Texture-Based Heuristics for Scheduling Revisited

J. Christopher Beck[*], Andrew J. Davenport[‡], Edward M. Sitarski[†], Mark S. Fox[*‡]

Department of Computer Science[*] and Department of Industrial Engineering[‡]
University of Toronto
Toronto, Ontario, CANADA, M5S 3G9
{chris, andrewd, msf}@ie.utoronto.ca

Numetrix Limited[†]
655 Bay St., Suite 1200
Toronto, Ontario, CANADA, M5G 2K4
ed@tor.numetrix.com

Abstract

Recent scheduling work has challenged the need for sophisticated heuristics such as those based on texture measurements. This paper examines these claims in the light of advances in scheduling technology. We compare a number of current heuristic commitment techniques against a texture-based heuristic. Our results demonstrate that texture-based heuristics can outperform these widely-used heuristic commitment techniques.

Introduction

Our research goal is to be able to model and quickly solve scheduling problems as they exist in the real world. We are less interested in optimal solutions than in fast approximate solutions: a quickly found solution that takes into account all the constraints in the real problem is of significantly more use than an optimal solution that either takes too long to find or does not accurately represent the problem. We are applying and extending constraint-directed scheduling techniques toward this end.

Our search philosophy is to spend significant but low polynomial effort ($O(n^2)$ or even $O(n^3)$) in the analysis of each search state to find the most critical constraint in that state, and then make a heuristic decision to reduce this criticality. We believe this will lead to strong algorithms due to the decomposition of a problem after critical decisions are made and the related minimization of the need for backtracking. Investigation of the validity of this philosophy of constraint-directed problem solving forms our long-term research agenda.

In this paper we address the use of *texture measurements* (Fox et al., 1989). Texture measurements are a foundation for sophisticated heuristic decision making and, as such, are complementary to recent advances in scheduling research (*e.g.*, edge-finding (Carlier and Pinson, 1989; Nuijten, 1994), constraint-based analysis (Smith and Cheng, 1993; Cheng and Smith, 1996), limited discrepancy search (Harvey and Ginsberg, 1995)).

A number of criticisms have appeared in the scheduling literature with regard to texture-based heuristics. In particular, it is claimed that:

- Texture measurements are too complicated and equal performance can be achieved with simpler heuristics (Smith and Cheng, 1993).
- A simple heuristic with sophisticated consistency techniques (edge-finding) outperforms texture-based heuristics with the same consistency techniques (Nuijten, 1994).

In this paper we re-evaluate texture-based heuristics in light of recent advances in scheduling technology and show that on two job shop scheduling problem sets (a widely used set of Operations Research benchmark problems and a set of randomly generated, hard problems) a texture-based heuristic outperforms heuristic commitment techniques found in the literature.

The $n \times m$ Job Shop Scheduling Problem 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 A must execute before B (that is, $A \rightarrow B$).
- 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 the last activity in the job must finish). In the decision problem, the release date of each job is 0 and a global due date is D. The problem is to determine whether there is an assignment of a start-time to each activity such that the constraints are satisfied and the maximum finish-time of all jobs is less than or equal to D. This problem is NP-complete (Garey and Johnson, 1979).

Notation For an activity, A_i, ST_i is the start-time variable and STD_i is the discrete domain of possible start-times. est_i and lst_i represent the earliest and latest possible start-times, while eft_i and lft_i represent the earliest and latest possible finish-times. dur_i is the duration of A_i. We will omit the subscript unless there is the possibility of ambiguity.

Texture Measurements for Scheduling Heuristics

It is our conjecture that an understanding of the structure of a problem will lead to more effective methods to solve the problem. Therefore, our goal is to formalize the concept of problem structure. Experience of both Operations Research and constraint-directed problem solving provides evidence that many problems can be adequately modeled by a constraint graph and that the structure of these graphs has significant impact on the efficacy of problem solving methods. Therefore we adopt the view that a problem's structure is defined by its constraint graph.

Given a constraint graph representation, are there problem invariant measurements of the problem topology that may form a basis for heuristic problems solving techniques? A number of such measures, called *texture measurements*, have been identified and experimented with (Fox et al., 1989; Sadeh, 1991). A texture is a measurement of a fundamental property of a constraint graph. A search heuristic (*e.g.*, variable and value ordering heuristics) is a function of one or more textures. *A texture measurement is not a heuristic.* Rather, it is a method of gathering information from the constraint graph which heuristics can then use. 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 decisions are made.

