ELSEVIER

Artificial Intelligence 117 (2000) 31–81

Artificial Intelligence

www.elsevier.com/locate/artint

Dynamic problem structure analysis as a basis for constraint-directed scheduling heuristics

J. Christopher Beck [a,*], Mark S. Fox [b,1]

[a] *ILOG, S.A. 9, rue de Verdun, BP 85 F-94253, Gentilly Cedex, France*

[b] *Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, ON, M5S 3G9, Canada*

Received 6 April 1999

Abstract

While the exploitation of problem structure by heuristic search techniques has a long history in AI (Simon, 1973), many of the advances in constraint-directed scheduling technology in the 1990s have resulted from the creation of powerful propagation techniques. In this paper, we return to the hypothesis that understanding of problem structure plays a critical role in successful heuristic search even in the presence of powerful propagators. In particular, we examine three heuristic commitment techniques and show that the two techniques based on dynamic problem structure analysis achieve superior performance across all experiments. More interestingly, we demonstrate that the heuristic commitment technique that exploits dynamic resource-level non-uniformities achieves superior overall performance when those non-uniformities are present in the problem instances.

Keywords: Scheduling; Heuristic search; Constraints; Problem structure

1. Introduction

The central thesis of this paper is that an understanding of the structure of a problem leads to high-quality heuristic problem solving performance. While exploration of this thesis has a history in the artificial intelligence literature [67] and constraint-directed scheduling in particular [33,34,63], much of the recent work in the latter area has concentrated on the use of propagation techniques [20–22,50,54,58]. Propagation techniques are a key component of constraint-directed scheduling algorithms, however,

* Corresponding author. Email: cbeck@ilog.fr.

[1] Email: msf@eil.utoronto.ca.

0004-3702/00/$ – see front matter
PII: S0004-3702(99)00099-5

it has been suggested that superior overall scheduling performance can be achieved with strong propagation and relatively weak heuristic commitment components [25,58,59,72]. In this paper, we evaluate heuristic commitment techniques in the presence of modern propagators and retraction techniques to investigate our dynamic search state analysis hypothesis and to evaluate these claims.

Our analysis of problem structure focuses on *texture measurements*: algorithms that implement dynamic analyses of each search state [34]. Texture measurements distill structural information from the constraint graph which is then used as a basis for heuristic decision making.

In this paper, we use the job shop scheduling paradigm to investigate our thesis. Given the well-studied nature of the job shop, there exist a number of heuristics that can be viewed as performing dynamic search state analysis to varying degrees. Our investigation examines the dynamic analysis of each of these heuristics and empirically evaluates the efficacy of each heuristic on a number of problem sets. The specific hypothesis that we investigate is that more in-depth dynamic search state analysis results in better overall heuristic search performance. By overall performance, we mean not simply higher quality heuristic commitments but also that the higher quality commitments lead to better performance in terms of fewer commitments, lower CPU time, and more problem instances solved.

1.1. Organization of paper

In the following section, we present the background to our studies including the notation used throughout the paper, a definition of job shop scheduling, and a brief description of the ODO scheduling framework, within which this work takes place. Also included in the section on the ODO framework are brief descriptions of the non-heuristic scheduling components (retraction techniques and propagators) used in our empirical studies. In Section 3 we present three heuristic commitment techniques for job shop scheduling. Two of these techniques (CBASlack and LJRand) are taken from the literature, while the other one is an extension of an existing texture measurement-based heuristic. In Section 4, we discuss the heuristic commitment techniques from the perspective of dynamic state analysis. Section 5 provides an overview of the empirical studies we undertake in this paper and the results of those studies are presented in Sections 6, 7, and 8. In-depth discussion of our results and of their meaning vis-a-vis the thesis of this paper is presented in Section 9. We conclude in Section 10.

2. Background

2.1. Notation

For an activity, A_i, and a set of activities, S, we use the notation in Table 1 through the balance of this paper. We will omit the subscript unless there is the possibility of ambiguity.

Table 1
Notation

Symbol	Description
ST_i	A 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

2.2. 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:

- 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;
- disjunctive resource constraints specifying that no 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 (or *makespan*) 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 [37]. A recent survey of techniques for solving the job shop scheduling problem can be found in [18].

2.3. The ODO framework

The ODO framework is the central cognitive and implementational tool developed in the ODO Project [11,27,28]. It provides concepts with which to model, compare, and understand constraint-directed scheduling algorithms. The ODO framework is based around the constraint graph representation of a scheduling problem, a sub-framework describing scheduling *policies* or strategies, and the unifying concept of search through the assertion and retraction of commitments. A high-level view of the ODO framework is shown in Fig. 1.

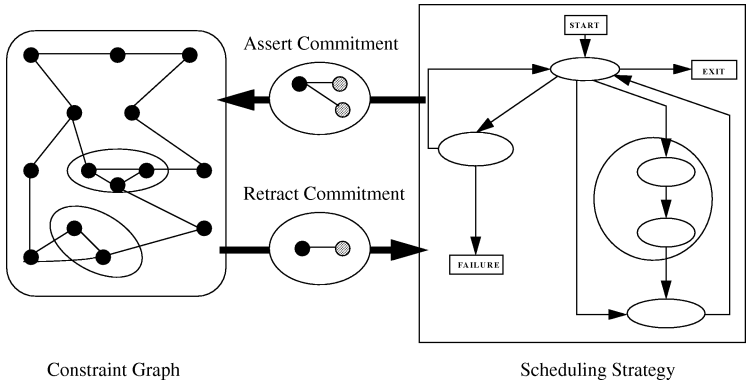

Fig. 1. A high-level view of the ODO framework.

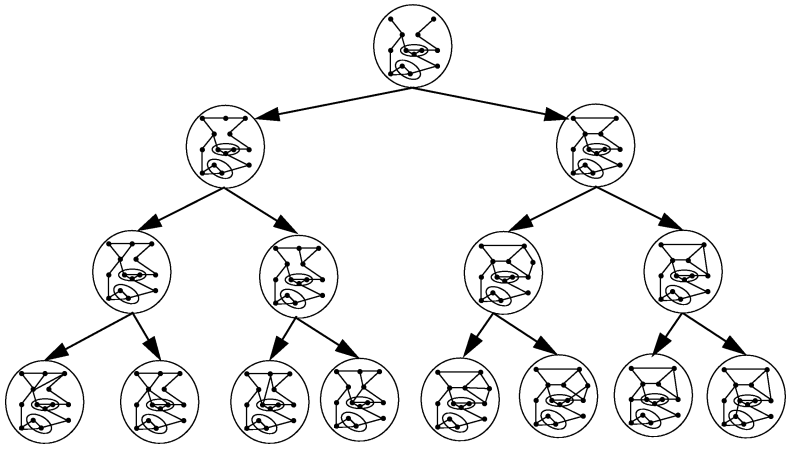

Fig. 2. A conceptual four-level constraint-directed search tree.

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. Fig. 2 displays our conceptual model of a constraint graph as the representation of each state in a search tree. As represented in Fig. 2, the state transitions are achieved by the modification of the constraint graph through the assertion and retraction of commitments. Examples of commitments in scheduling include:

- 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).
- posting a precedence constraint between activities (e.g., activity A executes before activity B) [25,72].
- 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.

- 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.

Central to this paper are the constraint graph representation and the search policy. We discuss these two concepts in this section. In-depth discussion of the ODO framework can be found in [10,11,15].

2.3.1. The constraint graph representation

As indicated in Fig. 2, 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 are instances of variables and constraints as usually understood in the constraint satisfaction problems [29,46]. In particular, we represent 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 [2], and resources and inventories with minimum and maximum constraints. The higher level constraint graph components can be expressed as, and are logically equivalent to, sets of basic constraint/variable components. For example, a unary resource can be represented by a clique of disjunctive constraints among the start and end time variables of a set of activities [5]. There are two main reasons that higher level objects are explicitly represented:

(1) Software engineering—Given that we are explicitly interested in the representation and the search for a solution to scheduling problems, the ability to directly represent scheduling level concepts is a significant benefit to research and development.
(2) Search efficiency—More importantly, it has been demonstrated [58,61,62,73] that the use of global constraints (as represented, for example, by a unary resource) rather than the basic local constraint representation (e.g., a clique of disjunctive constraints) can result in significant gains in constraint propagation and corresponding reductions in search effort.

The components of the constraint graph are not an innovation of the ODO model as most constraint-directed scheduling systems represent these concepts (e.g., [22,33,48,49,65,70, 76]).

2.3.2. The scheduling policy

A policy contains the components displayed in Fig. 3. The commitment assertion component is trivial as it requires adding a constraint to the existing graph. 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

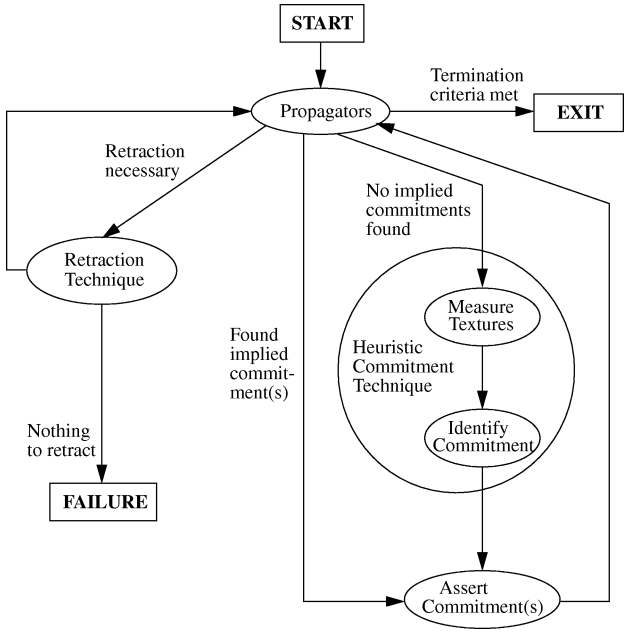

Fig. 3. Schematic of a policy.

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 criterion* 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.

This paper focuses on heuristic commitment techniques and so we return to them in Section 3. In the balance of this section we discuss the propagators and retraction techniques that are used in our empirical studies.

Propagators. A propagator is a function that analyzes the current search state to determine constraints that are implied by, but are not explicitly present in, the constraint graph. By making these constraints explicit, we can use them to prune the number of possibilities to be explored in the search. 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 both reduces the search space and often enables further constraints to be inferred.

Examples of propagators from a CSP perspective include the various consistency enforcement algorithms such as arc-consistency and k-consistency [35,36,53]. 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 (e.g., a unary "not-equals" constraint).

Many powerful propagation techniques have been developed for constraint-directed scheduling in recent years, for instance, variations of edge-finding [20,22,50,58] and shaving [21]. It has long been known that search can be drastically reduced by enforcing various degrees of consistency [36]. The effort to achieve high degrees of consistency, however, appears to be at least as expensive as more traditional algorithms.

In our empirical studies in this paper, we use four propagators. As our focus is on heuristic commitment techniques, these four propagators are used in all experimental conditions.

- Temporal Propagation—Temporal propagation enforces arc-B-consistency [52] on the temporal constraints in the scheduling problem. For example, 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: $est_j \geqslant est_i + dur_i$ and $lft_i \leqslant lft_j - dur_j$. Arc-B-consistency (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).
- Constraint-Based Analysis—Constraint-Based Analysis (CBA) [25,30,31,72] enforces arc-B-consistency on unary resource constraints. By the examination of the possible time windows for the execution of each pair of activities, CBA is able, in some circumstances, to find implied precedence constraints or dead-ends.
- Edge-Finding Exclusion—Edge-finding exclusion [20–23,58] examines subsets of activities on a resource and the time windows within which all the activities in the subset must execute. Under some conditions, edge-finding exclusion is able to derive new lower bounds on the start times of activities, new upper bounds on the end times of activities, and/or dead-ends.
- Edge-Finding Not-First/Not-Last—Edge-finding not-first/not-last [6,8,20–22,58] also examines subsets of activities on a resource and their time windows. Under some conditions, edge-finding not-first/not-last is able to infer that an activity cannot execute first (or last) of the activities in the subset. This inference allows the derivation of new upper (or lower) bounds on the time window of an activity and in some cases leads to the detection of a dead-end in the search.

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 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 $i < j < k$?

Different retraction techniques answer these questions in different ways [11]. In the empirical studies in this paper, we examine two retraction techniques in different experimental conditions:

- Chronological Backtracking—Chronological backtracking retracts 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. The question of intervening commitments is moot as there are none if the most recent commitment is retracted.
- Limited Discrepancy Search—Limited Discrepancy Search (LDS) [43,44] 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.

2.4. Texture measurements

The hypothesis of this paper is that heuristic commitment techniques based on the dynamic analysis of each search state lead to high quality overall heuristic search performance. To test our hypothesis it is necessary to be more precise about search space analysis. Given the constraint graph representation of a search state, an analysis of a search state corresponds to measurements of its constraint graph representation. A *texture measurement*, therefore, 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 [34,63], 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.

In non-technical usage, the word “texture” denotes a surface characteristic of an object. In contrast, “structure” is usually taken to indicate internal, hidden components of an object. This is precisely the meanings we wish to evoke by the use of the term texture measurement. Our chief hypothesis is that understanding of the structure of a problem is important for successful heuristic search. However, problem structure is, in many cases, internal and hidden. The purpose of texture measurements is to bring some of the hidden, structural information to the surface so that it can be easily accessed by heuristic commitment techniques. Texture measurements transform the hidden structural information into accessible, surface information.

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 [34]. 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 the 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.

3. Heuristic commitment techniques

In this section, we present three heuristic commitment techniques. The first, SumHeight, is an adaptation of the ORR/FSS heuristic due to Sadeh [63,66]. The other two heuristic commitment techniques, CBASlack and LJRand, are directly taken from the literature as described below. We also note some criticisms of texture-based heuristic arising from previous experimental work with CBASlack and LJRand.

3.1. SumHeight: A texture measurement-based heuristic

The SumHeight heuristic commitment technique is based on two texture measurements: contention and reliance [63]. These two texture measurements are estimated for each search state and then a heuristic commitment is chosen based on the distilled information.

Contention is used to identify the most critical resource and time point. Reliance is then used to identify the two activities that rely most on that resource and time point. The intuition is that by focusing on the most critical resource and activities, we can make a commitment that reduces the likelihood of reaching a search state where the resource is over-capacitated. Furthermore, once such critical decisions are made the problem is likely to be decomposed into simpler sub-problems.

3.1.1. Estimating contention and reliance

Contention is the extent to which variables linked by a disequality constraint compete for the same values. In the context of job shop scheduling, contention is the extent to which activities compete for the same resource over the same time interval.

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

In the SumHeight heuristic, reliance is estimated by an activity's probabilistic individual demand for a resource over time, while contention is the aggregation of the individual activity demands.

If an activity does not require a resource, it has no demand for it, so its demand is 0. Otherwise, to calculate an activity's individual demand, a uniform probability distribution over the possible start times of the activity is assumed: each start time has a probability of $1/|STD|$. (Recall that STD is the domain of the activity's start time variable. A uniform probability distribution is the "low knowledge" default. It may be possible to use some local propagation in the constraint graph to find a better estimate of the individual demand [56,63].) 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:

$$ID(A, R, t) = \sum_{t - dur_A < \tau \leqslant t} \sigma_A(\tau), \tag{1}$$

where

$$\sigma_A(\tau) = \begin{cases} \dfrac{1}{|STD_A|}, & \tau \in STD_A, \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

In the straightforward implementation of the individual demand, $ID(A, R, t)$ is calculated for each time point, t, $est_A \leqslant t < lft_A$. The time complexity of this calculation relies not only on the number of activities and resources, but also on the length of the scheduling horizon. To prevent such scaling, we use an event-based representation and a piece-wise linear estimation of the ID curve. The individual activity demand is represented by four (t, ID) pairs:

$$\left(est, \frac{1}{|STD|}\right), \quad \left(lst, \frac{min(|STD|, dur)}{|STD|}\right), \quad \left(eft, \frac{min(|STD|, dur)}{|STD|}\right), \quad (lft, 0). \tag{3}$$

The individual demand level between any two points on the ID curve is interpolated with a linear function.

To estimate contention, the individual demands of each activity are aggregated for each resource. Aggregation is done by summing the individual activity curves for that resource. This aggregate demand curve is used as a measure of the contention for the resource over time.

For example, given the activities in Fig. 4, the individual demand curves for each activity and the aggregate resource curve are displayed in Fig. 5.

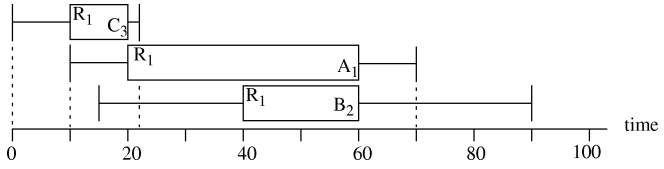

Fig. 4. Activities A_1, B_2, and C_3.

