addition, a changeover for $A \rightarrow B$ is only executed if B comes *directly* after A. If there is another activity C between A and B, then the changeover costs we have to deal with are the ones between A and C and between C and B.

We deal with changeovers in ODO by explicitly introducing changeover activities that, depending on scheduling decisions, may or may not exist in the eventual solution. This representation allows us to directly reason about the changeover activities including the computation of texture measurements. We are currently investigating heuristic commitment techniques that will make decisions based on local texture information that will tend to balance the various conflicting objectives.

7.5.5. Optimization and cost-based scheduling

There has been little basic research on the representation and optimization of the cost of a schedule in the constraint-directed scheduling literature though single performance measures (e.g. mean flow time, mean lateness, percentage of jobs tardy, and mean tardiness) and some combinations have been addressed [10, 25, 87–90]. For example, ISIS [10] included utility values as a surrogate for costs while MicroBOSS [25] explicitly represented costs in its constraint optimization algorithms. The actual cost of a schedule, however, is a much richer concept including not only performance measures but the cost of inventory storage, machine usage, machine maintenance, and product-quality trade-offs. There exists no general framework that can represent the various cost components and integrate them with constraint-directed scheduling technology.

This project will create a realistic cost model of the schedule in a multi-stage scheduling environment to explicitly deal with various cost components of real world scheduling. The key components of cost-based scheduling will be the creation of techniques to efficiently propagate cost information through the constraint graph and the formulation of new texture measurements that explicitly take cost information into account. We expect an interesting trade-off to arise from potential conflicts between feasibility and cost minimization: in some cases the commitment to be made to increase the probability of finding a solution may also be the commitment that will increase the expected cost the most. Understanding these trade-offs and the subsequent creation of commitment techniques that are able to deal with them are central aspects of this research.

8. CONCLUSIONS

In this paper we have provided an overview of the ODO project. The original vision of the project consisted of constructing a unified basis for constraint-directed scheduling encompassing both satisfaction of problem constraints and optimization of problem objectives. This vision has been embodied in the ODO framework, based on:

1. The use of the constraint graph as the primary knowledge representation tool.
2. The use of commitment assertion and retraction search operators.
3. A generic model of scheduling strategies.
4. The use of texture measurements as a basis for heuristic decision making.

We have presented a selection of the research we have performed with ODO, but as with any research project, there are perhaps more questions than answers. In particular, much of the promise of the use of commitments remains to be explored such as investigations of search states more amenable to certain styles of search and of the possibility of dynamic transition from one style to another as necessary during search. The study of texture measurements is also in its early stages as much needs to be done on the formalization of the underlying measures and estimations and on the understanding of heuristics based on such estimations.

The final area of work that we note here concerns the field of constraint-directed scheduling as a whole. It is only recently that researchers have turned to rigorous empirical comparisons of scheduling techniques rather than algorithmic 'track meets' [91]. At the same time, constraint-directed scheduling techniques are being applied in a variety of real-world contexts where there is a tendency to settle for a technique that is 'good enough' rather than understand the problem and the algorithms' behaviors. The ODO project is based on an attempt to balance the sometimes conflicting pressures of understanding algorithm behaviour versus (short-term) problem solving and industrial relevance. We are of the strong opinion that the field will not progress without both the rigorous scientific underpinning and the continual extension to the plethora of real-world constraints that were the original motivation for applying constraint technology to scheduling.

ACKNOWLEDGEMENTS

This research was funded in part by the Natural Science and Engineering Research Council of Canada, Numetrix Limited, the IRIS Research Network, the Materials and Manufacturing Ontario, and Digital Equipment of Canada.

With a project of this scope there are many people to be acknowledged. The authors would like to especially note discussions of the investigations of the ODO project with Norman Sadeh, Nicola Muscettola, Victor Saks, Edward Sitarski, Ioan Popescu, and Scott Hadley. Thanks also to the anonymous reviewers for detailed and cogent comments on earlier drafts of this paper.