The chief example of the use of texture measurements in scheduling is the Operation Resource Reliance/Filtered Survivable Schedules (ORR/FSS) heuristic implemented in the MicroBOSS Scheduler (Sadeh, 1991; Sadeh and Fox, 1996). ORR/FSS assigns start-times to activities using the *contention* and *reliance* texture measurements as a basis for the variable and value ordering heuristics. In general, contention is the extent to which two variables, connected by a disequality constraint, compete for the same value. Similarly, reliance is the extent to which a variable must be assigned a particular value if a solution is to be found. In scheduling, contention is the extent to which two activities compete for the same resource at the same time while reliance is the extent to which an activity must be executing on a specific resource at a particular time point in order to find a schedule. The exact calculation of these textures is prohibitively expensive, therefore, an estimation of the actual textures are used.

MicroBOSS identifies the most contended-for {resource, time interval} and assigns it to the activity that is most reliant on that resource and time interval. Simple chronological backtracking and intelligent backtracking (Sadeh et al., 1995) have been used in MicroBOSS to solve a number of constraint satisfaction and constraint optimization problems better than the best existing dispatch rules.

The SumHeight Heuristic[1]

The texture-based heuristic used here, SumHeight, is a variation on ORR/FSS. As with ORR/FSS, 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.

1. By referring to the "SumHeight heuristic" we are conflating two notions: the estimation of the texture measurements and the heuristic based on that estimation. Strictly speaking, SumHeight is the algorithm for the estimation of contention and reliance, that is, for the calculation of the individual and aggregate demand curves. The heuristic then, based on the contention and reliance estimate, identifies the commitment to be made. Because SumHeight is the only texture measurement estimation algorithm used in this paper, we refer to the SumHeight heuristic. It should be noted however that the same heuristic technique may be based on a different underlying texture measurement estimation algorithm (see (Beck et al., 1997b) for examples of such).

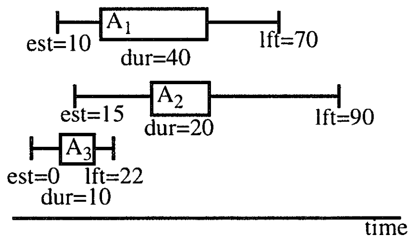

Figure 1. Earliest Start-times, Latest Finish-times, and Durations of A_1, A_2, and A_3

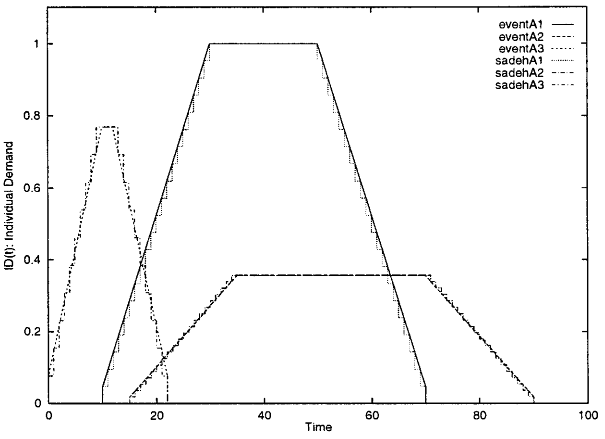

Figure 2. Individual Demand Curves for A_1, A_2, and A_3

3. Analyzes the consequences of each sequence possibility ($A \rightarrow B$ and $B \rightarrow A$) and chooses the one that appears to be superior.

The intuition is that by focusing on the most critical resource and activities, we can make a decision that reduces the likelihood of reaching a search state where the resource is over-capacitated and furthermore once such critical decisions are made the problem is likely to be decomposed into simpler sub-problems.

In the balance of this section, we present our estimations of the contention and reliance textures and expand on the SumHeight heuristic. These calculations rely heavily on Sadeh's original formulation (Sadeh, 1991).

Calculating Individual Demand

To calculate an activity's demand for a particular resource, a uniform probability distribution over the possible start-times for an activity is assumed: each start-time has probability $1/|STD|$.[2] The individual demand, $ID(A,R,t)$, is (probabilistically) the amount of resource R, required by activity A, at time t. It is calculated as follows, for all $est_A \leq t < lft_A$:

$$ID(A, R, t) = \frac{min(t, lst_A) - max(t - dur_A + 1, est_A)}{|STD|} \quad (1)$$

For example, we have 3 activities, A_1, A_2, and A_3, on resource, R_1. The earliest start-times, the latest finish-times, and the durations of each activity are shown in Figure 1.

2. A uniform probability distribution is the "low knowledge" default. Local propagation of value preferences can be used to find a different estimate of the individual demand (Sadeh, 1991; Muscettola, 1992).