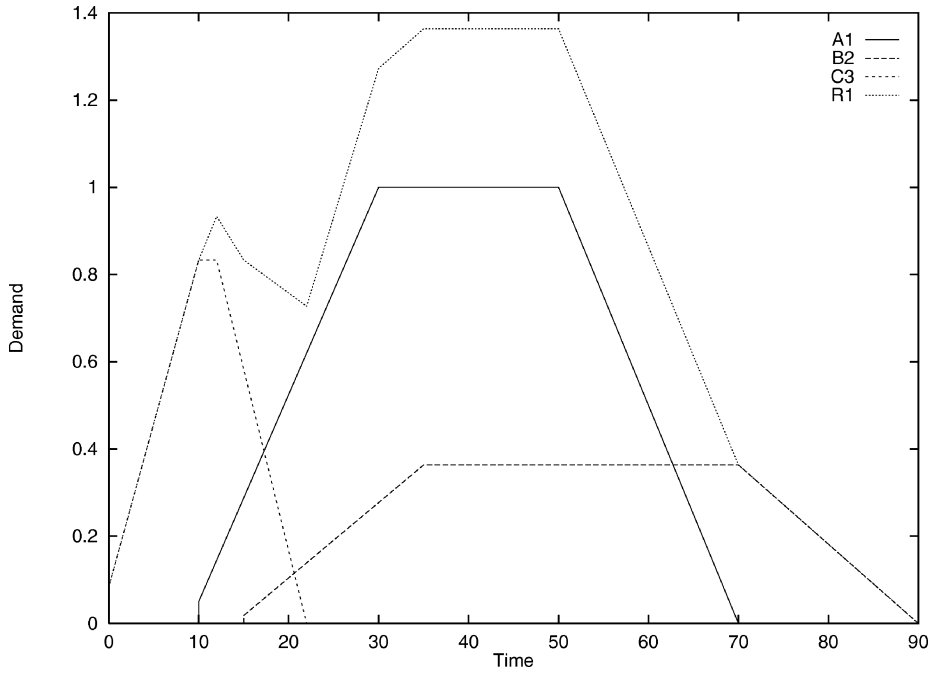

Fig. 5. Event-based individual demand curves (A_1, B_2, C_3) and their aggregate curve (R_1).

3.1.2. Heuristic commitment selection

Given the aggregate and individual demand curves, a heuristic commitment for this search state must be selected. SumHeight makes a commitment on the activities most reliant on the resource for which there is the highest contention. In more detail, SumHeight does the following:

(1) Identifies the resource and time point with the maximum contention.
(2) Identifies the two activities, A and B, which rely most on that resource at that time and that are not already connected by a path of temporal constraints.
(3) Analyzes the consequences of each sequence possibility ($A \rightarrow B$ and $B \rightarrow A$) and chooses the one that appears to be superior based on our sequencing heuristics (see Section 3.1.4).

3.1.3. Finding the critical activities

After the aggregate demand curves are calculated for each resource, we identify the resource, R^*, and time point, t^*, for which there is the highest contention (with ties broken arbitrarily). We then examine the activities that contribute individual demand to R^* at t^*. The two critical activities, A and B, are selected as follows:

- A is the activity with highest individual demand for R^* at t^* which is not yet sequenced with all activities executing on R^* (i.e., at least one activity executing on R^* is not yet connected to A via a path of temporal constraints).

- B is the activity not yet sequenced with A with highest individual demand for R^* at t^*.

Because these two activities contribute the most to the aggregate demand curve, they are the most reliant on the resource at that time.

It can be seen in Fig. 5 that one of the critical time points on R_1 is 35. There are only two activities that contribute to this time point, as C_3's latest end time is 22. Therefore, A_1 and B_2 are selected as the critical activities.

3.1.4. Sequencing the critical activities

To determine the sequence of the two most critical activities, we use three heuristics: MinimizeMax, Centroid, and Random. If MinimizeMax predicts that one sequence will be better than the other, we commit to that sequence. If not, we try the Centroid heuristic. If the Centroid heuristic is similarly unable to find a difference between the two choices, we move to Random.

MinimizeMax sequencing heuristic. The intuition behind the MinimizeMax (*MM*) sequencing heuristic is that since we are trying to reduce contention, we estimate the worst case increase in contention and then make the commitment that avoids it. *MM* identifies the commitment which satisfies the following:

$$MM = \min\bigl(max_{AD'}(A, B), max_{AD'}(B, A)\bigr), \tag{4}$$

where

$$max_{AD'}(A, B) = \max\bigl(AD'(A, A \to B), AD'(B, A \to B)\bigr). \tag{5}$$

$AD'(A, A \to B)$ is an estimate of the new aggregate demand at a single time point. It is calculated as follows:

- Given $A \to B$, we calculate the new individual demand curve of A and identify the time point, *tp*, in the individual demand of activity A that is likely to have the maximum increase in height. [2] This leaves us with a pair: {*tp*, Δ*height*}.
- We then look at the aggregate demand curve for the resource at *tp* and form $AD'(A, A \to B)$ by adding Δ*height* to the height of the aggregate demand curve at *tp*.

The same calculation is done for $AD'(B, A \to B)$ and the maximum (as shown in Eq. (5)) is used in $max_{AD'}(A, B)$. Eq. (4) indicates that we choose the commitment resulting in the lowest maximum aggregate curve height.

Centroid sequencing heuristic. The centroid of the individual demand curve is the time point that equally divides the area under the curve. [3] We calculate the centroid for each activity and then commit to the sequence that preserves the current ordering (e.g., if the centroid of A is at 15 and that of B is at 20, we post $A \to B$). Centroid is a variation of a heuristic due to [56].

Random sequencing heuristic. Randomly choose one of the sequencings.

[2] Subsequent propagation may have an impact on A's individual demand curve after a heuristic commitment. As a consequence, we do not calculate the actual increase in height of A, but rather estimate it based on temporal arc consistency of the to-be-posted precedence constraint (i.e., we find A's new *lft* based on local arc consistency propagation and use it to calculate the maximum change in the individual demand).

[3] This is a simplification of centroid that is possible because the individual activity curves, as we calculate them, are symmetric.

3.1.5. Complexity

The worst-case time complexity to find a heuristic commitment at a problem state is due to the aggregation of the demand curves for each resource and the selection of the critical activities. By storing the incoming and outgoing slopes of the individual curves at each point, we can sort the event points from all activities and then, with a single pass, generate the aggregate curve. This process has complexity of $O(mn \log n)$ (where n is the maximum number of activities on a resource and m is the number of resources). Selection of the pair of unsequenced activities on the resource requires at worst an additional $O(n^2)$. Thus the overall time complexity for a single heuristic commitment is $O(n^2) + O(mn \log n)$.

The space complexity is $O(mn)$ as we maintain an individual curve for each activity and these individual curves make up the aggregate curve for each resource.

3.1.6. The SumHeight heuristic versus the ORR/FSS heuristic

The SumHeight heuristic is an adaptation of the Operation Resource Reliance/Filtered Survivable Schedules (ORR/FSS) [63,64,66]. In particular, the use of contention and reliance and the intuitions behind the heuristic commitment are the same in the two techniques. The differences between the two are as follows:

(1) SumHeight uses an the event-based, linear interpolation implementation of the demand curves.
(2) SumHeight uses a single time point to identify the resource with maximum contention.
(3) SumHeight generates a heuristic commitment that sequences the two most critical activities.

In contrast, the ORR/FSS heuristic identifies the most critical resource and *time interval* in a search state. An interval equal to the average activity duration is used and the area under the aggregate curve for all such intervals is calculated. The most contended-for resource and time interval is defined to be the resource and time interval with the highest area under the aggregate demand curve. The activity not yet assigned to a start time that contributes the most area to the critical time interval is the most critical activity. This is the activity that the ORR heuristic predicts is important to schedule at this point in the search: it is the most reliant activity on the most contended-for {resource, time interval} pair.

A heuristic commitment is found by assigning the critical activity, A, to its most "survivable" start time. The most survivable start time is found by the FSS portion of the heuristic which rates each of its possible start times of the critical activity by using the demand curves. This rating takes into account the effect an assignment to A will have both on activities competing directly with A and on those temporally connected to A. A more detailed description of the start time assignment heuristic is beyond the scope of this document; interested readers are referred to [63,66].

3.2. The Constraint-Based Analysis Slack heuristic

The Constraint-Based Analysis Slack (CBASlack) heuristic forms the heuristic component of the Precedence Constraint Posting (PCP) scheduling algorithm [25,72].

Bslack (Eq. (8) below) is calculated for all pairs of activities that compete for the same resource. The pair with the smallest *Bslack* value is identified as the most critical. Once the

critical pair of activities is identified, the sequence that preserves the most slack is the one chosen. The intuition here is that a pair with a smaller *Bslack* is closer to being implied than one with a larger value. Once we have identified this pair, it is important to leave as much temporal slack as possible in sequencing them in order to decrease the likelihood of backtracking.

$$slack(i \rightarrow j) = lft_j - est_i - (dur_i + dur_j), \tag{6}$$

$$S = \frac{\min(slack(i \rightarrow j), slack(j \rightarrow i))}{\max(slack(i \rightarrow j), slack(j \rightarrow i))}, \tag{7}$$

$$Bslack(i \rightarrow j) = \frac{slack(i \rightarrow j)}{\sqrt{S}}. \tag{8}$$

The intuition behind the use of $\sqrt{S}$ is that there is search information in both the minimum and maximum slack values. While choosing activity pairs based on minimum slack remains a dominant criterion, the authors hypothesize that more effective search guidance may be achieved with a biasing factor that will tend to increase the criticality of activity pairs with similar minimum and maximum slack values. For example, it is unclear if an activity pair with a minimum slack of 5 and a maximum slack of 5 is more or less critical than a pair with minimum slack 3 and maximum slack 20. S (Eq. (7)) is a measure of the similarity of the minimum and maximum slack values for an activity pair and it is incorporated in the biased-slack calculation as shown in Eq. (8).

3.2.1. Complexity

The worst-case time complexity for the CBASlack heuristic is $O(mn^2)$ (where n is the maximum number of activities on a resource and m is the number of resources). For each resource, each pair of activities must be evaluated to determine its biased-slack value.

3.3. The Randomized Left-Justified heuristic

In the Randomized Left-Justified heuristic (LJRand) [58,59] the set of activities that can execute before the minimum earliest finish time of all unscheduled activities is identified. One of the activities is randomly selected and scheduled at its earliest start time.

In the job shop scheduling context, a new commitment can be inferred when a heuristic commitment made by LJRand is undone by chronological backtracking (or another provable retraction technique). Because a dead-end results when the activity, A, is assigned its earliest start time, the earliest start time of A can be updated to the smallest earliest finish time of all other unscheduled activities on the same resource as A. The provable backtracking has shown that some other activity must come before A in the resource in question [58].

3.3.1. Complexity

The time complexity of LJRand is $O(mn)$. At worst the start and end times of all activities must be evaluated to find the minimum end time of all unscheduled activities and then to randomly choose the activity to start first.

4. Problem structure and heuristic search

Problem structure can be broadly defined to be the relationships among the problem objects. Study of problem structure concerns the identification of structure and its correlation to the performance of heuristic search techniques. Given a representation of a problem, are there relationships among the components of the representation that can be identified and exploited by heuristic search techniques? For example, it is well known that constraint satisfaction problems (CSPs) with a tree structured constraint graph are solvable in polynomial time [36]. Similarly, algorithm-independent measures of problem difficulty have been developed based on the literal/clause ratio in SAT problems [55].

4.1. The problem structure hypothesis

The importance of the understanding of problem structure for heuristic search can be traced (within the AI community) at least as far back as the early work of Simon [67] and has been examined from a constraint perspective by Fox et al. [33,34,63]. Given this history, the question arises as to whether the problem structure hypothesis is, indeed, still in doubt. Dynamic variable ordering heuristics based on domain size, number of constraints, and other structural characteristics are relatively well established in the constraint-satisfaction community [38,42,68]. Despite the acknowledgement that such orderings have positive impact on heuristic search, however, much of the recent research has focused away from such heuristics. The majority of work in constraint research over the past ten years has examined the creation of efficient propagation techniques for global constraints. Global constraints are used to represent a sub-problem and have been shown in a number of applications to contribute significantly to problem solving ability [20–22, 50,54,58,61,62,73]. It is fair to say that most of the advances on constraint-directed search over the past ten years have arisen from the use of global constraints.

We are interested in further investigation of the problem structure hypothesis in the context of modern constraint techniques for two main reasons. The first is the basic question as to whether heuristics based on problem structure are still necessary given the use of global constraints. We believe, following the problem structure hypothesis, that such heuristics are complementary to the use of global constraints. It is not obvious, *a priori*, that this belief is correct. Indeed, some of the global constraint work can be interpreted as supporting the position that sophisticated heuristics are no longer necessary given strong global constraint propagation [25,58,59,72].

Our second motivation for the investigation of the problem structure hypothesis is the realization that specific application areas for constraint-directed search exhibit significantly more problem structure than generic constraint satisfaction problems. Given the structure that comes with such areas as scheduling, resource allocation, and configuration, it does not appear that analysis based on domain size or the number of constraints is really identifying important problem structure. Given the significantly richer structure of problems within specific domains, we have the opportunity to perform a much deeper structural analysis. The question we want to investigate, then, is precisely the problem structure hypothesis: does the analysis and exploitation of problem structure lead to superior heuristic search performance?

4.2. Problem structure and scheduling

The challenge in any specific problem domain, then, is to identify problem structure that can be exploited by heuristic search techniques. In scheduling, problem structure has typically referred to examination of a problem state in order to determine the load on each resource. It is a well-known industrial heuristic to identify the "bottleneck" resource, that is, the one with the highest usage. Effort can then be spent in scheduling the activities on the bottleneck.

In constraint-directed scheduling, ISIS [33] used dynamic analysis of resource usage to temporally constraint activities within an order. ISIS followed an order-based problem decomposition where the orders were prioritized based on their static properties. When an order was selected for scheduling, the resource usage was dynamically evaluated based on the already scheduled activities. The activities in the selected order were then temporally constrained (via dynamic programming) to execute at times when their resources were available while ensuring that the order due date would be satisfied. Each activity in the order was assigned resources and start-times before the analysis was performed again. OPIS [69,71] built on this analysis by opportunistically choosing a resource-based or order-based problem decomposition. Based on an analysis of resource usage, OPIS could schedule all activities in an order (like ISIS) or all activities competing for the same bottleneck resource. One of the critical advances in terms of problem structure analysis in OPIS is the fact that bottlenecks resources are not simply identified at the beginning of the problem solving process. Rather after scheduling a resource or an order, the resource levels were re-analyzed.

As described above, in the ORR/FSS heuristic of MicroBoss [63,66] a probabilistic estimate of the activity demand was used to estimate the aggregate demand for each resource. Following ISIS and OPIS, the major contribution of the ORR/FSS heuristic was the incorporation of micro-opportunistic search. Rather than following the order- or resource-based decomposition, micro-opportunistic search makes each activity an individual choice point. After a heuristic commitment is made to assign a start time to an activity, the aggregate demand estimations are re-calculated.

As scheduling performance evolved through ISIS, OPIS, and ORR/FSS it was demonstrated that both more detailed problem structure analysis and finer control of the heuristic search process based on the problem structure analysis results in better heuristic search performance. Not only does ORR/FSS re-analyze the problem state more frequently, it is also guided more strongly by the results of the analysis: rather than having to focus on a critical resource or critical order, ORR/FSS focuses on critical activities. Muscettola [57] further demonstrated that worse overall performance results when either the frequency of the recomputation of search state information is lower or when that information is ignored.

4.2.1. Criticisms of dynamic problem structure analysis

Other work, however, has called into question the efficacy of using in-depth problem structure analysis as a basis for heuristic commitment techniques.

As part of the evaluation of the PCP scheduling algorithm, Smith and Cheng [25, 72] compare the ORR/FSS heuristic (using temporal and resource propagation, but

no retraction technique) with the CBASlack heuristic (using the CBA propagator (Section 2.3), temporal propagation, and no retraction). The results, on a set of job shop scheduling benchmarks, indicated that equal or better performance is achieved with the CBASlack heuristic. Given the equal performance and the relative simplicity of the CBASlack heuristic compared with ORR/FSS, the authors conclude that the more complicated implementation associated with in-depth problem structure analysis is not justified.

The SOLVE scheduling algorithm [58] consists of LJRand, a set of sophisticated propagators including edge-finding exclusion (Section 2.3), and bounded chronological backtracking with restart as the retraction technique.[4] On a set of difficult job shop benchmark problems from the Operations Research literature [9], SOLVE significantly outperformed both Sadeh's ORR/FSS algorithm (using chronological retraction) and the ORR/FSS heuristic augmented with the propagators used in SOLVE but still using chronological retraction [58].

Both of these empirical results appear to be evidence against the use of dynamic search state analyses as a basis for heuristic commitment techniques: dynamic analysis of the problem structure (as implemented by ORR/FSS) appears to be counterproductive in terms of overall scheduling performance. In the former case, CBASlack does perform some analysis of the problem structure through the calculation of slack values on all pairs of activities. This analysis, however, concentrates on pair-wise relationships while the ORR/FSS heuristic aggregates information from all activities competing for a resource over time. If correct, these results indicate that either the aggregate information provides no advantage and/or that the ORR/FSS heuristic is not able to exploit the advantage that is possible through the extra information. The latter result is even more damaging to dynamic problem structure analysis. The LJRand heuristic randomly searches the space of "active" schedules [4] performing little, if any, search state analysis.