REFERENCES

1. M. Fox, 'Observations on the role of constraints in problem solving', in *Proc. of the 6th Canadian Conf. on Artificial Intelligence*, 1986.
2. C. Le Pape and P. Baptiste, 'An experimental comparison of constraint-based algorithms for the preemptive job shop scheduling problem', in *CP97 Workshop on Industrial Constraint-Directed Scheduling*, 1997.
3. N. Muscettola, 'On the utility of bottleneck reasoning for scheduling', in *Proc. AAAI-94*, 1994, pp. 1105–1110.
4. J. C. Beck, A. J. Davenport and M. S. Fox, 'Five pitfalls of empirical scheduling research', in G. Smolka, (ed.), *Proc. 3rd International Conf. on Principles and Practice of Constraint Programming* (*CP97*), Springer, Berlin, 1997, pp. 390–404.
5. Y. Caseau and F. Laburthe, 'Cumulative scheduling with task intervals', in *Proc. Joint Int. Conf. and Symp on Logic Programming*, MIT Press, Cambridge, MA, 1996.
6. C. Le Pape and P. Baptiste, 'Constraint propagation techniques for disjunctive scheduling: the preemptive case', in *Proc. ECAI-96*, 1996.
7. D. Neiman, D. Hildum, V. Lesser and T. Sandholm 'Exploiting meta-level information in a distributed scheduling system', in *Proc. 12th National Conf. on Artificial Intelligence*, Seattle, WA, 1994, pp. 394–400.
8. W. P. M. Nuijten, 'Time and resource constrained scheduling: a constraint satisfaction approach', *Ph.D. Thesis*, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1994.
9. V. Saks, 'Distribution planner overview', *Technical Report*, Carnegie Group, Pittsburgh, PA, 15222, 1992.
10. M. S. Fox, 'Constraint-directed search: a case study of job-shop scheduling', *Ph.D. Thesis*, Carnegie Mellon University, Intelligent Systems Laboratory, The Robotics Institute, Pittsburgh, PA, 1983. CMU-RI-TR-85-7.
11. A. K. Mackworth, 'Consistency in networks of relations', *Artificial Intelligence*, **8,** 99–118 (1977).
12. E. P. K. Tsang, *Foundations of Constraint Satisfawction*, Academic Press, New York, 1993.
13. R. Dechter, A. Dechter and J. Pearl, 'Optimization in constraint networks', in R. Oliver and J. Smith (eds.), *Influence Diagrams, Belief Nets, and Decision Analysis*, Wiley, Chicester, England, 1990.
14. S. Minton, M. Johnston, A. Philips and P. Laird, 'Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems', *Artificial Intelligence*, **58,** 161–205 (1992).
15. D. Navinchandra and D. H. Marks, 'Design exploration through constraint relaxation', in *Expert Systems in Computer-Aided Design*, Elsevier, Amsterdam, 1987.
16. D. Navinchandra, *Exploration and Innovation in Design*, Springer, New York, 1991.
17. V. Kumar, 'Algorithms for constraint satisfaction problems: A survey', *AI Mag.*, 1992, pp. 32–44.
18. S. F. Smith, P. Ow, D. Matthys and J. Potvin, 'OPIS: An opportunistic factory scheduling system', in *Proc. Int. Symp. for Computer Scientists*, 1989.
19. M. Zweben, E. Davis, B. Daun and M. Deale, 'Informedness vs. computational cost of heuristics in iterative repair scheduling', in *Proc. IJCAI-93*, 1993, pp. 1416–1422.
20. M. Zweben, B. Daun, E. Davis and M. Deale, 'Scheduling and rescheduling with iterative repair', in M. Zweben and M. Fox (eds.), *Intelligent Scheduling*, Chap. 8, Morgan Kaufmann Publishers, San Francisco, 1994, pp. 241–256.
21. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guid to the Theory of NP-Completeness*, W. H. Freeman and Company, New York, 1979.
22. M. S. Fox, 'Constraint-guided scheduling—a short history of research at CMU', *Comput. Ind.*, **14,** 79–88 (1990).
23. J. Blazewicz, W. Domschke and E. Pesch, 'The job shop scheduling problem: Conventional and new solution techniques', *Eur. J. Oper. Res.*, **93**(1), 1–33 (1996).
24. D. Applegate and W. Cook, 'A computational study of the job-shop scheduling problem', *ORSA J. Comput.*, **3,** 149–156 (1990).
25. N. Sadeh, 'Lookahead techniques for micro-opportunistic job-shop scheduling', *Ph.D. Thesis*, Carnegie-Mellon Univeristy, 1991, CMU-CS-91-102.
26. C. Le Pape, 'Constraint-based programming for scheduling: An historical perspective', in *Working Papers of the Operations Research Seminar on Constraint Handling Techniques*, 1994.
27. J. Carlier and E. Pinson, 'An algorithm for solving the job-shop problem', *Management Sci.*, **35**(2), 164–176 (1989).
28. J. Erschler, F. Roubellat and J. P. Vernhes, 'Finding some essential characteristics of the feasible solutions for a scheduling problem', *Oper. Res.*, **24,** 772–782 (1976).
29. J. Erschler, F. Roubellat and J. P. Vernhes, 'Characteristing the set of feasible sequences for n jobs to be carried out on a single machine', *Euro. J Oper. Res.*, **4,** 189–194 (1980).