The individual demand curves of each activity as calculated by Equation (1) are illustrated in Figure 2 (the "step" functions: sadehA1, sadehA2, and sadehA3). For an intuitive feel for the calculation of the curve for A_3, one may imagine placing A_3 at its earliest start-time, 0, and, for all time points at which A_3 is "executing", adding a demand of 1/13 (A_3 can start at time points 0 through 12 inclusive). We can then place A_3 at its second earliest start-time, 1, and again add 1/13 for all the time points and so on through to the latest start-time of A_3.

A naive calculation of individual demand results in an algorithm that scales with the number of activities and resources, and with the length of the scheduling horizon. To escape the dependence on the scheduling horizon, 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:

$$\left(est, \frac{1}{|STD|}\right), \left(lst, \frac{min(|STD|, dur)}{|STD|}\right), \left(eft, \frac{min(|STD|, dur)}{|STD|}\right), (lft, 0) \tag{2}$$

The use of Equation (1) to calculate the points in Expression (2) results in the second set of curves displayed in Figure 2 (eventA1, eventA2, and eventA3).

Aggregating Individual Demand

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.

With the example used above, the aggregate resource demand curve for resource R_1 is shown in Figure 3 (sadehR1 is the aggregate curve using Sadeh's original formulation and eventR1 is the aggregate curve using our event-based implementation).

Finding the Critical Activities

Once the aggregate demand curves are found for each resource, we identify the resource and time point for which there is the highest contention (with ties broken arbitrarily). We then examine the activities that contribute individual demand to the resource at that time point. The two activities which are not connected to each other by a path of temporal constraints and which contribute the most demand to the aggregate resource demand at the critical time point are defined to be the most critical activities. 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 Figure 3 that one of the critical time points on R_1 is 35. There are only two activities that contribute to this time point, as A_3's latest end-time is 22 (see Figure 2). Therefore, A_1 and A_2 are selected as the critical activities.

Sequencing the Critical Activities

ORR/FSS assigns start-times to activities and so identifies the single most reliant activity and uses a value ordering heuristic to determine a value assignment.[3] Because we post sequencing constraints between the two most critical activities, rather than assigning start-times, we do not use

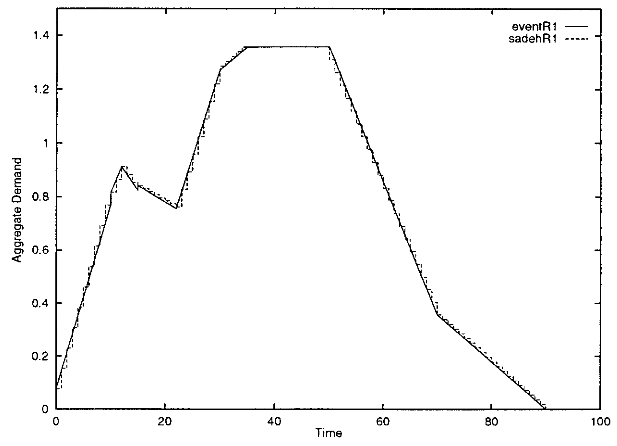

Figure 3. The Aggregate Demand Curve on R_1

Sadeh's value ordering. Instead, 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 MinimizeMax (MM) identifies the commitment which satisfies the following:

$$MM = min(max_{AD'}(A, B), max_{AD'}(B, A)) \tag{3}$$

where:

$$max_{AD'}(A, B) = max(AD'(A, A \rightarrow B), AD'(B, A \rightarrow B)) \tag{4}$$

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

- Given A $\rightarrow$ 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. 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 $\rightarrow$ B) by adding Δ*height* to the height of aggregate demand curve at *tp*.

The same calculation is done for *AD'*(B, A $\rightarrow$ B) and the maximum (as shown in Equation (4)) is used in $max_{AD'}$(A, B). Equation (3) indicates that we choose the commitment resulting in the lowest maximum aggregate curve height. The intuition is that since we are trying to reduce contention, we estimate the worst case increase and then make the commitment that avoids it.

Centroid Sequencing Heuristic The centroid of the individual demand curve is the time point that equally divides the area under the curve.[4] 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 $\rightarrow$ B). Centroid is a variation of a heuristic due to (Muscettola, 1992).

3. ORR/FSS also uses a time interval equal to the average activity duration rather than a single time point in identifying the critical resource and activities.

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

Random Sequencing Heuristic Randomly choose one of the sequencings.

In our example the MinimizeMax heuristic calculates that $max_{AD'}(A_1, A_2) = 1.357$ and $max_{AD'}(A_2, A_1) = 1.700$. Therefore the commitment $A_1 \rightarrow A_2$ is made.

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 all the event points and then, with a single pass, generate the aggregate curve. This process has complexity of $O(mn \log n) + O(mn)$. 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 $O(n^2) + O(mn \log n) + O(mn)$.