The flaw in both pieces of research is the authors' model of scheduling algorithms. Rather than viewing the algorithms as composed of components, the researchers treat them as monolithic wholes. As a consequence, the heuristic commitment technique is not the only component that is varied among algorithms. In the case of Smith and Cheng [25,72], the CBA propagator was used as part of the PCP algorithm, but not as part of the ORR/FSS algorithm. Similarly, though Nuijten [58] uses the same propagators in SOLVE and in the augmented ORR/FSS, the retraction techniques are different: SOLVE uses bounded chronological backtracking with restart while ORR/FSS uses chronological backtracking. Given these differences, to what should we attribute the observed experimental results? Is it really the case that the CBASlack heuristic achieves equal performance to ORR/FSS, or is the quality of the PCP algorithm due to the combination of the CBASlack heuristic with the CBA propagator? Does LJRand really significantly outperform ORR/FSS, or does the performance difference arise from the different retraction techniques?

[4] Bounded chronological backtracking performs a limited number of chronological backtracks before restarting the search completely. The randomized nature of the heuristic commitment technique will tend to minimize repeated visits to the same portion of the search space.

4.3. Evaluating the heuristic value of problem structure analysis

To further investigate the importance of knowledge of problem structure to heuristic search performance and to investigate the criticisms of problem structure based heuristics noted above, we have undertaken an empirical study of scheduling heuristics. In this paper, we focus on the three heuristics described above: SumHeight, CBASlack, and LJRand.

SumHeight, like ORR/FSS, is based on using the aggregation of the probabilistic individual demand of each activity to estimate the contention for a resource. Of the three heuristics, it makes the most use of dynamic information distilled from the constraint graph in forming heuristic decisions.

CBASlack, though not originally conceptualized as a texture-based heuristic, uses a form of the contention texture to form commitments. CBASlack calculates the biased-slack between all pairs of activities that compete for the same resource and selects the pair with minimum biased-slack as the most critical. We view the pair-wise biased-slack as an estimation of pair-wise contention: the lower the biased-slack the more the two activities contend for the same resource over the same time window. While pair-wise contention distills less information from the constraint graph than the more aggregate measure used in SumHeight, it is nonetheless a dynamic analysis of the search state. Note that pair-wise contention is a measure of how much a pair of activities compete for a resource while the SumHeight texture is a measure of the extent to which all activities compete for a resource. So while SumHeight and CBASlack are both based on forms of contention, they are not based on an identical texture measurement: we expect that the difference in aggregation level will lead to different heuristic performance.

Finally, the LJRand heuristic identifies the unassigned activities that are able to start before the minimum end time of all unassigned activities. One of these activities is randomly selected and assigned to its earliest start time. LJRand performs the least analysis of the search state of the three heuristics tested as it randomly searches the space of active schedules.

Given these heuristics with differing dynamic analysis characteristics, we want to evaluate the extent to which analysis of problem structure via texture measurements correlates with the quality of overall scheduling performance. More specifically, we want to evaluate the heuristic importance of dynamic information about probabilistic resource levels. As noted, SumHeight is specifically designed to exploit such information while CBASlack exploits similar information at the level of activity pairs. LJRand is not concerned at all with such information.

5. Empirical studies

While the primary purpose of our empirical studies are to evaluate the use of dynamic problem structure analysis as a basis for heuristic commitment techniques, we do not limit our experiments to this question. The empirical data can be used more broadly to provide insight into search behavior. In particular, we use two retraction techniques (chronological backtracking and LDS) in separate experimental conditions. While the two conditions provide richer data upon which to base the analysis of the heuristic commitment

techniques, the data can also be used to compare performance of the two retraction techniques. While LDS has been used in the context of scheduling [43,44], this is the first work of which we are aware that compares LDS and chronological backtracking in the presence of state-of-the-art constraint-directed heuristic and propagation techniques.

In the rest of this paper, we present and analyze empirical results of three experiments. Experiment 1 uses a well-known benchmark set of job shop scheduling problems. It provides a baseline for the comparison of scheduling performance without the specific manipulation of problem characteristics. This is a subset of the problem set used by Nuijten [58] and so the experiment will evaluate the extent that the experimental flaws noted above contributed to those previous results. Experiment 2 uses randomly generated job shop scheduling problems. The independent variable is the size of the problem. Clearly, problem size is a relevant factor in the evaluation of scheduling performance. In the final experiment, Experiment 3, we turn directly to the manipulation of resource usage. By introducing bottleneck resources we directly address the importance of problem structure information.

5.1. The reporting of time-outs

The experiments in this paper are run with a bound on the CPU time. Each algorithm must either find a schedule or prove that no schedule exists for a problem instance within that bound. If an algorithm is unable to do so within a limit on the CPU time (in all our experiments the limit is 20 minutes), a time-out is recorded. A time-out indicates that the algorithm was unable to find a solution or prove that no solution exists for a particular scheduling problem instance.

The primary reason for reporting time-out results is that it allows us to use problem sets that contain both soluble and insoluble (over-constrained) problems. The phase transition work in combinatorial problems such as SAT and CSPs [39,40] demonstrates that the hardest problem instances are found in locations of the problem space where approximately half of the problems are over-constrained. A “hard instance” is one that cannot be easily shown to be either over-constrained or soluble: significant search effort is required to find a solution or show that none exists. While the space of scheduling problems is not as well-understood as SAT or CSP in terms of phase transition phenomena [17], we want to take advantage of this insight in order to generate challenging problem instances. We construct our problem sets so that as the independent variable varies, the problem instances move from an over-constrained area in the problem space to an under-constrained area. In the former area, proofs of insolubility can often be easily found while in the latter area, solutions can be easily found. It is in the middle range of problems where we expect to find the most difficult problems.

The use of time-outs as a search statistic allows us to integrate search performance on over-constrained problems and soluble problems into a single statistic. The intuition is that algorithms fail when they can find neither a solution nor a proof of insolubility. By using the number of failures, in this way, we get a clearer picture of the algorithm performance. For example, plotting the number of problems for which a solution is found obscures the fact that some algorithms may be performing very well on over-constrained problems (by finding proofs of insolubility) whereas others are not able to find any such proofs.

Table 2
Summary of experimental scheduling algorithms

Strategy	Heuristic commitment technique	Propagators	Retraction techniques
SumHeightChron	SumHeight	All[a]	Chronological backtracking
CBASlackChron	CBASlack	All	Chronological backtracking
LJRandChron	LJRand	All	Chronological backtracking
SumHeightLDS	SumHeight	All	LDS
CBASlackLDS	CBASlack	All	LDS
LJRandLDS	LJRand	All	LDS

[a] Temporal propagation, edge-finding exclusion, edge-finding not-first/not-last, and CBA.

5.2. Instantiations of the ODO framework

The scheduling algorithms that we evaluate are instantiations of a scheduling strategy in the ODO framework. As our primary purpose in these experiments is to evaluate the efficacy of heuristic commitment techniques, the only difference among the strategies in our experiments is the heuristic commitment technique. Specifically, we hold the set of propagators constant across all experiments and create two experimental conditions based on the retraction technique.

Heuristic commitment techniques. We use three heuristic commitment techniques in our experiments: SumHeight, CBASlack, and LJRand.

Propagators. Four propagators are used in the following order: temporal propagation, edge-finding exclusion, edge-finding not-first/not-last, and CBA (see Section 2.3.2).

Retraction techniques. We use two retraction techniques in two separate experimental conditions: chronological backtracking and Limited Discrepancy Search (LDS) (see Section 2.3.2).

Termination criterion. The termination criterion for all experiments is to find a solution by fully sequencing the activities on each resource, or to exhaust the CPU limit. The CPU time limit for all experiments is 20 minutes on a Sun UltraSparc-IIi, 270 MHz, 128 M memory, running SunOS 5.6. If an algorithm exhausts the CPU time on a problem, it is counted as a failure to solve the problem.

Table 2 displays a summary of the experimental scheduling strategies.

6. Experiment 1: Operations Research library

6.1. Problem set

The basic set of problems for the first experiment is a set of 20 job shop scheduling problems (Table 3) from the Operations Research library of benchmark problems [9]. The problems are the union of the problem sets used in [75] and [7] with the exception of one problem (la02) which was removed as it was small and easily solved by all algorithms.

Table 3
Test problems

Source	Problem(lower bound)
Adams et al. [1]	abz5(1234), abz6(943)
Fisher & Thompson [32]	ft10(930)
Lawrence [47]	la19(842), la20(902), la21(1046), la24(935), la25(977), la27(1235), la29(1142), la36(1268), la37(1397), la38(1196), la39(1233), la40(1222)
Applegate & Cook [3]	orb01(1059), orb02(888), orb03(1005), orb04(1005), orb05(887)

In order to generate a problem set with problems of varying difficulty, we took the set of 20 problems and the known optimal or lower-bound makespan, and created a total of six problem sets by varying the makespan of the problem instances. Recall that, in the classical job shop optimization problem, the goal is to find the minimum makespan within which all activities can be scheduled (see Section 2.2). Rather than adopting an optimization approach, we attempt to satisfy each problem instance at different makespans. This data provides more information on the performance of each algorithm across problems with a range of difficulties. We generate problems with varying makespans by using the *makespan factor*, the primary independent variable in this experiment. It is the factor by which the optimal or lower-bound makespan is multiplied to give the makespan within which we attempt to solve the problem instances. In this experiment, the makespan factor is varied from 1.0 (the optimal makespan) to 1.25 (25% greater than the optimal makespan) in steps of 0.05, producing six sets of 20 problems each.

6.2. Results

Fig. 6 displays the fraction of problems in each problem set for which each algorithm (using chronological backtracking) was unable to find a solution. Fig. 7 presents the same data for the algorithms using LDS. Overall, with either retraction technique, LJRand times out on significantly more problems than either CBASlack or SumHeight, while there is no significant difference between SumHeight and CBASlack.[5] In terms of the comparison between the retraction techniques, the only significant difference was for LJRand which was able to find solutions to significantly more problems when using LDS than when using chronological backtracking.

Figs. 8 and 9 show the mean CPU time for the chronological backtracking algorithms and the LDS algorithms respectively. As with the number of problems timed out, overall there is no significant difference between SumHeight and CBASlack while both perform significantly better than LJRand. These results hold regardless of retraction technique. In comparing the retraction techniques, we see significantly larger mean CPU times with chronological retraction than with LDS. The magnitude of the improvement for heuristics indicates that LJRand has a larger improvement in moving to LDS from chronological backtracking than the other two heuristics.

[5] Unless otherwise noted, statistical tests are performed with the bootstrap paired-t test [26] with $p \leqslant 0.0001$.

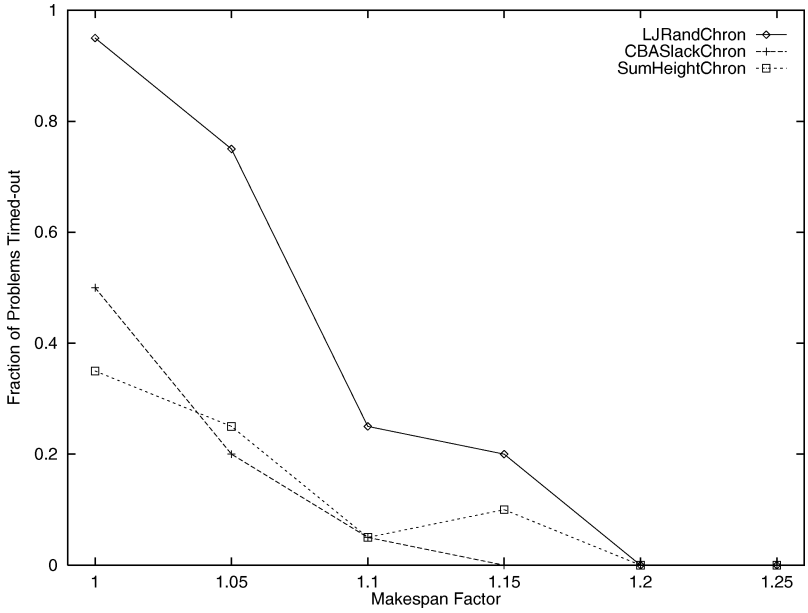

Fig. 6. The fraction of problems in each problem set for which each algorithm timed out (chronological backtracking).

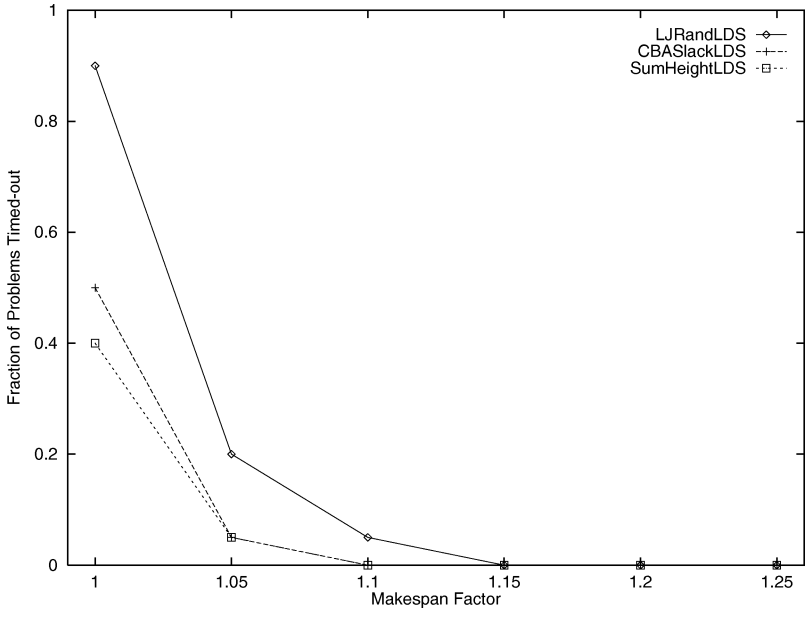

Fig. 7. The fraction of problems in each problem set for which each algorithm timed out (LDS).

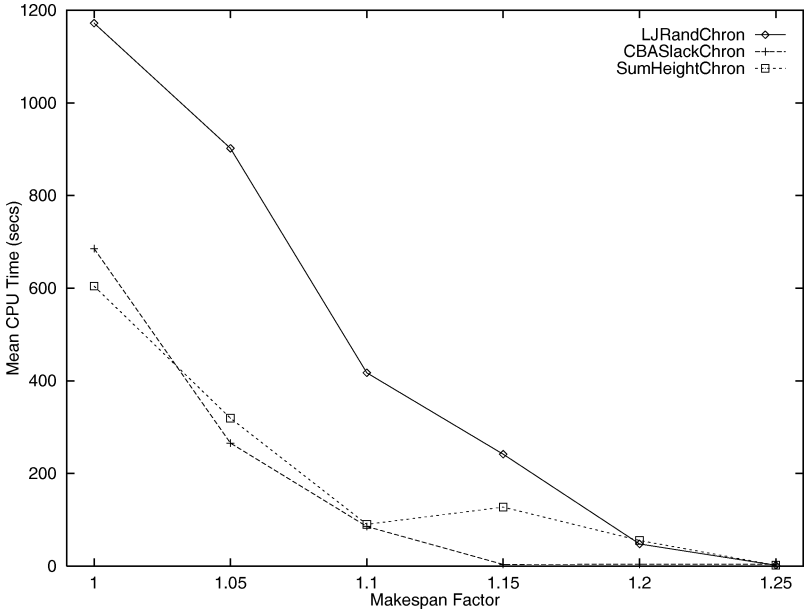

Fig. 8. The mean CPU time in seconds for each problem set (chronological backtracking).

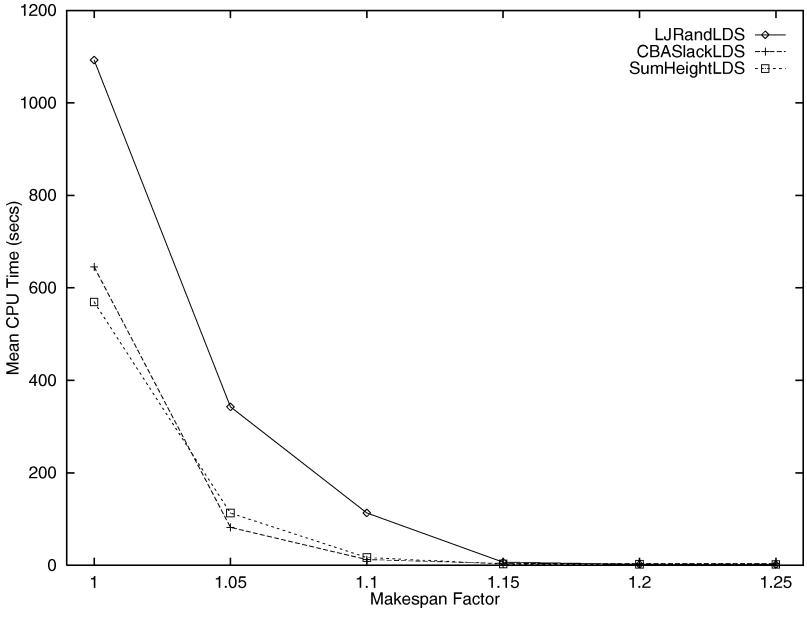

Fig. 9. The mean CPU time in seconds for each problem set (LDS).