30. N. Sadeh and M. Fox, 'Focus of attention in an activity-based schedule', in *Proc NASA Conf. on Space Telerobotics*, 1989.
31. K. Sycara, S. Roth, N. Sadeh and M. Fox, 'Distributed constrained heuristic search', *IEEE Trans. Systems, Man, Cybernet*., **SMC-21**(6), 1446–1461 (1991).
32. A. Collinot and C. Le Pape, 'Controlling constraint propagation', in *Proc. 10th Int. Joint Conf. on Artificial Intelligence*, 1987.
33. P. Burke and P. Prosser, 'The distributed asynchronous scheduler', in M. Zweben and M. Fox (eds.), *Intelligent Scheduling*, Chap. 11, Morgan Kaufmann Publishers, San Francisco, 1994, pp. 309–339.
34. P. Van Hentenryck, *Constraint Satisfaction in Logic Programming*, MIT Press, Cambridge, MA, 1989.
35. Y. Caseau and F. Laburthe, 'Improving branch and bound for jobshop scheduling with constraint propagation', in *Proc. 8th Franco-Japanese Conference CCS'95*, 1995.
36. C. Le Pape, 'Using a constraint-based scheduling library to solve a specific scheduling problem', in *Proc. AAAI-SIGMAN Workshop on Artificial Intelligence Approaches to Modelling and Scheduling Manufacturing Processes*, 1994.
37. K. Baker, *Introduction to Sequencing and Scheduling*, Wiley, New York, 1974.
38. J. Little, K. Murty, D. Sweeney and C. Karel, 'An algorithm for the traveling saleman problem', *Oper. Res*., **11**(6), 972–989 (1963).
39. F. Glover, 'Tabu search part I', *Oper. Res. Soc. Am.* (*ORSA*) *J. Comput*., **1**(3), 109–206 (1989).
40. F. Glover, 'Tabu search part II', *Oper. Res. Sco. Am.* (*ORSA*) *J. Comput*., **2**(1), 4–32 (1990).
41. E. Nowicki and C. Smutnicki, 'A fast taboo search algorithm for the job shop problem', *Management Sci*., **42**(6), 797–813 (1996).
42. U. Dorndorf and E. Pesche, 'Evolution based learning in a job shop scheduling environment', *Comput. Oper. Res*., **22**(1), 25–40 (1995).
43. P. Laarhoven, E. Aarts and J. Lenstra, 'Job shop scheduling by simulated annealing', *Oper. Res*., **40**(1), 113–125 (1992).
44. E. D. Davis, '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, 4 Taddle Creek Road, Toronto, Ontario M5S 3G9, Canada, 1994.
45. E. D. Davis and M. S. Fox, 'ODO: a constraint-base scheduling shell', in *Proc. Workshop on Production Planning, Scheduling and Control*, *IJCAI-93*, 1993.
46. J. F. Allen, 'Maintaining knowledge about temporal intervals', *Commun. ACM*, **26**(11), 832–843 (1983).
47. C. Le Pape, 'Implementation of resource constraints in ILOG Schedule: A library for the development of constraint-based scheduling systems', *Intelligent Systems Engng*., **3**(2), 55–66 (1994).
48. Y. Caseau and F. Laburthe, 'Improved CLP scheduling with task intervals', in *Proc. 11th Int. Conf. on Logic Programming*, MIT Press, Cambridge, MA, 1994.