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.

Alternative Heuristics for Scheduling

A number of techniques have been suggested for making heuristic decisions in the job shop scheduling problem. We look at three of these in this section.

The CBASlack Heuristic

The CBASlack heuristic was originally proposed as part of the Precedence Constraint Posting (PCP) algorithm (Smith and Cheng, 1993; Cheng and Smith, 1996). PCP uses the Constraint-Based Analysis (CBA) (Erschler et al., 1976; Erschler et al., 1980) propagator to find implied commitments and the CBASlack heuristic to identify heuristic commitments.

CBASlack analyzes all pairs of activities, not connected by a path of temporal constraints, on each resource. The heuristic decision is made using the *Bslack* ("biased-slack") calculation in Equations (5) through (7).

$$Bslack(A_i \rightarrow A_j) = \frac{slack(A_i \rightarrow A_j)}{\sqrt{S}} \tag{5}$$

$$S = \frac{min(slack(A_i \rightarrow A_j), slack(A_j \rightarrow A_i))}{max(slack(A_i \rightarrow A_j), slack(A_j \rightarrow A_i))} \tag{6}$$

$$slack(A_i \rightarrow A_j) = lft_j - est_i - (dur_i + dur_j) \tag{7}$$

Bslack is calculated for all temporally unconnected pairs of activities on the same resource and the pair with the smallest *Bslack* value is identified as the most critical. The sequence of the critical pair that preserves the most slack is the one chosen. The intuition here is that a pair with a smaller *Bslack* is closer to being non-viable than one with a larger value. Once this pair is found, it is important to leave as much temporal slack as possible to decrease the likelihood of backtracking to this decision.

The time complexity of the CBASlack heuristic is $O(mn^2)$ as all pairs of activities on each resource may have to be examined.

The Left-Justified Randomized Heuristic

Nuijten (Nuijten et al., 1993; Nuijten, 1994) presents a randomized scheduling algorithm, SOLVE, which uses powerful propagation techniques (edge-finding) to identify commitments that are implied by the search state. Heuristic decisions are made using the Left-Justified Randomized Heuristic (LJRand). LJRand finds the smallest earliest finish-time of all the unscheduled activities and then identifies the set of activities which are able to start before this time. One of the activities in this set is selected randomly and scheduled at its earliest start-time. Bounded chronological backtracking with restart is used to escape dead-ends.

Experiments compared SOLVE with the ORR/FSS heuristic (using chronological backtracking, temporal propagation and arc-consistency propagation on the resource constraints) and with ORR/FSS augmented with the consistency techniques used in SOLVE. These experiments found that SOLVE strongly outperforms augmented ORR/FSS, which in turn strongly outperforms unaugmented ORR/FSS. It is not clear whether chronological backtracking or the ORR/FSS heuristic are to blame for the poor performance relative to SOLVE.

The time-complexity of the LJRand heuristic is $O(mn)$ as all the unsequenced activities must be examined and then one randomly chosen.

The FirstCommit Heuristic

(Baptiste et al., 1995) propose a technique for making heuristic decisions implemented with the ILOG Schedule constraint library (Le Pape, 1994). The heuristic finds the resource with the least slack and then analyses the unsequenced activities on that resource. Based on comparison of the time-windows available for each unsequenced activity on the critical resource, one is selected to execute first. All activities on the critical resource are sequenced before identification of the next critical resource.

Slack on a resource is defined to be the difference between supply of the resource (the time interval between the minimum *est* and the maximum *lft* of all unsequenced activities) and demand (the sum of the durations of all unsequenced activities). The resource with the smallest slack is selected. Once a resource is selected, an activity is chosen to execute first among the unsequenced activities on the resource. Consistency techniques can be used to identify activities that can not execute first, however it is unclear precisely which are used by (Baptiste et al., 1995). Once the set of activities that can execute first is identified, one activity is selected by the EST–LST rule: the activity with the smallest *est* breaking ties with the smallest *lst*. All activities on a resource are scheduled before moving to another resource.

Following (Baptiste et al., 1995), we have implemented FirstCommit. The resource with smallest slack is identified as described above and the set of activities that can execute first is found using an $O(n)$ consistency algorithm based on (Caseau and Laburthe, 1995). From this set the EST–LST rule is used to choose the activity to execute first. Once the resource is completely sequenced, FirstCommit moves to the next unsequenced resource with minimum slack.

Time-complexity is $O(mn)$ to calculate the resource slack (which is not done for every heuristic commitment) and $O(n)$ to identify the activity to schedule first once the critical resource is known.