Other measurements of search effort (i.e., mean number of backtracks, mean number of commitments, mean number of heuristic commitments) parallel the CPU time results: no significant differences between SumHeight and CBASlack while both are superior to LJRand. Comparison of the retraction techniques on the other statistics also follow the CPU time results: all algorithms using LDS significantly outperform the corresponding algorithm using chronological backtracking and the magnitude of the improvement appears to be larger for LJRand than for the other two heuristics.

6.3. Summary

The basic results from Experiment 1 are that:

- SumHeight and CBASlack are both superior to LJRand while there are no significant differences when SumHeight is compared directly with CBASlack.
- LDS is superior to chronological retraction for all heuristics tested.

7. Experiment 2: Scaling with problem size

While Experiment 1 used a set of hard instances of job shop scheduling problems, there was no effort to evaluate the performance of the algorithms as the size of the scheduling problems increases. In addition, the use of a variety of problem sets of varying difficulty and characteristics should aid in the differentiation of algorithms [12]. The primary purpose of the second experiment, then, is to evaluate the scaling behavior of the algorithms.

7.1. Problem set

Four sizes of problems were selected (5×5, 10×10, 15×15, and 20×20) and 20 problems of each size were generated with the Taillard random job shop problem generator [74].[6] For each problem, a lower bound on the makespan was calculated by running the propagators used for each algorithm. The lower bound is the smallest makespan that the propagators, on their own, could not show was over-constrained. This lower bound calculation is due to [58].

Once the lower bounds were calculated, we applied makespan factors from 1.0 to 1.3 in steps of 0.05. For this problem set, however, we only know a lower bound, not the optimal makespan as above. Therefore, the makespan factor is a multiplier of that lower bound, not of the optimal, and we expect most of the problems to be over-constrained at a makespan factor of 1.0. For each size of problem, we have 120 problems divided into six equal sets based on the makespan factor.

7.2. Results

Figs. 10 and 11 display the fraction of problems in each problem set that timed out. Statistically, CBASlack times out on significantly ($p \leqslant 0.005$) fewer problems than

[6] The duration of each activity is randomly selected, with uniform probability from the domain [1, 100]. Further details, sufficient to generate similar problem sets (varying only based on the random seed) can be found in [74].

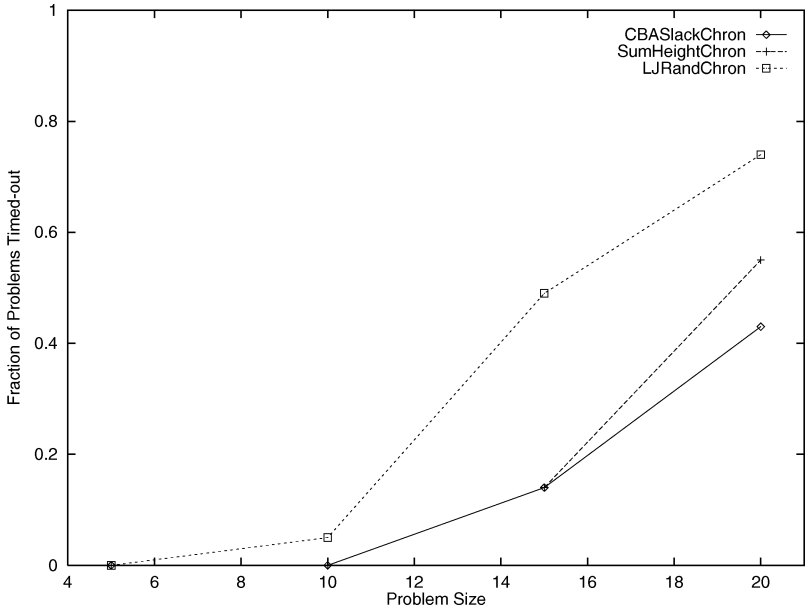

Fig. 10. The fraction of problems in each problem set for which each algorithm timed out (chronological backtracking).

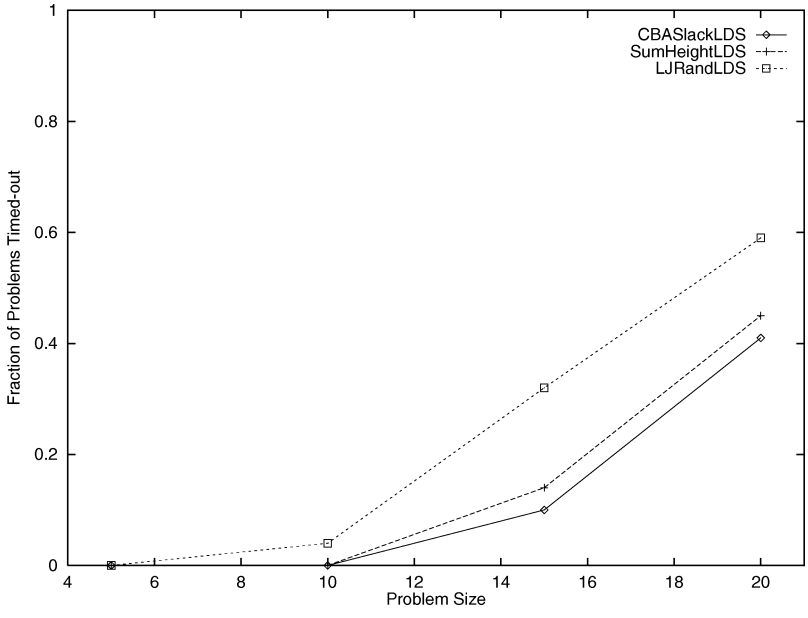

Fig. 11. The fraction of problems in each problem set for which each algorithm timed out (LDS).

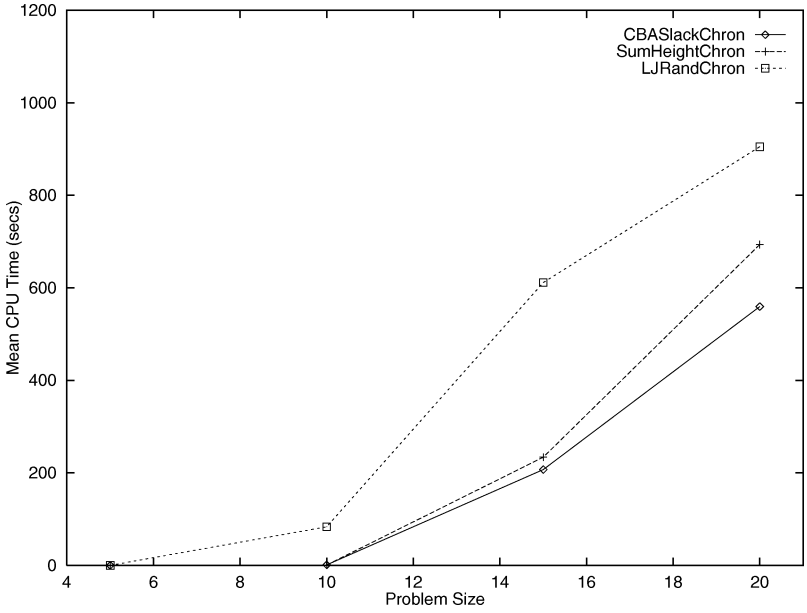

Fig. 12. The mean CPU time in seconds for each problem set (chronological backtracking).

SumHeight, and SumHeight in turn significantly outperforms LJRand under both the chronological backtracking and LDS conditions.

Differences between the retraction techniques are similar to the results for Experiment 1. Overall, LDS times out on significantly fewer problems than chronological retraction when LJRand is used as the heuristic. The overall differences between retraction techniques when using SumHeight or CBASlack as the heuristic are not significant, and, in fact, there are no significant differences between chronological backtracking and LDS on any problem set when CBASlack is used. These results confirm the observation from Experiment 1 that LDS appears to improve the weaker heuristics (based on their performance with chronological backtracking) more than the stronger ones.

The mean CPU time results are shown in Fig. 12 for chronological backtracking and Fig. 13 for LDS. These graphs are quite similar to the results for the percentage of problems timed out as the CPU time on the unsolved problems tends to dominate the mean results. Therefore, the overall CPU results tend to mirror the timed-out results: CBASlack achieves significantly lower mean CPU time than SumHeight ($p \leqslant 0.001$) which in turn achieves a significantly lower mean CPU time than LJRand. In terms of comparing the results for different problem sizes, CBASlack significantly outperforms SumHeight only on the 5×5 and 20×20 problem sets. There is no significant difference at 15×15 and SumHeight achieves a significantly lower mean CPU time than CBASlack on the 10×10 problems. LJRand is significantly worse than CBASlack for all problem sizes and significantly worse than SumHeight for all sizes except 5×5, where there is no significant difference.

Comparing the retraction techniques, we observe that, overall, LDS with LJRand is significantly better than chronological backtracking with LJRand. The overall differences

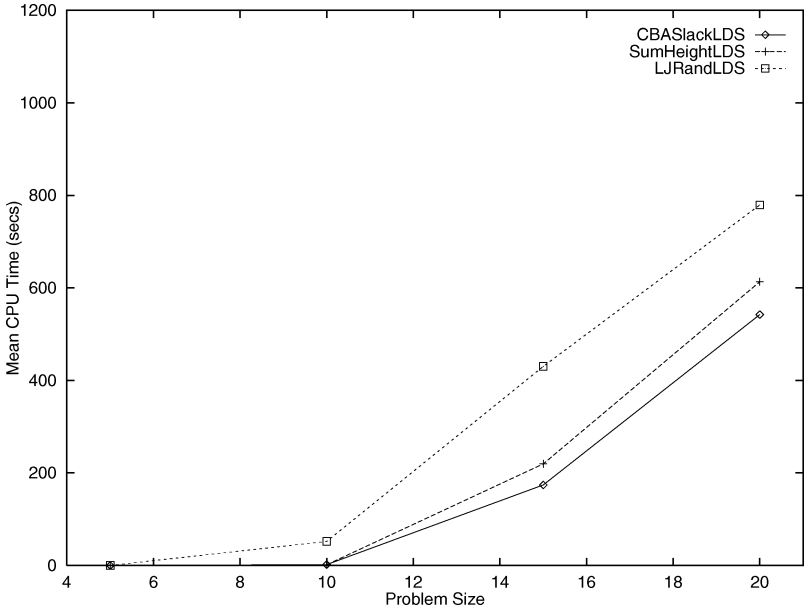

Fig. 13. The mean CPU time in seconds for each problem set (LDS).

with the other two heuristics are not significant, though they are in favor of LDS over chronological backtracking. Interestingly, on the 10×10 problems both CBASlack and SumHeight use significantly less CPU time ($p \leqslant 0.005$) when using chronological backtracking than with LDS. As well, on the 5×5 problems LJRand with chronological backtracking significantly outperforms LJRand with LDS. On the two larger problem sizes, LDS significantly outperforms chronological retraction with LJRand while there is no significant difference with the other two heuristics.

With other search statistics, CBASlack makes significantly fewer commitments, significantly more heuristic commitments, and about the same number of backtracks as SumHeight with both retraction techniques. Both CBASlack and SumHeight significantly outperform LJRand on these statistics. Comparing LDS and chronological backtracking, we see significantly fewer backtracks for LDS, and significantly more commitments and heuristic commitments for LDS (regardless of heuristic used) than for chronological backtracking. As with our other experiments, it appears that the magnitude of the improvement in using LDS rather than chronological backtracking is greater with heuristics that perform worse with chronological backtracking.

7.3. Summary

The results of Experiment 2 show that:

- CBASlack significantly outperforms SumHeight which in turn significantly outperforms LJRand.
- LDS significantly outperforms chronological backtracking.

The primary difference between these results and those of Experiment 1 is that we now see differences between CBASlack and SumHeight. These differences, though significant in a number of cases, are not as clear cut as the differences between LJRand and the other two heuristics. On almost every problem set and every performance heuristic, LJRand is significantly worse than SumHeight and CBASlack.

8. Experiment 3: Bottleneck resources

8.1. Problem sets

The goal in this experiment is to create problems with specific bottleneck resources, and evaluate the performance of the heuristic commitment techniques and retraction techniques as the number of bottlenecks in a problem increase.

Our starting point is the 10×10, 15×15, and 20×20 problems, all with makespan factor 1.2, used in Experiment 2. Recall that the makespan factor is the factor by which a lower bound on the makespan of each problem is multiplied to generate the makespan of the problem instance. The makespan factor was chosen so that many, and perhaps all, of the problem instances in the starting set are not over-constrained. This final choice was made because by adding bottlenecks we are further constraining the problem instances and we do not want the problems without bottlenecks to already be over-constrained.

The bottlenecks were generated for each problem and each problem set independently by examining the base problem and randomly selecting the bottleneck resources. On each bottleneck resource, we then inserted five new activities of approximately equal duration to reduce the slack time on that resource to 0. The five new activities (on each bottleneck resource) are completely ordered by precedence constraints.

For example, for problem A with one bottleneck we may select resource R_2 as the bottleneck. In the base problem the overall makespan is, perhaps, 200 and the sum of the durations of all activities on R_2 is 120. We insert five new activities, each of which executes on R_2 and each with a duration of 16 time units. The sum of the duration of activities on R_2 is 200, the same as the makespan. This has the effect of reducing the slack time on R_2 to 0. To continue the example, when generating the problem set with three bottlenecks, we take problem A and randomly select three resources to be bottlenecks. These resources may be R_5, R_8, and R_{10}.[7] For each of these resources five completely ordered activities are added to reduce the slack time on the resource to 0.

With this technique, problem sets are generated for each problem size. For the 10×10 problems, six problem sets, each containing 20 problems, were generated with the number of bottlenecks ranging from 0 to 10 inclusive, in steps of 2. For the 15×15 problems seven problem sets of 20 problems each were generated with the number of bottlenecks ranging from 2 to 14 inclusive, in steps of 2. Finally for the 20×20 problems, six problem sets of 20 problems each were generated with the number of bottlenecks ranging from 0 to 20 inclusive, in steps of 4.

[7] R_2 may be selected again as a bottleneck. However, R_2 is no more likely to be selected than any other resource.

8.2. Results

8.2.1. 10×10 problems

For the 10×10 problems, the fraction of the problems that each algorithm timed out on are displayed in Fig. 14 for the algorithms using chronological backtracking and in Fig. 15 for those algorithms using LDS. Slightly obscured by the plot, in Fig. 14, is the fact that SumHeight does not time out on any problems. Regardless of the retraction technique used, statistical analysis indicates that SumHeight times out on significantly fewer problems than CBASlack ($p \leqslant 0.005$ for the comparison using chronological backtracking) and LJRand, while CBASlack in turn times out on significantly fewer problems than LJRand.

The mean CPU time results are shown in Fig. 16 (chronological backtracking) and Fig. 17 (LDS). These results are consistent with the timed-out results, as, regardless of retraction technique SumHeight incurs significantly less mean CPU time than either CBASlack or LJRand while CBASlack incurs significantly less mean CPU time than LJRand.

Comparison of the retraction techniques show that there is no significant difference in terms of the number of problems timed out between the algorithms using chronological backtracking and those using LDS. However, both SumHeight and CBASlack incur a lower mean CPU time when used with chronological backtracking rather than LDS.

Results with other search statistics are as follows:

- For the algorithms using chronological backtracking, all statistics tested (the number of backtracks, the total number of commitments, and the number of heuristic

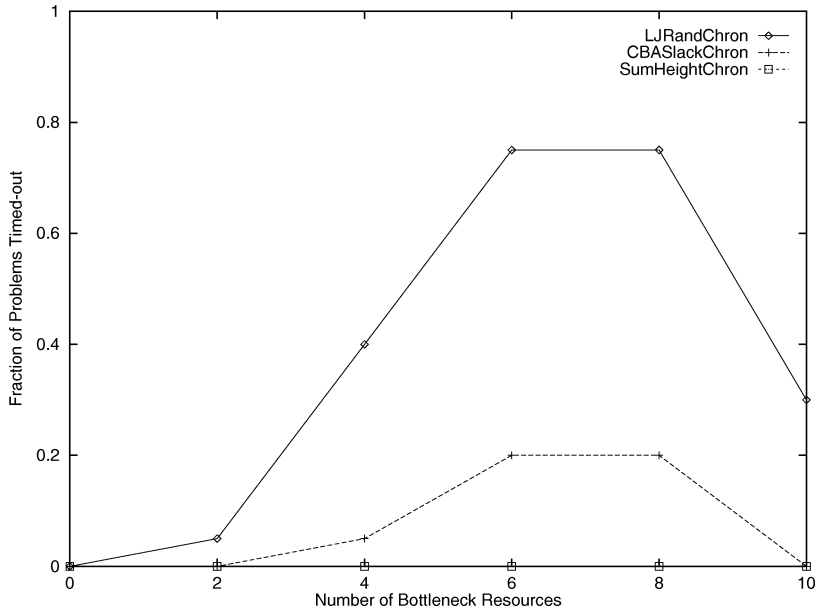

Fig. 14. The fraction of problems in each problem set for which each algorithm timed out (10×10 problems—chronological backtracking).