49. S. F. Smith and C. C. Cheng, 'Slack-based heuristics for constraint satisfaction scheduling', in *Proc. AAAI-93*, 1993, pp. 139–144.
50. C. C. Cheng and S. F. Smith, 'Applying constraint satisfaction techniques to job shop scheduling', *Ann. Oper. Res*., *Special Volume on Scheduling*: *Theory and Practice*, Vol. 1, 1996, forthcoming.
51. B. Selman, H. Levesque and D. G. Mitchell, 'A new method for solving hard satisfiability problems', in *Proc. AAAI-92*, AAAI Press/MIT Press, Cambridge, MA, 1992, pp. 440–446.
52. B. DeBacker, V. Furnon, P. Kilby, P. Prosser and P. Shaw, 'Local search in constraint programming: application to the vehicle routing problem', in *CP97 Workshop on Industrial Constraint-Directed Scheduling*, 1997.
53. R. J. M. Vaessens, E. H. L. Aarts and J. K. Lenstra, 'Job shop scheduling by local search', Technical Report COSOR Memorandum 94-05, Eindhoven University of Technology, 1994. *Inform. J. Comput*., submitted for publications.
54. D. Clements, J. Crawford, D. Joslin, G. Nemhauser, M. Puttlitz and M. Savelsbergh, 'Heuristic optimization: A hybrid AI/OR approach', in *Proc. CP97 Workshop on Constraint-Directed Scheduling,* November 1997.
55. M. S. Fox, N. Sadeh and C. Baykan, 'Constrained heuristic search', in *Proc. IJCAI-89*, 1989, pp. 309–316.
56. N. Muscettola, ' Scheduling by iterative partition of bottleneck conflicts', *Technical Report CMU-RI-TR-92-05*, The Robotics Institute, Carnegie Mellon University, 1992.
57. E. C. Freuder, 'Synthesizing constraining expressions', *Commun. Assoc. Comput. Mach*., **21**(11), 958–966 (1978).
58. E. C. Freuder, 'A sufficient condition for backtrack-free search', *J. Assoc. Comput. Mach*., **21**(1), 24–32 (1982).
59. O. Lhomme, 'Consistency techniques for numeric CSPs', in *Proc. IJCAI-93*, Vol. 1, 1993, 232–238.
60. J. Carlier and E. Pinson, 'Adjustment of heads and tails for the job-shop problem', *Eur. J Oper. Res*., **78,** 146–161 (1994).
61. P. Martin and D. Shmoys, 'A new approach to computing optimal schedules for the job shop scheduling problem', in *Proc. 5th Conf. on Integer Programming and Combinatorial Optimization*, 1996.
62. M. Ginsberg, 'Dynamic backtracking', *J. Artificial Intelligence Res*., **1,** 25–46 (1993).
63. J. Gaschnig, 'Experimental case studies of backtrack vs. waltz-type vs. new algorithms for satisficing assignment problems', in *Proc. 2nd Nat. Conf. of the Canadian Society for Computational Studies of Intelligence*, 1978.
64. P. Prosser, 'Hybrid algorithms for the constraint satisfaction problem', *Computational Intelligence*, **9**(3), 268–299 (1993).