Experiments

Our experiments are designed to focus on the evaluation of techniques for making heuristic commitments and so we manipulate only the way in which heuristic commitments are made. Three consistency techniques (edge-finding, CBA, and temporal arc-B-consistency (Lhomme, 1993)) are used after each new heuristic commitment is made. If the domain of possible start-times of any activity becomes empty, a dead end in the search has been reached and we backtrack. We use two backtracking techniques: chronological backtracking and limited discrepancy search (LDS) (Harvey and Ginsberg, 1995). The scheduling algorithm is outlined in Figure 4. The CPU time limit for all experiments is 20 minutes on a 100 MHz. HP 9000/712 running HPUX 9.05.

Experiment 1

We used a set of 21 job shop scheduling problems (see Table 1) from the Operations Research library of benchmark problems (Beasley, 1990). The problems are the union of the problem sets used in (Vaessens et al., 1994) and (Baptiste et al., 1995).

Each heuristic commitment technique is run on a number of instances of each problem with a range of makespans. Specifically, for each problem, the optimal (or best known upper bound) makespan is used initially. If a solution can not be found within the time limit, we increase this makespan by 0.005 times the optimal makespan and re-run the algorithm. Lengthening of the makespan continues until a solution is found. Algorithms are then compared based on the mean relative error between the makespan that they were able to solve and the optimal makespan.

Figures 5 and 6 plot mean relative error for each heuristic against the mean number of heuristic commitments made, for chronological backtracking and LDS respectively. The graph layout means that the better algorithms will be closer to the lower left corner. Note the differing scales of the two graphs. We do not report CPU time because it is dominated by our relatively inefficient implementation of edge-finding. This issue is discussed in detail in (Beck et al., 1997a).

With chronological backtracking, we see that LJRand finds schedules with significantly higher mean relative error (tested with a boot-strap paired *t* test (Cohen, 1995)) than all the other heuristics ($p \leq 0.0001$) with significantly fewer heuristic commitments ($p \leq 0.001$, tested with a boot-strap, two-sample *t* test (Cohen, 1995)). SumHeight finds signifi-

Source	Problem (lower/upper bound)
(Adams et al., 1988)	abz5(1234), abz6(943)
(Fisher and Thompson, 1963)	ft10(930)
(Lawrence, 1984)	la02(655), la19(842), la20(902), la21(1046), la24(935), la25(977), la27(1235), la29(1130/1153), la36(1268), la37(1397), la38(1196), la39(1233), la40(1222)
(Applegate and Cook, 1991)	orb01(1059), orb02(888), orb03(1005), orb04(1005), orb05(887)

Table 1: Test Problems

```
finished := false
while(finished = false){
  edge-finding
  if (edge-finding makes no commitments)
    CBA
  if (no commitments from CBA or
        from edge-finding)
    make heuristic commitment
  if (dead-end)
    backtrack
  else
    arc-B-consistency temporal propagation
  if (all-activities-sequenced OR
        CPU limit reached)
    finished := true
}
```

Figure 4. Basic Scheduling Algorithm

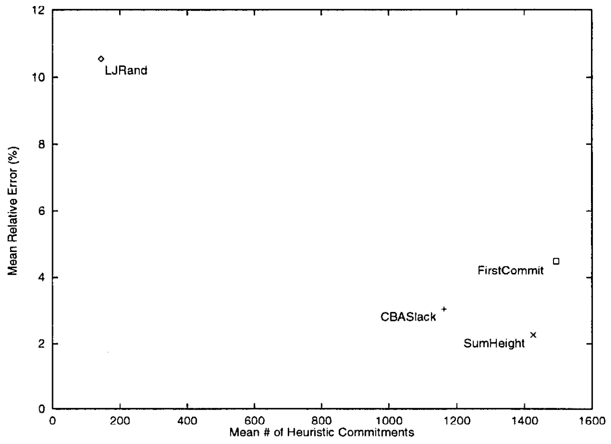

Figure 5. Mean Relative Error vs. Mean # of Heuristic Commitments Using Chronological Backtracking

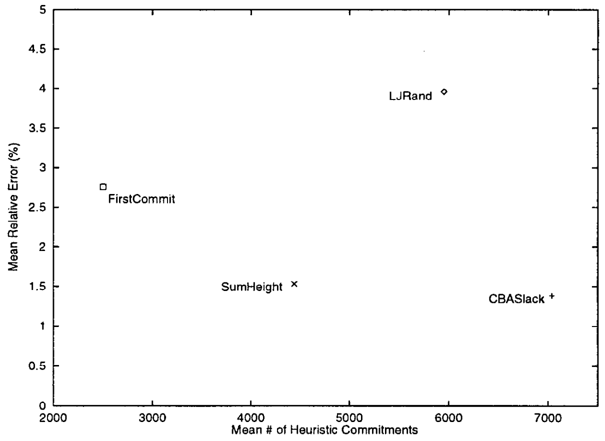

Figure 6. Mean Relative Error vs. Mean # of Heuristic Commitments Using LDS

cantly better schedules that both CBASlack ($p \leq 0.05$) and FirstCommit ($p \leq 0.0001$) and CBASlack in turn finds better schedules than FirstCommit ($p \leq 0.01$). There are no significant differences in the number of heuristic commitments amongst SumHeight, CBASlack, and FirstCommit.