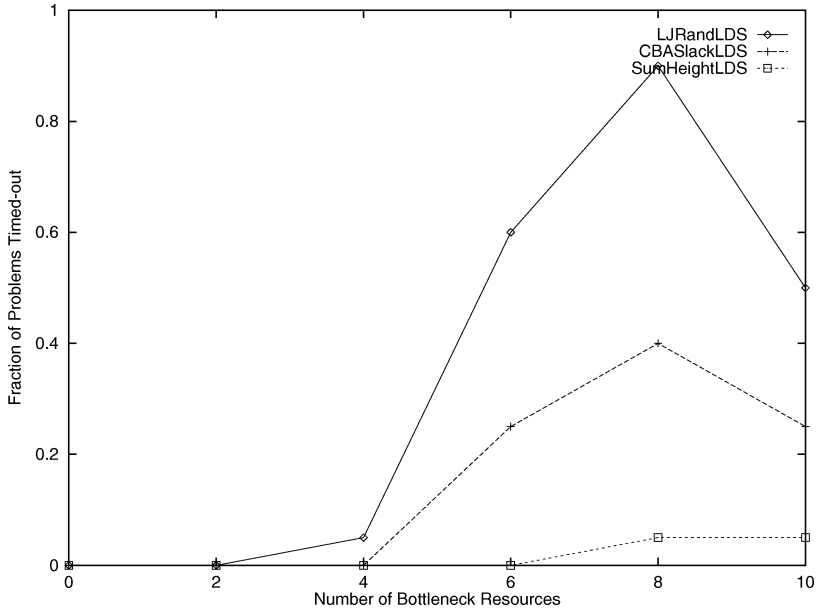

Fig. 15. The fraction of problems in each problem set for which each algorithm timed out (10 × 10 problems—LDS).

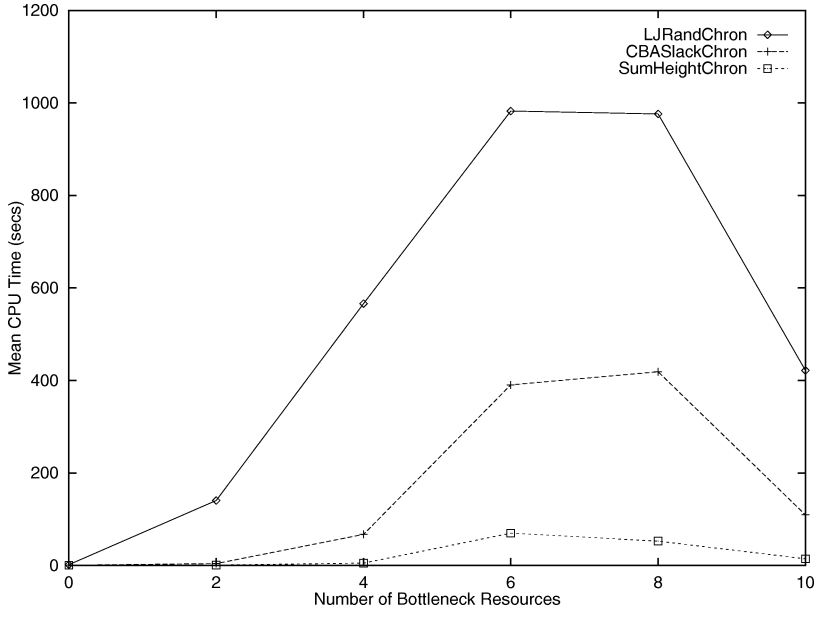

Fig. 16. The mean CPU time in seconds for each problem set (10 × 10 problems—chronological backtracking).

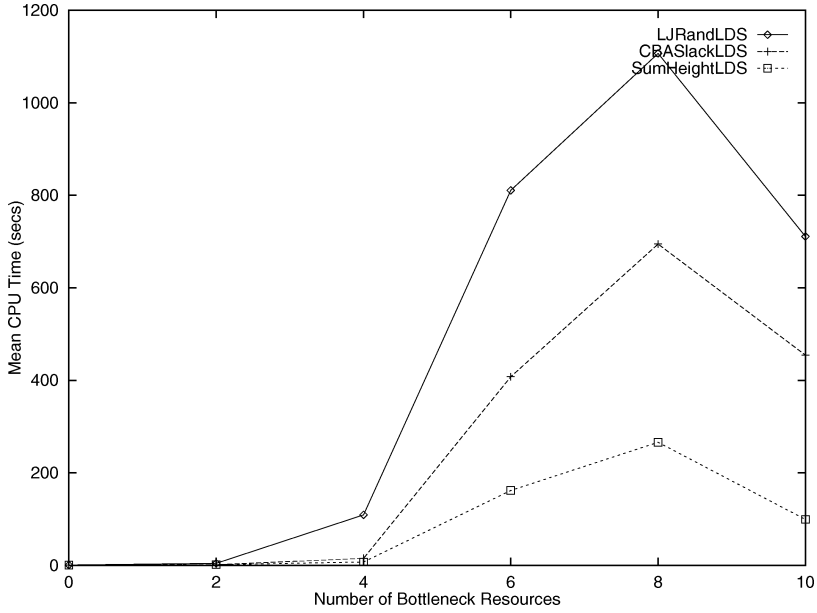

Fig. 17. The mean CPU time in seconds for each problem set (10 × 10 problems—LDS).

commitments) indicate that SumHeight significantly outperforms CBASlack and LJRand. CBASlack in turn significantly outperforms LJRand. These results are repeated when LDS is used as the retraction technique except in the case of the number of backtracks. For that statistic, while SumHeight is significantly better than all other heuristics, there is no significant difference between CBASlack and LJRand.

- Comparison of the retraction techniques shows that, for the number of backtracks, SumHeight incurs significantly fewer when used with chronological backtracking than with LDS, CBASlack shows no significant differences, and LJRand incurs significantly fewer backtracks with LDS than with chronological backtracking. For both the overall number of commitments and the number of heuristic commitments, SumHeight and CBASlack make significantly fewer with chronological backtracking while there is no significant difference for LJRand.

8.2.2. 15×15 *problems*

The fraction of problems in each problem set for which each algorithm was unable to find a solution (or show that the problem was over-constrained) is shown in Fig. 18 for those algorithms using chronological backtracking and in Fig. 19 for those algorithms using LDS.

Statistical analysis mirrors the informal impression of these graphs: SumHeight times out on significantly fewer problems than CBASlack which, in turn, times out on significantly fewer problems than LJRand. These results hold in both retraction component conditions.

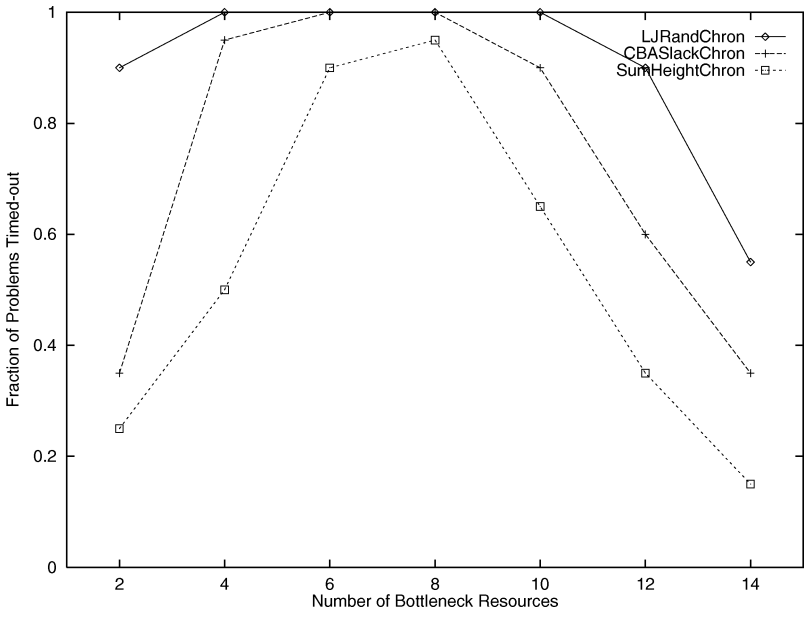

Fig. 18. The fraction of problems in each problem set for which each algorithm timed out (15 × 15 problems—chronological backtracking).

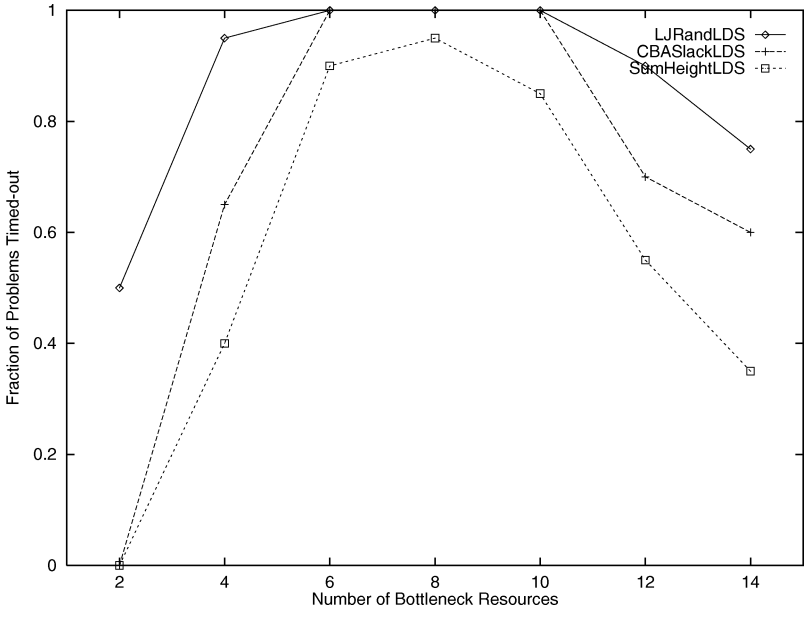

Fig. 19. The fraction of problems in each problem set for which each algorithm timed out (15 × 15 problems—LDS).

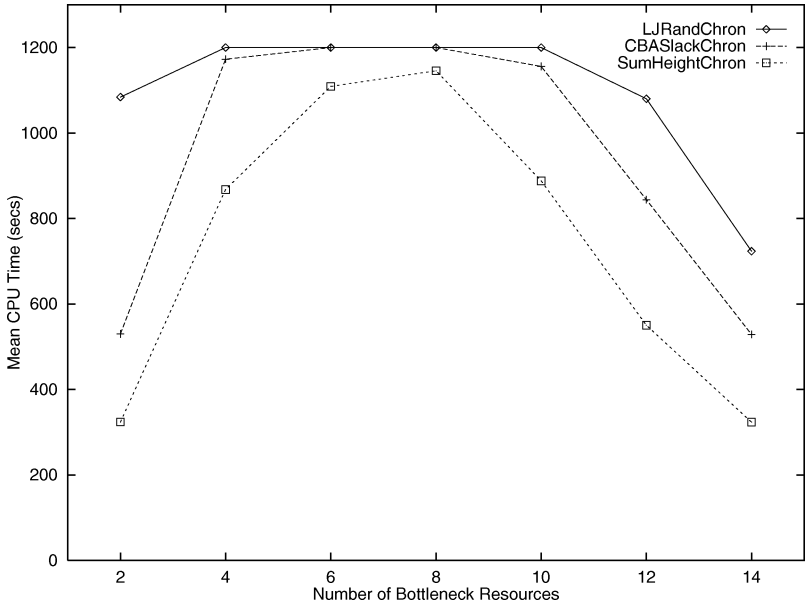

Fig. 20. The mean CPU time in seconds for each problem set (15×15 problems—chronological backtracking).

Comparing the retraction techniques while holding the heuristic commitment technique constant shows no overall statistically significant differences between chronological backtracking and LDS. For the problems sets with a low number of bottlenecks (two and four), we see that LDS solves significantly more problems than chronological backtracking for each heuristic (with the exception of SumHeight on the problems with four bottlenecks where there is no significant difference). For problem sets with higher number of bottlenecks (e.g., 12 and 14) we see the reverse: the algorithms with chronological backtracking time out on fewer problems; however, these differences are not statistically significant.

The results for the mean CPU time for each problem set are displayed in Fig. 20 (chronological backtracking) and Fig. 21 (LDS).

These results reflect the timed-out results: with either retraction technique SumHeight incurs significantly less mean CPU time than CBASlack which in turn incurs significantly less mean CPU time than LJRand.

Comparison of retraction techniques shows no overall significant differences in mean CPU time. Again, however, the pattern of chronological retraction being inferior at low bottlenecks and superior at high bottlenecks is observed. Chronological backtracking with both CBASlack and LJRand on the problem set with two bottlenecks incurs significantly more CPU time than the corresponding algorithms using LDS. For problem sets with 12 and 14 bottlenecks, chronological backtracking with both SumHeight and CBASlack incurs significantly less CPU time than the corresponding algorithm with LDS.

Results with other search statistics are as follows:

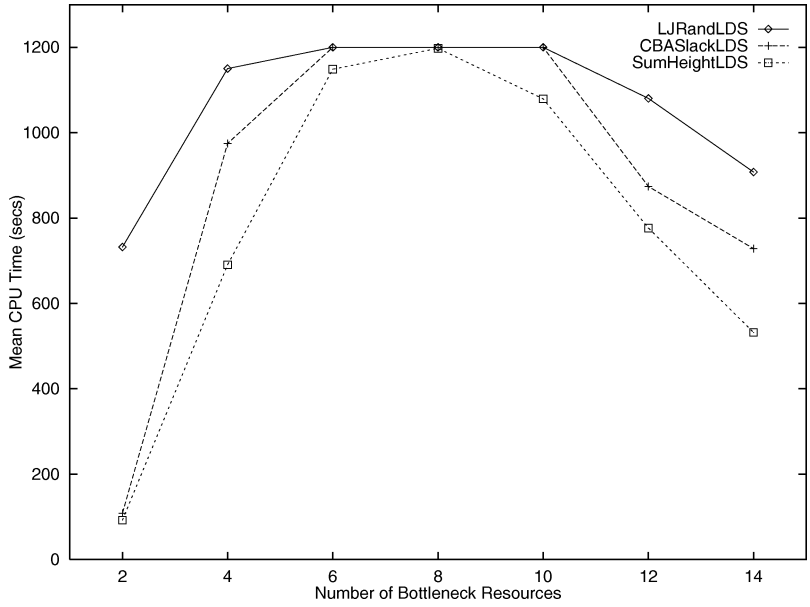

Fig. 21. The mean CPU time in seconds for each problem set (15×15 problems—LDS).

- With chronological backtracking, SumHeight makes significantly fewer backtracks than either LJRand or CBASlack. There is no significant difference between CBASlack and LJRand. With LDS, SumHeight uses significantly more backtracks than CBASlack; however, LJRand incurs fewer backtracks than both SumHeight ($p \leqslant 0.005$) and CBASlack.
- In terms of the number of commitments, there is no significant difference between SumHeight and CBASlack (in either retraction condition) while LJRand makes significantly more commitments than the other algorithms, regardless of retraction technique.
- SumHeight makes significantly fewer heuristic commitments than either of the other algorithms with both LDS and chronological backtracking. CBASlack makes significantly more heuristic commitments than LJRand using LDS.
- Finally, in the comparison of the retraction techniques we see that the algorithms using chronological backtracking make significantly more backtracks, significantly fewer commitments, and significantly fewer heuristic commitments than the corresponding algorithms using LDS. These results hold regardless of the heuristic.

It may seem inconsistent that SumHeight makes fewer backtracks and heuristic commitments than CBASlack but not significantly fewer overall commitments. This anomaly can be understood based on the fact that, for SumHeight, the percentage of the total commitments that are heuristic commitments is significantly smaller than for CBASlack. While CBASlack backtracks more and makes more heuristic commitments, because it makes fewer propagated commitments for each heuristic commitment, the overall number of commitments is not significantly different from that of SumHeight. Given the differing com-

putational expense of heuristic commitments, propagated commitments, and backtracking, the existence of such a comparison is one reason we use mean CPU time as one of our primary search statistics.

8.2.3. 20×20 *problems*

The fraction of problems timed out for each algorithm on the 20×20 problems are shown in Figs. 22 and 23 for chronological backtracking and LDS respectively. Regardless of the retraction condition, statistical analysis indicates no significant difference between SumHeight and CBASlack while both time out on significantly fewer problems than LJRand.

The mean CPU time results are shown in Fig. 24 for chronological backtracking and Fig. 25 for LDS. As with the timed-out results, the mean CPU time results indicate that there is no significant differences between SumHeight and CBASlack, regardless of retraction technique, while both SumHeight and CBASlack incur significantly less mean CPU time than LJRand, again, regardless of retraction technique.

In comparing the retraction techniques themselves, we see that there are no significant differences between the chronological backtracking algorithms and the LDS algorithms in terms of the fraction of problems timed out or the mean CPU time. This result holds for each heuristic commitment technique.