65. R. Dechter, 'Enhancement schemes for constraint processing: backjumping, learning and cutset decomposition', *Artificial Intelligence*, **41,** 273–312 (1990).
66. W. D. Harvey, 'Nonsystematic backtracking search', *Ph.D. Thesis*, Department of Computer Science, Stanford University, 1995.
67. W. D. Harvey and M. L. Ginsberg, 'Limited discrepancy search', in *Proc. IJACI-95*, 1995, pp. 607–613.
68. R. Korf, 'Improved limited discrepancy search', in *Proc. AAAI-96*, 1996.
69. T. Walsh, 'Depth-bounded discrepancy search', in *Proc. IJCAI-97*, 1997.
70. W. Havens, 'Nogood caching for multiagent backtrack search', in *Proc. AAAI-97 Constraints and Agents Workshop*, Providence, RI, 1997.
71. N. Sadeh, K. Sycara and Y. Xiong, 'Backtracking techniques for the job shop scheduling constraint satisfaction', *Artificial Intelligence*, **76,** 455–480 (1995).
72. N. Sadeh, 'Micro-opportunistic scheduling', in M. Zweben and M. Fox (eds.), *Intelligent Scheduling*, Chap. 4, Morgan Kaufmann Publishers, San Francisco, 1994, pp. 99–138.
73. N. Sadeh and M. S. Fox, 'Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem', *Artificial Intelligence J*., **86**(1), (1996).
74. W. P. M. Nuijten, E. H. L. Aarts, D. A. A. van Arp Taalman Kip and K. M. van Hee, 'Randomized constraint satisfaction for job shop scheduling', in *Proc. IJCAI'93 Workshop on Knowledge-Based Production, Scheduling and Control*, 1993, pp. 251–262.
75. S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, 'Optimization by simulated annealing', *Science*, **220,** 671–680 (1983).
76. M. D. Johnston and G. Miller, 'SPIKE: Intelligent scheduling of Hubble space telescope observations', in M. Zweben and M. Fox (eds.), *Intelligent Scheduling*, Chap. 14, Morgan Kaufmann Publishers, San Francisco, 1994, pp. 391–422.
77. P. Baptiste, C. Le Pape and W. Nuijten, 'Constraint-based optimization and approximation for job-shop scheduling', in *Proc. AAAI-SIGMAN Workshop on Intelligent Manufacturing Systems, IJCAI-95*, 1995.
78. J. E. Beasley, 'OR-library: distributing test problems by electronic mail', *Journal of the Operational Research Society*, **41**(11), 1069–1072 (1990). Also available by ftp from ftp://graph.ms.ic.ac.uk/pub/paper.txt.
79. J. C. Beck, A. J. Davenport, E. M. Sitarski and M. S. Fox, 'Texture-based heuristics for scheduling revisited', in *Proc. AAAI-97*, AAAI Press, Menlo Park, California, 1997.
80. J. C. Beck, A. J. Davenport, E. M. Sitarski and M. S. Fox, 'Beyond contention: extending texture-based scheduling heuristics', in *Proc. AAAI-97*, AAAI Press, Menlo Park, California, 1997.
81. W. Y. Chiang and M. S. Fox, 'Protection against uncertainty in a deterministic schedule', in *Proc. Int. Conf. on Expert Systems for Production and Operations Management*, 1990.
82. V. J. Leon, S. D. Wu and R. H. Storer, 'Robustness measures and robust scheduling for job shop', *IIE Trans*., **26**(5), 32–43 (1994).
83. H. Gao, 'Building robust schedules using temporal protection—an empirical study of constraint based scheduling under machine failure uncertainty', *Master's Thesis*, Department of Industrial Engineering, University of Toronto, 1995.
84. G. Laporte, 'The vehicle routing problem: an overview of exact and approximate algorithms', *Eur. J. Oper. Res*., **59,** 345–358 (1992).
85. P. Peasant, M. Gendreau and J. Rousseau, 'GENIUS-CP: A generic single-vehicle routing algorithm', in G. Smolka (ed.), *Proc. 3rd Int. Conf. on Principles and Practice of Constraint Programming* (*CP97*), Springer, Berlin, 1997.
86. J. C. Beck, 'Constraint-directed techniques for real-world scheduling', *Ph.D. Thesis*, University of Toronto, 1998, Forthcoming.
87. P. Ow and T. Morton, 'The single machine early/tardy problem', *Management Sci*., **35**(2), 177–191 (1989).
88. T. Morton, S. Lawrence, Rajagopolan and S. Kerre, 'SCHED-STAR: a price-based shop scheduling module', *J. Manuf. Oper. Management*, **1**(2), 131–181 (1988).
89. K. Baker and G. Scudder, 'Sequencing with earliness and tardiness penalties: A review', *Oper. Res*., **38**(1), 22–36 (1990).
90. B. Faaland and T. Schmit, 'Cost-based scheduling of workers and equipment in a fabrication and assembly shop', *Oper. Res*., **41**(2), 253–268 (1993).
91. J. N. Hooker, 'Testing heuristics: we have it all wrong', *J. Heuristics*, **1,** 33–42 (1996).