The LDS results show no significant difference between SumHeight and CBASlack in terms of mean relative error but FirstCommit finds significantly worse schedules ($p \leq 0.001$) and LJRand finds significantly worse schedules than FirstCommit ($p \leq 0.0001$). In terms of the number of heuristic commitments the only significant result is that FirstCommit uses fewer heuristic commitments than any of the other heuristics ($p \leq 0.05$).

In terms of comparing chronological backtracking with LDS, we see that all heuristics find significantly better schedules with LDS, although requiring significantly more heuristic commitments to do so. SumHeight has significantly less improvement than all other heuristics ($p \leq 0.05$, tested using the boot-strap paired t test) when it uses LDS as compared to when it uses chronological backtracking. LJRand has significantly more improvement in using LDS than all the others ($p \leq 0.0001$). This may be the result of a ceiling effect (*i.e.,* SumHeight has less room for improvement than LJRand due to its better performance with chronological backtracking), but it appears that LDS is helping the weaker heuristics (as judged by their performance with chronological backtracking) more than the stronger.

Experiment 2

A difficulty with the results of Experiment 1 is that the experimental design, though common in scheduling research, may result in each algorithm solving different problems due to different makespans. This limits the conclusions we can draw from an analysis of the experiments using data other than mean relative error. For example, for the chronological backtracking results we see that LJRand uses significantly fewer heuristic commitments than all other heuristics. Perhaps this is a real effect due to some characteristic of LJRand; however there is an alternative, and, in this case, more likely, explanation: LJRand solves problems with significantly larger makespans and fewer heuristic commitments simply due to the easier problems. Any explanation of performance differences (other than those concerning mean relative error) is suspect: there is always the argument that the difference is due to solving different problems.

To address this issue we ran a second experiment using a larger problem set of varying sizes and (expected) difficulties. Using Taillard's (Taillard, 1993) generator of job shop scheduling problems, we created 5 sets of 60 problems each, with sizes of {10x10, 12x12, 15x15, 18x18, 20x20}. We generated a makespan for each problem so that the problems within each size span the phase transition that has been observed in job shop problems (Beck and Jackson, 1997).

Using a CPU bound of 20 minutes an attempt was made, using each algorithm, to solve each problem. If the bound was reached, failure on that problem was returned. The results using chronological backtracking are shown in Figures 7 and 8 while the LDS results are displayed in Figures 9 and 10. In Figures 8 and 10 (note the differing scales) we display the mean number of heuristic commitments made by each heuristic. For problems that are not solved, we include the number of heuristic commitments that were

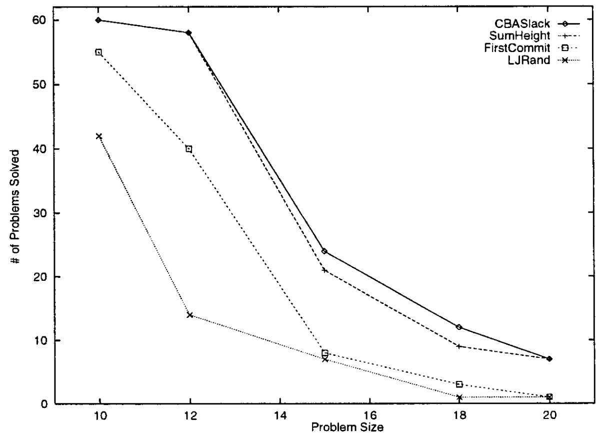

Figure 7. Number of Problems Solved vs. Problem Size for Chronological Backtracking

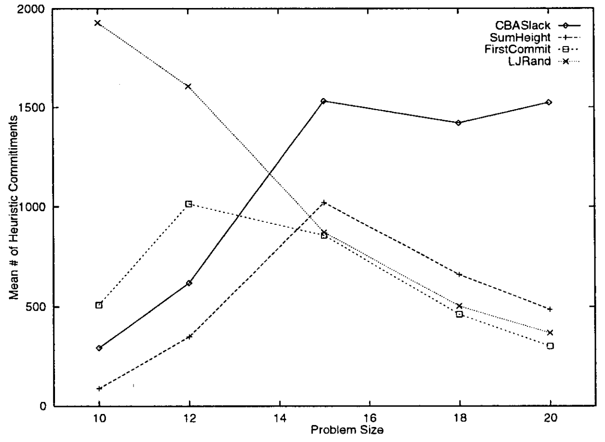

Figure 8. Mean Number of Heuristic Commitments vs. Problem Size for Chronological Backtracking

made when the CPU time limit was reached. Therefore, the mean number of heuristic commitments is a lower bound on the actual number of heuristic commitments required to solve these problems.