Results with other search statistics are as follows:

- In terms of the number of backtracks, there are no significant differences among the heuristics except when LDS is used as the retraction technique. In that condition,

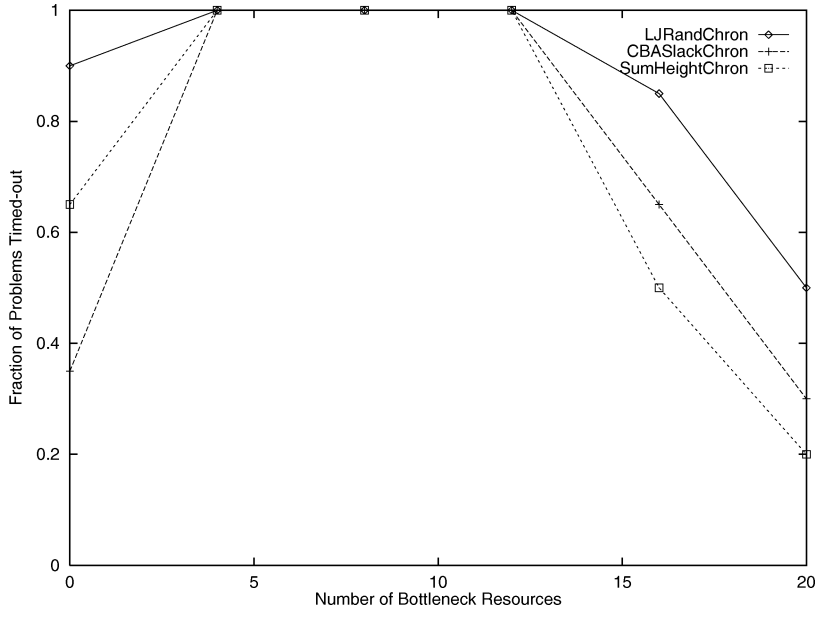

Fig. 22. The fraction of problems in each problem set for which each algorithm timed out (20×20 problems—chronological backtracking).

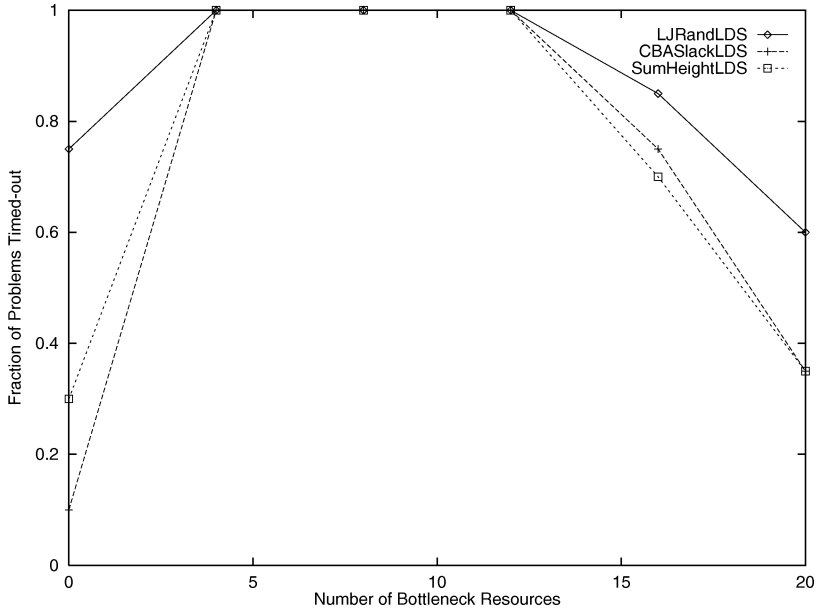

Fig. 23. The fraction of problems in each problem set for which each algorithm timed out (20 × 20 problems—LDS).

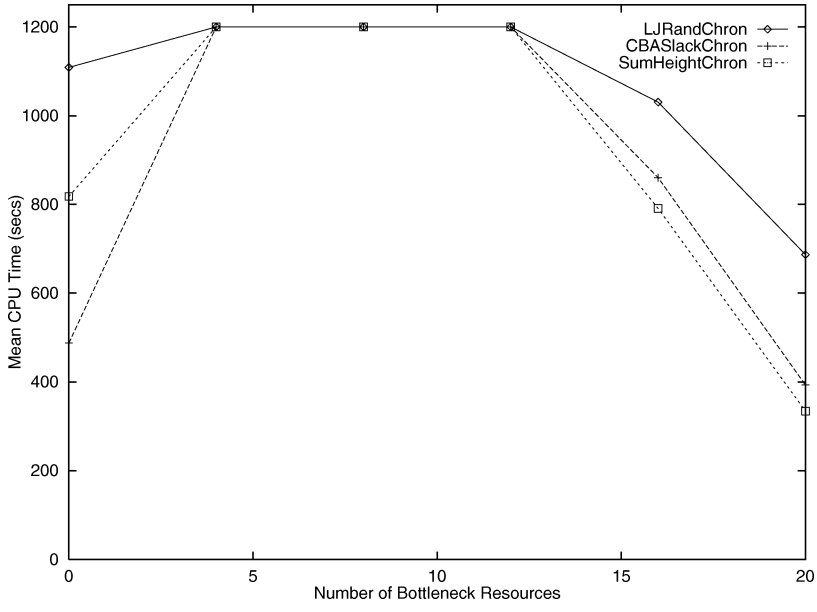

Fig. 24. The mean CPU time in seconds for each problem set (20 × 20 problems—chronological backtracking).

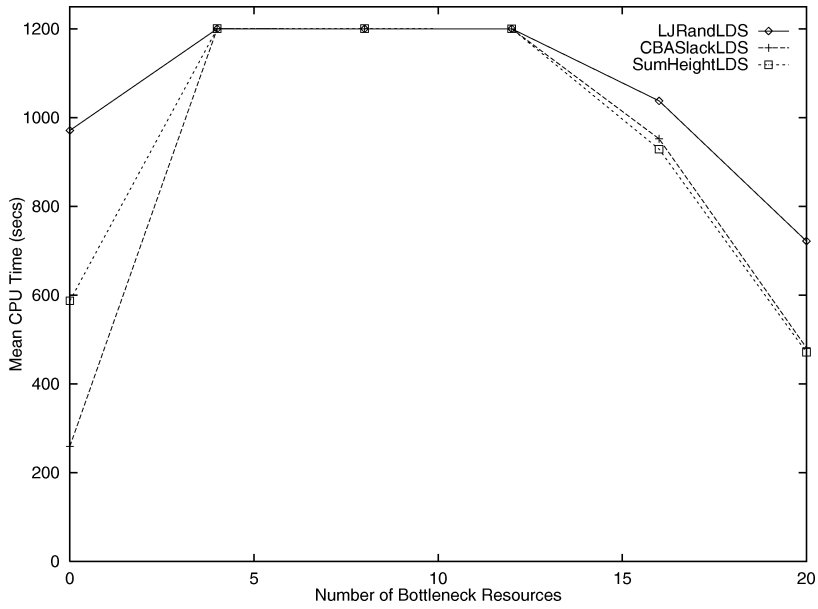

Fig. 25. The mean CPU time in seconds for each problem set (20 × 20 problems—LDS).

SumHeight and CBASlack both incur significantly more backtracks than LJRand while there is no significant difference between SumHeight and CBASlack.

- Regardless of the retraction technique used, CBASlack makes significantly fewer overall commitments than SumHeight or LJRand while SumHeight makes significantly fewer than LJRand.
- For the number of heuristic commitments, SumHeight makes significantly fewer than all other heuristics using either chronological backtracking or LDS. Similarly, LJRand makes significantly fewer heuristic commitments than CBASlack regardless of retraction technique used.
- Finally, in comparing the retraction techniques, we see that algorithms using chronological backtracking make significantly more backtracks, significantly fewer commitments, and significantly fewer heuristic commitments than corresponding algorithms using LDS.

8.3. Summary

The results from Experiment 3 indicate that:

- With the exception of the 20 × 20 problems, SumHeight outperforms CBASlack and LJRand while CBASlack in turn outperforms LJRand. On the 20 × 20 problems there was little difference between SumHeight and CBASlack while both outperformed LJRand.
- The comparison of chronological backtracking with LDS is less clear. With the exception of the 10 × 10 problems, there is no difference in terms of CPU time

or the fraction of problems timed out. On the 10×10 problems, chronological backtracking is able to achieve a lower mean CPU time when used with SumHeight or CBASlack. The other statistics show that, typically, chronological backtracking incurs more backtracks than LDS but fewer overall commitments and fewer heuristic commitments. An exception is observed in the 10×10 results where SumHeight with chronological backtracking incurs significantly fewer backtracks than SumHeight with LDS.

9. Discussion

9.1. Heuristic commitment techniques

The clearest result from the experiments is that LJRand does not perform as well as either of the other two heuristics. This comparison holds regardless of the retraction technique and across all the experiments. This result is in conflict with that of Nuijten [58] where it was shown that LJRand was able to outperform the ORR/FSS heuristic on the Operations Research library problems (a subset of which were used in Experiment 1). As noted above, however, for the experiments performed by Nuijten, LJRand was run with a different retraction technique (chronological backtracking with restart) than the ORR/FSS heuristic was (chronological retraction). Therefore, in light of our work, Nuijten's results can be attributed to the differing retraction techniques used in the experiments rather than as a result of the differing heuristic commitment techniques. [8]

The comparison between SumHeight and CBASlack is more complicated as we see no significant difference in Experiment 1, while CBASlack performs better in Experiment 2 and SumHeight performs better in Experiment 3.

Recall (Section 4) that SumHeight and CBASlack are different methods of measuring similar underlying characteristics of a constraint graph: the contention for a resource. Where SumHeight measures this contention by aggregating probabilistic demand of each activity, CBASlack identifies the pair of activities that compete most with each other. The results of our experiments can be understood based on the relative sensitivity of the two heuristics to non-uniformity at the resource level. To understand this concept, it is helpful to examine the properties of our problem sets. In Experiment 2, the duration of each activity and the sequence of resources used by each job were randomly generated. There is little difference, therefore, in resource utilization: these problems have a relatively uniform resource utilization. In Experiment 3, we augmented the randomly generated problems by selecting a subset of resources and adding activities such that their utilization was 100%. A scheduling problem with a wide ranging resource utilization is said to have a non-uniform resource utilization.

Imagine two scheduling problems, P_{uniform} and $P_{\text{non-uniform}}$, with a uniform and a non-uniform resource utilization respectively. Assume that $P_{\text{non-uniform}}$ was generated from

[8] The ORR/FSS heuristic does not have a random component; therefore, it cannot be directly used with a retraction technique like chronological backtracking with restart which depends on such a component to explore different paths in the search tree. A number of researchers [19,41,60] have investigated ways of adding a random component to otherwise deterministic heuristic commitment techniques.

P_{uniform} by the method used to generate bottleneck problems. $P_{\text{non-uniform}}$ has a superset of the activities in P_{uniform}. The CBASlack calculations (before any commitments have been made) on P_{uniform} find the biased-slack for each pair of activities. The calculation depends wholly upon the time-windows of each activity which in turn depend on the precedence constraints within a job: there are, as yet, no inter-job constraints. When the same calculations are done on $P_{\text{non-uniform}}$ the pairs of activities that the problems have in common will have exactly the same biased-slack values. CBASlack only estimates the contention between pairs of activities and so it will calculate the same biased-slack value for a pair of activities regardless of the other activities contending for the same resource. Therefore, the biased-slack value of two activities in P_{uniform} is the same as in $P_{\text{non-uniform}}$. Gradually, as commitments are made, the activities on a bottleneck resource will tend to have lower biased-slack values; however, for the first few commitments, CBASlack does not detect the non-uniformity. As has been noted elsewhere [43], the first few commitments are often critical to finding a solution. SumHeight, in contrast, immediately focuses on one of the bottlenecks as it aggregates probabilistic demand information from all activities on the resource. Assuming that it is truly more important to make decisions on tighter resources, the ability to focus on bottleneck resources, *when they exist*, is an explanation of why SumHeight is able to outperform CBASlack on non-uniform problems as in Experiment 3.

When problems have a uniform resource utilization, no bottleneck resources exist. SumHeight identifies a critical resource and time point; however, it is likely that other resources and time points have criticality measurements that are close (or even equal) to the one selected. Only the activities on the "critical" resource are then examined in order to post a commitment. In contrast, CBASlack looks at all pairs of activities. It may be the case that though the resource utilization is uniform, there are pairs of activities that have a very low biased-slack value. For example, imagine a pair of activities such as *A* and *B* in Fig. 26. The wide time windows on each activity lead to a relatively low contention as measured by SumHeight, but CBASlack will identify these activities as critical. While SumHeight may identify some other resource as critical and make a commitment, CBASlack will focus on a critical pair regardless of the resource utilization and, therefore, it is likely to make a commitment at a more critical area of the search space than SumHeight.

An underlying assumption in this explanation is that when a resource-level non-uniformity exists, it is more important to make a commitment on activities that execute on a highly contended-for resource even though there may be an activity pair with a lower biased-slack value on another resource. While we believe this assumption to be reasonable, at this time we have no empirical evidence to support it (without begging the question we are trying to explain).

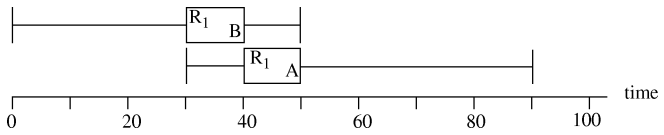

Fig. 26. Activities *A* and *B*.

To lend support to this explanation, we examined the resource uniformity of the problems in Experiments 2 and 3. The resource usage of a resource, R, is the fraction of the total scheduling horizon during which a resource is used by some activity. We represent the resource usage by $RU(R)$ which can be found by summing the durations of the activities on R and dividing by the length of the scheduling horizon. Given $RU(R)$ for each resource in a problem, $RU(P)$ is defined to be the mean resource usage across all resource in problem, P. Assuming that $RES(P)$ denotes the set of resources in problem P, the standard deviation for the resource usage in a problem, $\sigma(RU(P))$, can be calculated as shown in Eq. (9).

$$\sigma\big(RU(P)\big) = \sqrt{\frac{\sum_{R \in RES(P)} (RU(R) - RU(P))^2}{|RES(P)| - 1}}. \tag{9}$$

The standard deviation of resource usage in a problem is a measure of its resource non-uniformity. The higher the standard deviation, the more varied is the resource usage in the problem.

Fig. 27 plots the difference in CPU time between SumHeight and CBASlack (both using chronological backtracking) against the standard deviation of resource usage in each problem. Points above zero on the y-axis indicate problems where SumHeight incurred greater CPU time than CBASlack, while those below zero indicate problems

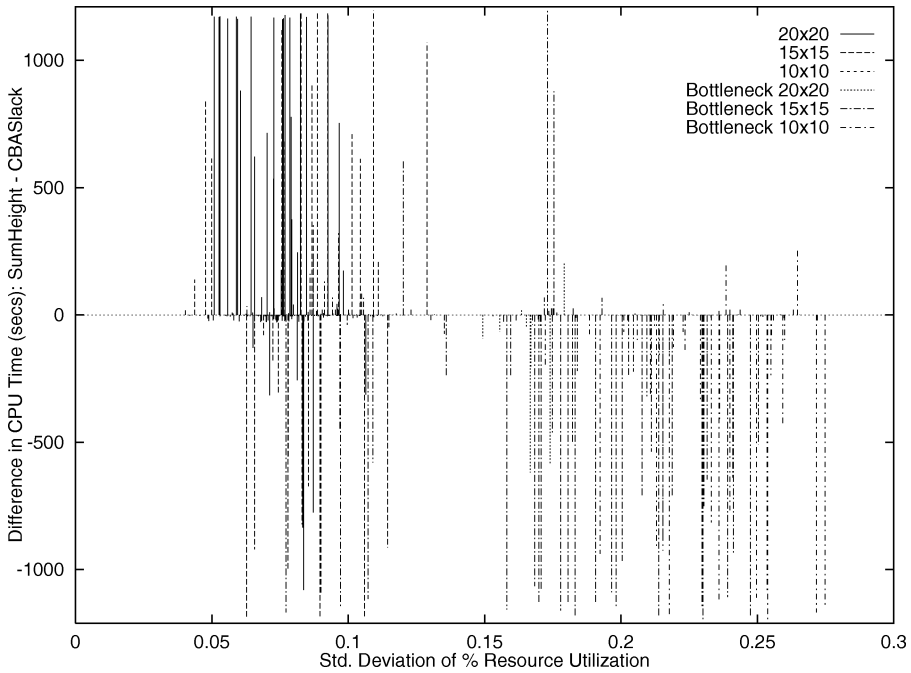

Fig. 27. The standard deviation in resource usage for each problem in the 10×10, 15×15, and 20×20 problems sets of Experiment 2 and Experiment 3 versus the difference in CPU time in seconds between SumHeight and CBASlack (chronological backtracking).

where SumHeight incurred less CPU time. The horizontal axis is the standard deviation of the percent resource utilization for each problem. A larger standard deviation indicates a larger difference in utilization among resources in a problem and therefore a greater resource-level non-uniformity.

Fig. 27 indicates that on problems with more resource-level non-uniformity, SumHeight tended to outperform CBASlack. The reverse is true with problems with more uniform resource usage. These results do not prove that the difference in resource-level uniformity causes the performance difference. The results do, however, indicate a correlation which we see as lending credence to the resource-level uniformity explanation.