In terms of the number of problems solved with chronological backtracking (tested using a boot-strap paired t test on each problem size), there is no significant difference between SumHeight and CBASlack. SumHeight solves significantly more problems than FirstCommit ($p \leq 0.001$) and LJRand ($p \leq 0.005$) as does CBASlack ($p \leq 0.001$ for both). FirstCommit solves significantly more problems than LJRand ($p \leq 0.005$).

Boot-strap, paired t tests on the number of heuristic commitments made for each problem show that all differences in Figure 8 are significant ($p \leq 0.001$) with the exception of that between FirstCommit and LJRand on problems of size 15x15 where there is no significant difference. In particular, SumHeight makes significantly fewer heuristic commitments than CBASlack across all problems sizes ($p \leq 0.0001$ for all sizes except 12x12 where $p \leq 0.001$).

Turning to the results with LDS, we again see no significant difference in terms of the number of problems solved between SumHeight and CBASlack, while each solves significantly more problems than each of FirstCommit and LJRand ($p \leq 0.005$). FirstCommit solves significantly more problems than LJRand ($p \leq 0.05$).

All the differences in the number of heuristic commitments shown in Figure 10 are significant ($p \leq 0.05$) except for those between SumHeight and FirstCommit at both 10×10 and 15×15 and that between LJRand and CBASlack at 18×18. In particular, SumHeight makes significantly fewer heuristic commitments than CBASlack at all problem sizes ($p \leq 0.0001$).

In comparing the chronological backtracking results with those for LDS, we observe that each heuristic solved significantly more problems with LDS than with chronological backtracking. The only significant difference in the magnitude of the difference is between CBASlack and LJRand: using LDS instead of chronological backtracking increases the number of problems LJRand solves significantly more than it increases the number of problems CBASlack solves ($p \leq 0.05$). From the perspective of the number of heuristic commitments, for all heuristics, on problems of size 15×15 or larger the algorithm using LDS makes significantly ($p \leq 0.0001$) more heuristic commitments than that using chronological backtracking. For the 12×12 problems there is no significant differences except for LJRand ($p \leq 0.0001$) and for the 10×10 problems CBASlack shows no significant difference while SumHeight with LDS makes significantly more heuristic commitments than SumHeight with chronological backtracking ($p \leq 0.01$). Interestingly, for FirstCommit and LJRand, chronological backtracking results in more heuristic commitments than LDS on the 10×10 problems ($p \leq 0.005$ and $p \leq 0.05$, respectively).

Summary of Experimental Results

On the OR-Library benchmark problems of Experiment 1 SumHeight outperforms the LJRand and FirstCommit heuristics in terms of the quality of the schedules found within our CPU time bound. The quality of the schedules found by SumHeight was better than those found by CBASlack with chronological backtracking, although equal performance was achieved with LDS.

Experiment 2 indicates that SumHeight and CBASlack solve the same number of problems across all problem sizes while both solve significantly more than either LJRand or FirstCommit. In equaling the performance of CBASlack in terms of the number of problems solved, SumHeight uses significantly fewer heuristic commitments

These results strongly support our contention that texture-based heuristics are still worthwhile in the context of modern scheduling techniques.

Conclusions

In this paper we have examined texture-based heuristics for scheduling in the light of several criticisms concerning their performance, and in light of recent advances in scheduling technology (*e.g.*, constraint propagation techniques, limited discrepancy search).