9.1.1. Heuristic commitments versus implied commitments

Given the use of both heuristic commitment techniques and propagators in our algorithms, it is instructive to examine the percentage of heuristic commitments with different algorithms. It has been suggested [7] that, given the power of propagators in constraint-directed scheduling, a good heuristic is one that makes decisions that result in many subsequent propagated commitments. The intuition seems to be primarily pragmatic (i.e., to maximize the contributions from sophisticated propagators) however there is some theoretical evidence from SAT heuristics that is consistent with such a suggestion [45]. While further research is necessary to evaluate whether a good heuristic *necessarily* results in more propagated commitments, it is interesting from the perspective of understanding the search behavior to examine the interactions between heuristic commitment techniques and propagators.

The graphs in Figs. 28 and 29 display the percentage of commitments in the search that were heuristic commitments for each algorithm in Experiment 1. Recall that the total set of commitments are composed of heuristic commitments found by the heuristic commitment technique and the implied commitments found by the propagators.[9] The graphs show that LJRand makes a significantly smaller percentage of heuristic commitments than does SumHeight, which in turn makes a significantly smaller proportion of commitments than CBASlack. This holds with both chronological backtracking and LDS.

There are two requirements for such a comparison to be meaningful. First, it is important that the heuristics make the same type of commitment. Clearly, if one heuristic completely sequences a resource in a single heuristic commitment, it will incur a lower percentage of heuristic commitments than a heuristic that posts a sequencing constraint between a pair of activities. The second requirement is similar search performance. A search state for a problem is more constrained if it occurs deep in a search tree (after a number of commitments have been added to the graph) than early in the tree. A chronological backtracking algorithm that is not performing well is likely to spend a great deal of time near the bottom of a search tree where each heuristic commitment results in many implied commitments. In contrast, an algorithm that easily finds a solution spends a relatively small portion of the search at the bottom of the tree and therefore will have a higher percentage of heuristic commitments.

[9] For this statistic we take into account the commitments found by both types of edge-finding and by CBA. We do not count the commitments found by temporal propagation as these commitments are typically implemented by removing values from the domains of start time variables rather than by the explicit assertion of a constraint.

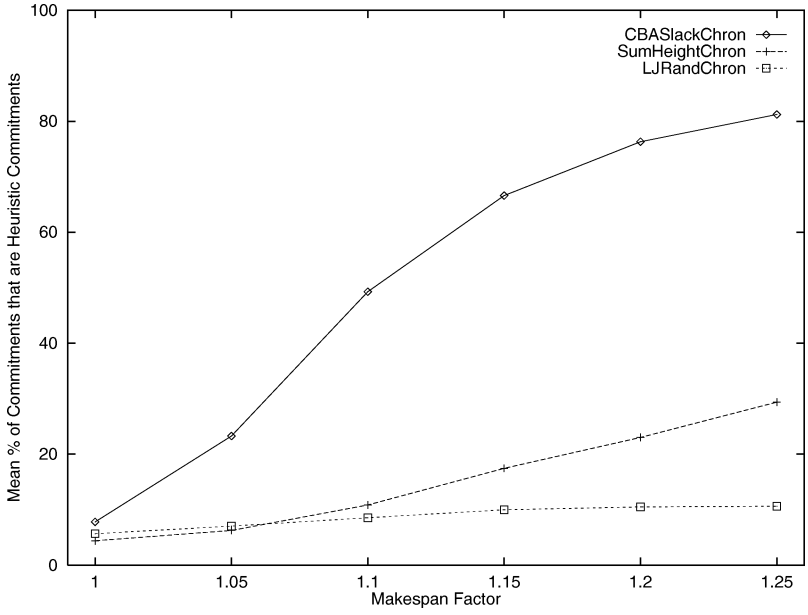

Fig. 28. The mean percentage of commitments made by the heuristic commitment technique (chronological backtracking).

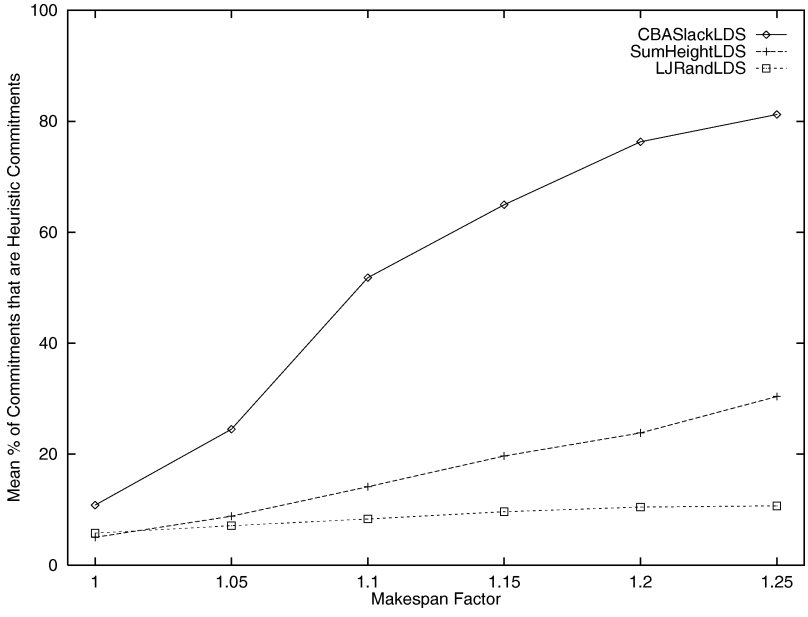

Fig. 29. The mean percentage of commitments made by the heuristic commitment technique (LDS).

LJRand makes a different type of commitment than the other two heuristics. By assigning an activity a start time, LJRand makes a highly constraining commitment from which we might expect a large number of implied commitments to be readily identifiable. This appears to be the case in our experimental results as LJRand demonstrates the lowest percentage of heuristic commitments. Given the difference in heuristic commitment type and the fact that LJRand is significantly worse than the other heuristics on all the problem solving performance measures, it is not very useful to compare the LJRand percentage of heuristic commitments to that of SumHeight and CBASlack. It is interesting, however, to note that the percentage of heuristic commitments for LJRand does not vary much with the makespan factor. This is somewhat surprising: as the makespan factor gets smaller, the problems get tighter and therefore it is expected that the percentage of commitments that are implied should grow.

The comparison of the heuristic commitment percentage between SumHeight and CBASlack is striking. Recall that there were no statistically significant differences in either the number of commitments or the number of heuristic commitments between CBASlack and SumHeight. Nonetheless, SumHeight averaged (non-significantly) more overall commitments and (non-significantly) fewer heuristic commitments than CBASlack. When these results are combined, significant differences arise. These results show that, as compared with SumHeight, a much larger portion of the CBASlack commitments are heuristic even though exactly the same set of propagators were used in the experiments. These results can be understood based on how the heuristics choose a pair of activities to sequence. CBASlack chooses the activity pair with minimum biased-slack regardless of the presence of any other competing activities. After a CBASlack commitment, it may be the case that few other activities are affected and, therefore, there is little propagation. SumHeight, on the other hand, selects a pair of activities precisely because they are part of a larger subset of highly competitive activities. SumHeight explicitly looks for areas where there are a number of activities. It is reasonable, therefore, to expect that a single commitment will, though the propagators, lead to a greater number of implied commitments than CBASlack.

Heuristic commitment percentage and heuristic quality. An important consideration of the difference in heuristic commitment percentage is the meaning, if any, that it has for the quality of the heuristic. Here we present intuitions about the nature of heuristic commitment techniques. These intuitions are speculative and their investigation forms the basis for our future work on heuristics.

We believe that there are two characteristics that correlate highly with the success of a heuristic commitment technique.

(1) The Short Dead-end Proof Intuition: the ability of a heuristic to quickly discover that the search is at a dead-end. Imagine a search that progresses through a series of states to state S. Assume that there are no solutions in the subtree below S, but that we are unable to prove this at S. Some heuristic commitments must be made to explore the subtree below S in order to prove that S is a dead-end.[10] A heuristic that is able, through its heuristic commitments and subsequent propagation, to prove

[10] We are assuming a provable retraction technique like chronological backtracking.

a dead-end while exploring a small number of states in the subtree will be superior to one that must explore a larger subtree.

(2) The Few Mistakes Intuition: a high likelihood of making a commitment that leads to a solution. A heuristic that has a higher likelihood of making a commitment that leads to a solution, that is one that does not result in a dead-end, is likely to be a higher quality heuristic than one with a lower probability of making the correct decision.

These two characteristics are independent from the perspective that if either characteristic were infallible, search performance would be independent of the other characteristic. If a heuristic never made a commitment that resulted in a dead-end state, the ability to detect such a state is irrelevant. Similarly, if a search could immediately detect that it was at a dead-end, then the quality of the commitments is irrelevant: any mistake is immediately detected and repaired.

With real, fallible heuristics, these two characteristics interact: it may be the case that a heuristic with a high probability of making the right decision results in a relatively large effort in discovering when it has made a mistake. Conversely, a heuristic with a lower probability of making a right decision may be able to find a shorter proof of a dead-end (on average) and therefore recover from mistakes quickly. The best heuristic will be the one that minimizes the size of the overall search tree and this may well be achieved by some trade-off between the two characteristics.

Returning to SumHeight and CBASlack, it is our intuition that the lower percentage of heuristic commitments for SumHeight than for CBASlack indicates that SumHeight is able to find shorter proofs of a dead-end. More commitments in the search tree below an unknown dead-end will be implied commitments and therefore the size of the subtree that must be heuristically investigated will tend to be smaller.

In Experiments 1 and 2, this ability to find short proofs (if our intuitions are accurate) does not result in superior performance. A possible explanation for this is a higher probability for CBASlack to make the correct decision. CBASlack identifies an activity pair, such as the one shown in Fig. 26, where the sequence is almost implied by the existing time windows. SumHeight on the other hand selects a pair of activities that may have significantly more overlapping time windows. We speculate, therefore, that in Experiment 2, the "more obvious" commitment made by CBASlack has a higher probability of being in a solution than the commitment made by SumHeight and further, that this higher probability is enough to overcome the shorter-proof characteristics of SumHeight.

In Experiment 3, we believe that the presence of bottleneck resources reduces the probability that CBASlack will make the correct commitment to such a point that the short proof characteristic of SumHeight results in superior overall performance. Imagine two problems, P and $P_{\text{bottleneck}}$, with the only difference between them being that extra activities are added to resource R_1 in $P_{\text{bottleneck}}$ to make it a bottleneck resource. A heuristic commitment in P can have greater impact on the activities of R_1 without resulting in an eventual dead-end than the same commitment would have in $P_{\text{bottleneck}}$. Because CBASlack ignores the presence of a bottleneck, the likelihood that it will make a correct commitment is smaller in $P_{\text{bottleneck}}$, where a bottleneck is present, than in P. We speculate that this

reduction in the likelihood that a commitment will be correct can account for the results of Experiment 3.

While it is reasonable that the presence of a bottleneck enhances SumHeight short proof characteristic, it is not clear what effect it has on the probability that SumHeight will make a correct commitment. The sequencing heuristics are based on the individual demand curves of each activity, which are unchanged whether or not the two activities participate in a bottleneck. Therefore, there does not appear to be a compelling reason to speculate that the presence of a bottleneck changes the correct commitment characteristic.

9.2. Retraction techniques

For Experiments 1 and 2, LDS was clearly superior to chronological backtracking both in terms of the number of problems and mean CPU time. There was little statistically significant overall difference in Experiment 3. One pattern seen in almost all experiments is that LDS significantly outperforms chronological backtracking on the looser problems (e.g., problems with higher makespan factor or fewer bottlenecks) while there is no significant difference on the tighter problems.

In general, these results are consistent with previous work and our expectations. LDS makes larger jumps in the search space than chronological backtracking, undoing a number of commitments in one backtrack. The larger jumps mean that there will often be more effort for each backtrack of LDS than of chronological backtracking. Often, as in Experiments 1 and 2, the extra effort pays off in terms of solving more problems in less time. In Experiment 3, however, the extra effort at each backtrack did not result in better overall performance: the effort incurred by making significantly more commitments (while making fewer backtracks) was not reflected in solving more problems.

LDS significantly outperforms chronological backtracking on the looser problems (e.g., problems with higher makespan factor or fewer bottlenecks) while there is no significant difference on the tighter problems. This pattern can be understood by the presence of over-constrained problems in our problem sets. In an over-constrained problem, an algorithm using LDS will expend more resources proving a problem is over-constrained than the corresponding algorithm using chronological backtracking. This is due to the fact that LDS must revisit some search states in performing a complete search. In our experiments, the problems on which chronological backtracking outperforms LDS can be attributed to the presence of over-constrained problems.

9.2.1. Improving heuristics

In terms of comparing the retraction components, we see that on all the search performance measures, the LDS algorithms tend to outperform their counterparts using chronological backtracking. The size of the improvement when using LDS rather than chronological backtracking is also an interesting statistic. It has been observed [14,51] that when moving from chronological backtracking to LDS, the heuristics that performed worse with chronological backtracking are improved more than the heuristics that performed better: the differences among the heuristics are less with LDS than with chronological backtracking. If we examine the magnitude of improvement, we also observe this trend. Fig. 30 plots the mean difference in CPU time between LDS algorithms and chronological

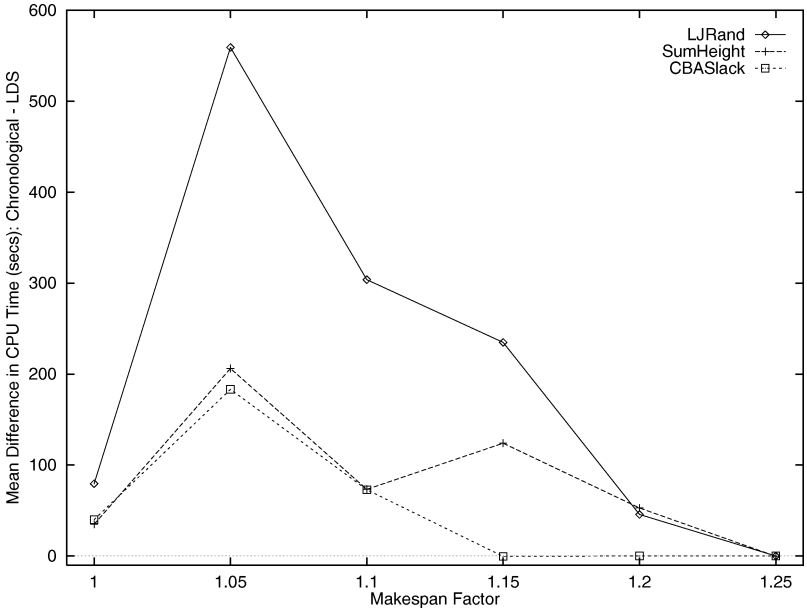

Fig. 30. The mean reduction in CPU time in seconds when using LDS instead of chronological backtracking for each heuristic.

backtracking algorithms for each problem set in Experiment 1. The greatest improvement is seen with LJRand which is improved significantly more ($p \leqslant 0.005$) by using LDS than either SumHeight or CBASlack. No significant difference in the magnitude of the improvement is observed between CBASlack and SumHeight.

While the trend of a greater improvement for weaker heuristics has been observed before, it remains at the level of observation. It may be that some or all of this phenomenon is due to a ceiling effect:[11] because LJRand does so poorly with chronological backtracking there is much more room for improvement when using LDS than is the case for the other, better heuristics. If the effect is real, rather than an artifact of the experimentation, we do not have and are not aware of any explanation for it. It remains an interesting observation, but whether it has any implications for retraction techniques or heuristics remains future work.

It should be noted that despite improving the weaker heuristics more than the stronger ones, LDS did not change the relative heuristic performance. In many cases where there was a significant difference among heuristics, that difference was manifest with both chronological backtracking and LDS. In some cases, a significant difference disappeared; however, we were not able to find results where there was a significant difference between

[11] A ceiling effect is when two techniques are judged to be not significantly different due to the fact that they both perform very well on an experimental problem set. It may be the case that with a more difficult set of problems a significant difference would be detected. In this context, we speculate that a ceiling effect may contribute to the smaller improvement of SumHeight over LJRand when using LDS instead of chronological backtracking.

heuristics with one retraction technique and the opposite significant difference with the other technique.

9.3. Statistical significance and real significance

Our goal in conducting these experiments was to investigate if superior search performance is achieved by heuristic commitment techniques that exploit dynamic problem structure information. Given that we demonstrated statistically superior performance in the presence of resource non-uniformities, the question arises as to the real significance of these results: can we expect similar results to be observed with other types of scheduling problems or even with non-scheduling problems modeled in the constraint-directed paradigm?

For scheduling problems, there are two main dimensions for generalizability. The first dimension is whether the results can be generalized to larger scheduling problems. The experiments using 20×20 job shop problems indicate that there is little difference among the heuristics: none were very likely to solve the problems within the 20 minute CPU time limit. While the experimental design, specifically the use of a time limit, may hide differences among the heuristics, we interpret these results to indicate that, *from a practical perspective*, there is currently little difference among the heuristics on larger job shop problems. The second dimension of generalizability is whether the results can be extended beyond the job shop scheduling model. Given the simplicity of the job shop model, many other scheduling models, and, in particular, real-world scheduling problems, have significantly more complex structure [33]. In scheduling problems involving, for example, alternative activities [16], activities that can consume and produce inventory [10], or non-unary resource capacities [24], there are more problem characteristics that can be dynamically exploited by heuristic commitment techniques. Given the results of Experiment 3, we expect therefore, that the superior results observed for the heuristics that incorporate dynamic analyses of each search state will also be observed in more complicated scheduling domains. [12]

Finally, we expect that the superior performance of heuristics based on dynamic search space analysis will also be observed in non-scheduling problems that are formulated in the constraint-directed search paradigm. As noted in Section 3.1, the contention texture measurement is based on the estimation of the extent the two variables, connected by a disequality constraint compete for the same values. Such constraints are not limited to scheduling problems and, furthermore, there exist extensions of contention can be applied to more general classes of constraints [10,13]. Therefore, we believe that our results can be generalized to other constraint-directed problem models.

10. Conclusions

In this paper, we conducted empirical analysis of three heuristic commitment techniques for constraint-directed scheduling. Our primary goal was to evaluate the efficacy of

[12] Indeed, this expectation, has been borne out by empirical results [10,16,24].

using dynamic search state analysis (in the form of texture measurements) as a basis for heuristic commitment techniques. Results support the use of texture measurements as a basis for heuristic commitment techniques as the two heuristics (SumHeight and CBASlack) based on an analysis of each search state outperformed the less informed heuristic (LJRand) across all experiments. The comparison of SumHeight and CBASlack suggests that an explanation of their relative performance depends upon the presence (or absence) of non-uniformities in problem structure. This result also supports the use of texture-based heuristics as one of the key motivations for texture measurements is to distill non-uniformities in a search state.

We also examined the percentage of the commitments in a search that are created by the heuristic commitment technique as opposed to those created by propagators. Even though SumHeight and CBASlack post the same type of commitment and achieved the same overall search performance (on the specific problem set used), SumHeight makes a significantly smaller percentage of heuristic commitments than CBASlack. These results are used as a basis for a further, though speculative, explanation of the overall search differences between the heuristic search techniques.

Our experiments made use of two retraction techniques (chronological backtracking and Limited Discrepancy Search (LDS)) in two experimental conditions. In general, the performance of the heuristics was similar in the two conditions. In some cases significant differences with one retraction technique were not observed with the other, however it was never the case that a significant difference with one retraction technique was replaced by the opposite significant difference with the other. This demonstrates the robustness of our comparison of heuristic commitment techniques. Overall, algorithms using LDS achieved superior search performance when compared to those using chronological backtracking. This is consistent with previous work on such a comparison, however, this is the first work of which we are aware that compares LDS with chronological backtracking in the presence of state-of-the-art propagation and heuristic techniques. In addition, it was observed that while LDS tends to improve performance when used in place of chronological backtracking, it tends to improve the weaker algorithms more than the stronger. The performance gain by the LJRand algorithm in using LDS rather than chronological backtracking was significantly greater than the improvement of the other two algorithms.

Overall, this paper supports the thesis that an understanding of the structure of a problem leads to high-quality heuristic problem solving performance. In the presence of modern propagation and retraction techniques, it was demonstrated that heuristic commitment techniques based on the dynamic analysis of the constraint graph representation of each search state achieve better overall heuristic search performance in terms of fewer commitments, lower mean CPU time, and more problem instances solved.

Acknowledgements

This research was performed while the first author was a Ph.D. student at the Department of Computer Science, University of Toronto. It was funded in part by the Natural Sciences Engineering and Research Council, IRIS Research Network, Manufacturing Research Corporation of Ontario, Baan Limited, and Digital Equipment of Canada. Thanks to

Andrew Davenport, Angela Glover, Edward Sitarski, and Ioan Popescu for discussion of and comments on previous versions of this paper.

References

[1] J. Adams, E. Balas, D. Zawack, The shifting bottleneck procedure for job shop scheduling, Management Science 34 (1988) 391–401.
[2] J.F. Allen, Maintaining knowledge about temporal intervals, Comm. ACM 26 (11) (1983) 832–843.
[3] D. Applegate, W. Cook, A computational study of the job-shop scheduling problem, ORSA J. Comput. 3 (1991) 149–156.
[4] K. Baker, Introduction to Sequencing and Scheduling, Wiley, New York, 1974.
[5] P. Baptiste, C. Le Pape, Disjunctive constraints for manufacturing scheduling: Principles and extensions, in: Proc. 3rd International Conference on Computer Integrated Manufacturing, 1995.
[6] P. Baptiste, C. Le Pape, Edge-finding constraint propagation algorithms for disjunctive and cumulative scheduling, in: Proc. 15th Workshop of the UK Planning and Scheduling Special Interest Group, 1996. Available from http://www.hds.utc.fr/baptiste/.
[7] P. Baptiste, C. Le Pape, W. Nuijten, Constraint-based optimization and approximation for job-shop scheduling, in: Proc. AAAI-SIGMAN Workshop on Intelligent Manufacturing Systems, IJCAI-95, Montreal, Quebec, 1995.
[8] P. Baptiste, C. Le Pape, W. Nuijten, Incorporating efficient operations research algorithms in constraint-based scheduling, in: Proc. First Joint Workshop on Artificial Intelligence and Operations Research, 1995. Workshop Proceedings available on World Wide Web from http://www.cirl.uoregon.edu/aior/.
[9] J.E. Beasley, OR-library: Distributing test problems by electronic mail, J. Oper. Res. Soc. 41 (11) (1990) 1069–1072. Also available by ftp from ftp://graph.ms.ic.ac.uk/pub/paper.txt.
[10] J.C. Beck, Texture measurements as a basis for heuristic commitment techniques in constraint-directed scheduling, Ph.D. Thesis, University of Toronto, 1999.
[11] J.C. Beck, A.J. Davenport, E.D. Davis, M.S. Fox, The ODO project: Toward a unified basis for constraint-directed scheduling, J. Scheduling 1 (2) (1998) 89–125.
[12] J.C. Beck, A.J. Davenport, M.S. Fox, Five pitfalls of empirical scheduling research, in: G. Smolka (Ed.), Proc. 3rd International Conference on Principles and Practice of Constraint Programming (CP-97), Springer, Berlin, 1997, pp. 390–404.
[13] J.C. Beck, A.J. Davenport, E.M. Sitarski, M.S. Fox, Beyond contention: Extending texture-based scheduling heuristics, in: Proc. AAAI-97, Providence, RI, AAAI Press, Menlo Park, CA, 1997.
[14] J.C. Beck, A.J. Davenport, E.M. Sitarski, M.S. Fox, Texture-based heuristics for scheduling revisited, in: Proc. AAAI-97, Providence, RI, AAAI Press, Menlo Park, CA, 1997.
[15] J.C. Beck, M.S. Fox, A generic framework for constraint-directed search and scheduling, AI Magazine 19 (4) (1998) 101–130.
[16] J.C. Beck, M.S. Fox, Scheduling alternative activities, in: Proc. AAAI-99, Orlando, FL, AAAI Press, Menlo Park, CA, 1999.
[17] J.C. Beck, K. Jackson, Constrainedness and the phase transition in job shop scheduling, Technical Report, School of Computing Science, Simon Fraser University, Burnaby, BC, 1997.
[18] J. Blazewicz, W. Domschke, E. Pesch, The job shop scheduling problem: Conventional and new solution techniques, European J. Oper. Res. 93 (1) (1996) 1–33.
[19] J.L. Bresina, Heuristic-based stochastic sampling, in: Proc. AAAI-96, Portland, OR, 1996, pp. 271–278.
[20] J. Carlier, E. Pinson, An algorithm for solving the job-shop problem, Management Science 35 (2) (1989) 164–176.
[21] J. Carlier, E. Pinson, Adjustment of heads and tails for the job-shop problem, European J. Oper. Res. 78 (1994) 146–161.
[22] Y. Caseau, F. Laburthe, Improved CLP scheduling with task intervals, in: Proc. 11th International Conference on Logic Programming, MIT Press, Cambridge, MA, 1994.
[23] Y. Caseau, F. Laburthe, Improving branch and bound for jobshop scheduling with constraint propagation, in: Proc. 8th Franco-Japanese Conference CCS-95, 1995.

[24] A. Cesta, A. Oddi, S.F. Smith, An iterative sampling procedure for resource constrained project scheduling with time windows, in: Proc. IJCAI-99, Stockholm, Sweden, 1999.
[25] C.C. Cheng, S.F. Smith, Applying constraint satisfaction techniques to job shop scheduling, Ann. Oper. Res. (Special Volume on Scheduling: Theory and Practice) 70 (1997) 327–378.
[26] P.R. Cohen, Empirical Methods for Artificial Intelligence, MIT Press, Cambridge, MA, 1995.
[27] 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, Toronto, Ontario, 1994.
[28] E.D. Davis, M.S. Fox, Odo: A constraint-based scheduling shell, in: Proc. IJCAI-93 Workshop on Production Planning, Scheduling and Control, Chambéry, France, 1993.
[29] R. Dechter, A. Dechter, J. Pearl, Optimization in constraint networks, in: R. Oliver, J. Smith (Eds.), Influence Diagrams, Belief Nets, and Decision Analysis, Wiley, Chichester, England, 1990.
[30] J. Erschler, F. Roubellat, J.P. Vernhes, Finding some essential characteristics of the feasible solutions for a scheduling problem, Oper. Res. 24 (1976) 772–782.
[31] J. Erschler, F. Roubellat, J.P. Vernhes, Characterising the set of feasible sequences for *n* jobs to be carried out on a single machine, European J. Oper. Res. 4 (1980) 189–194.
[32] H. Fisher, G.L. Thompson, Probabilistic learning combinations of local job-shop scheduling rules, in: J.F. Muth, G.L. Thompson (Eds.), Industrial Scheduling, Prentice Hall, Englewood Cliffs, NJ, 1963, pp. 225–251.
[33] M.S. Fox, Constraint-directed search: A case study of job-shop scheduling, Ph.D. Thesis, CMU-RI-TR-85-7, Carnegie Mellon University, Intelligent Systems Laboratory, The Robotics Institute, Pittsburgh, PA, 1983.
[34] M.S. Fox, N. Sadeh, C. Baykan, Constrained heuristic search, in: Proc. IJCAI-89, Detroit, MI, 1989, pp. 309–316.
[35] E.C. Freuder, Synthesizing constraint expressions, Comm. ACM 21 (11) (1978) 958–966.
[36] E.C. Freuder, A sufficient condition for backtrack-free search, J. ACM 29 (1) (1982) 24–32.
[37] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, New York, 1979.
[38] I. Gent, E. MacIntyre, P. Prosser, B. Smith, T. Walsh, An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem, in: E.C. Freuder (Ed.), Proc. 2nd International Conference on Principles and Practice of Constraint Programming (CP-96), Springer, Berlin, 1996, pp. 179–193.
[39] I.P. Gent, E. MacIntyre, P. Prosser, T. Walsh, The constrainedness of search, in: Proc. AAAI-96, Portland, OR, Vol. 1, 1996, pp. 246–252.
[40] I.P. Gent, T. Walsh, The hardest random SAT problems, in: B. Nebel, L. Dreschler-Fischer (Eds.), Proc. KI-94: Advances in Artificial Intelligence. 18th German Annual Conference on Artificial Intelligence, Springer, Berlin, 1994, pp. 355–366.
[41] C.P. Gomes, B. Selman, H. Kautz, Boosting combinatorial search through randomization, in: Proc. AAAI-98, Madison, WI, 1998, pp. 431–437.
[42] R.M. Haralick, G.L. Elliot, Increasing tree search efficiency for constraint satisfaction problems, Artificial Intelligence 14 (1980) 263–314.
[43] W.D. Harvey, Nonsystematic backtracking search, Ph.D. Thesis, Department of Computer Science, Stanford University, 1995.
[44] W.D. Harvey, M.L. Ginsberg, Limited discrepancy search, in: Proc. IJCAI-95, Montreal, Quebec, 1995, pp. 607–613.
[45] J.N. Hooker, V. Vinay, Branching rules for satisfiability, J. Automat. Reason. 15 (1995) 359–383.
[46] V. Kumar, Algorithms for constraint satisfaction problems: A survey, AI Magazine 13 (1) (1992) 32–44.
[47] S. Lawrence, Resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques (Supplement), Ph.D. Thesis, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA, 1984.
[48] C. Le Pape, Implementation of resource constraints in ILOG Schedule: A library for the development of constraint-based scheduling systems, Int. Syst. Engrg. 3 (2) (1994) 55–66.
[49] 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.

[50] C. Le Pape, P. Baptiste, Constraint propagation techniques for disjunctive scheduling: The preemptive case, in: Proc. 12th European Conference on Artificial Intelligence (ECAI-96), Budapest, Hungary, 1996.
[51] C. Le Pape, P. Baptiste, An experimental comparison of constraint-based algorithms for the preemptive job shop scheduling problem, in: Proc. CP-97 Workshop on Industrial Constraint-Directed Scheduling, Schloss Hagenberg, Austria, 1997.
[52] O. Lhomme, Consistency techniques for numeric CSPs, in: Proc. IJCAI-93, Chambery, France, Vol. 1, 1993, pp. 232–238.
[53] A.K. Mackworth, Consistency in networks of relations, Artificial Intelligence 8 (1977) 99–118.
[54] P. Martin, D.B. Shmoys, A new approach to computing optimal schedules for the job shop scheduling problem, in: Proc. 5th Conference on Integer Programming and Combinatorial Optimization, 1996.
[55] D. Mitchell, B. Selman, H. Levesque, Hard and easy distributions of SAT problems, in: Proc. AAAI-92, San Jose, CA, 1992, pp. 459–465.
[56] N. Muscettola, Scheduling by iterative partition of bottleneck conflicts, Technical Report CMU-RI-TR-92-05, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 1992.
[57] N. Muscettola, On the utility of bottleneck reasoning for scheduling, in: Proc. AAAI-94, Seattle, WA, 1994, pp. 1105–1110.
[58] 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.
[59] W.P.M. Nuijten, E.H.L. Aarts, D.A.A. van Arp Taalman Kip, K.M. van Hee, Randomized constraint satisfaction for job shop scheduling, in: Proc. IJCAI-93 Workshop on Knowledge-Based Production, Scheduling and Control, Chambéry, France, 1993, pp. 251–262.
[60] A. Oddi, S.F. Smith, Stochastic procedures for generating feasible schedules, in: Proc. AAAI-97, Providence, RI, AAAI Press, Menlo Park, CA, 1997.
[61] J. Régin, A filtering algorithm for constraints of difference in CSPs, in: Proc. AAAI-94, Seattle, WA, Vol. 1, 1994, pp. 362–367.
[62] J. Régin, Generalized arc consistency for global cardinality constraint, in: Proc. AAAI-96, Portland, OR, Vol. 1, 1996, pp. 209–215.
[63] N. Sadeh, Lookahead techniques for micro-opportunistic job-shop scheduling, Ph.D. Thesis, CMU-CS-91-102, Carnegie-Mellon University, Pittsburgh, PA, 1991.
[64] N. Sadeh, Micro-opportunistic scheduling, in: M. Zweben, M.S. Fox (Eds.), Intelligent Scheduling, Morgan Kaufmann, San Francisco, CA, 1994, Chapter 4, pp. 99–138.
[65] N. Sadeh, M.S. Fox, Preference propagation in temporal/capacity constraint graphs, Technical Report CMU-RI-TR-89-2, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 1989.
[66] N. Sadeh, M.S. Fox, Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem, Artificial Intelligence 86 (1) (1996) 1–41.
[67] H.A. Simon, The structure of ill-structured problems, Artificial Intelligence 4 (1973) 181–200.
[68] B.M. Smith, S.A. Grant, Trying harder to fail first, Technical Report 97.45, School of Computer Science, University of Leeds, 1997.
[69] P.S. Smith, S.F. Ow, D.C. Matthys, J.Y. Potvin, OPIS: An opportunistic factory scheduling system, in: Proc. International Symposium for Computer Scientists, 1989.
[70] S.F. Smith, Exploiting temporal knowledge to organize constraints, Technical Report CMU-RI-TR-83-12, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 1983.
[71] S.F. Smith, OPIS: A methodology and architecture for reactive scheduling, in: M. Zweben, M.S. Fox (Eds.), Intelligent Scheduling, Morgan Kaufmann, San Francisco, CA, 1994, Chapter 2, pp. 29–66.
[72] S.F. Smith, C.C. Cheng, Slack-based heuristics for constraint satisfaction scheduling, in: Proc. AAAI-93, Washington, DC, 1993, pp. 139–144.
[73] K. Stergiou, T. Walsh, Encodings of non-binary constraint satisfaction problems, in: Proc. AAAI-99, Orlando, FL, 1999, pp. 163–168.
[74] E. Taillard, Benchmarks for basic scheduling problems, European J. Oper. Res. 64 (1993) 278–285.
[75] R.J.M. Vaessens, E.H.L. Aarts, J.K. Lenstra, Job shop scheduling by local search, Technical Report COSOR Memorandum 94-05, Eindhoven University of Technology, 1994. Submitted for publication in INFORMS Journal on Computing.
[76] P. Van Hentenryck, Constraint Satisfaction in Logic Programming, MIT Press, Cambridge, MA, 1989.