The central aim of this paper is the comparison of texture-based heuristics with other heuristic commitment techniques for scheduling. We have shown that, with experimental designs that manipulate only the heuristic commitment technique, claims of the inferiority of texture-based heuristics are not supported. In fact, we have demon-

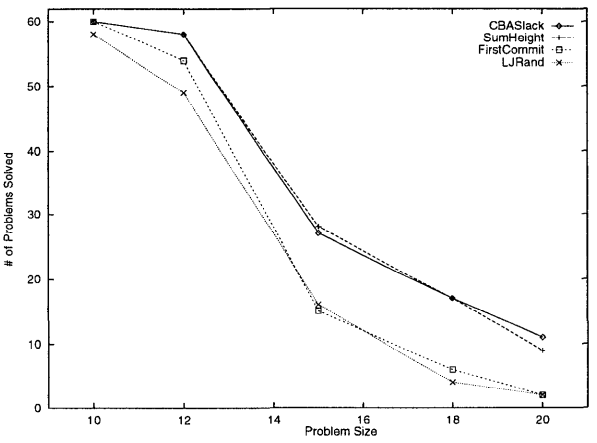

Figure 9. Number of Problems Solved vs. Problem Size for LDS

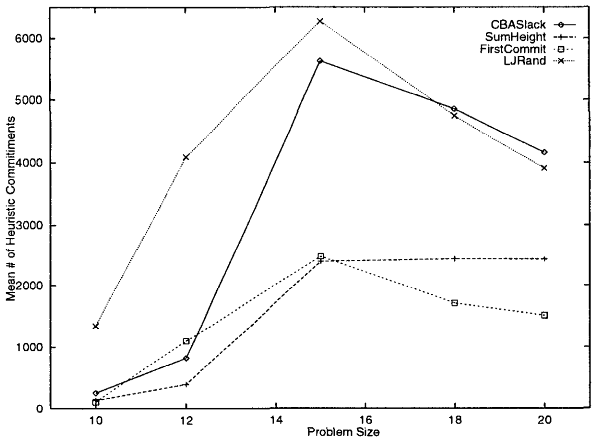

Figure 10. Mean Number of Heuristic Commitments vs. Problem Size for LDS

strated significant performance gain in using such sophisticated heuristics.

In addition, this is the first work of which we are aware that evaluates limited discrepancy search (LDS) in scheduling in the context of state-of-the-art heuristics and propagation techniques. The results strongly show that LDS results in increased performance over chronological backtracking. The fact that LDS appears to help the weaker algorithms (as judged by their performance with chronological backtracking) to a greater extent than the stronger algorithms is interesting but preliminary.

Our results are consistent with our aim: sophisticated heuristics based on texture measurements can outperform other heuristic commitment techniques, and therefore deserve to be re-examined.

Acknowledgments

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

Research Corporation of Ontario, and Digital Equipment of Canada.

Thanks to Ioan Popescu, Rob Morenz, Ken Jackson, Morten Irgens, Victor Saks, Angela Glover, and the anonymous reviewers for comments on and discussion of earlier drafts of this paper.

References

Adams, J., Balas, E., and Zawack, D. (1988). The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34:391–401.

Applegate, D. and Cook, W. (1991). A computational study of the job-shop scheduling instance. *ORSA Journal on Computing*, 3:149–156.

Baptiste, P., Le Pape, C., and Nuijten, W. (1995). Constraint-based optimization and approximation for job-shop scheduling. In *Proceedings of the AAAI-SIGMAN Workshop on Intelligent Manufacturing Systems, IJCAI-95*.

Beasley, J. E. (1990). OR-library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072. Also available by ftp from ftp://graph.ms.ic.ac.uk/pub/paper.txt.

Beck, J. C., Davenport, A. J., and Fox, M. S. (1997a). Pitfalls of empirical scheduling research. Technical report, Department of Industrial Engineering, University of Toronto, Department of Industrial Engineering, 4 Taddle Creek Road, Toronto, Ontario M5S 3G9, Canada.

Beck, J. C., Davenport, A. J., Sitarski, E. M., and Fox, M. S. (1997b). Beyond contention: extending texture-based scheduling heuristics. In *Proceedings of AAAI-97*. AAAI Press, Menlo Park, California.

Beck, J. C. and Jackson, K. (1997). Stalking the wily phase transition: kappa and job shop scheduling. Technical report, Department of Industrial Engineering, University of Toronto, Department of Industrial Engineering, 4 Taddle Creek Road, Toronto, Ontario M5S 3G9, Canada.

Carlier, J. and Pinson, E. (1989). An algorithm for solving the job-shop problem. *Management Science*, 35(2):164–176.

Caseau, Y. and Laburthe, F. (1995). Improving branch and bound for jobshop scheduling with constraint propagation. In *Proceedings of the 8th Franco-Japanese Conference CCS'95*.

Cheng, C. C. and Smith, S. F. (1996). Applying constraint satisfaction techniques to job shop scheduling. *Annals of Operations Research, Special Volume on Scheduling: Theory and Practice*, 1. To appear, forthcoming.

Cohen, P. R. (1995). *Empirical Methods for Artficial Intelligence*. The MIT Press, Cambridge, Mass.

Erschler, J., Roubellat, F., and Vernhes, J. P. (1976). Finding some essential characteristics of the feasible solutions for a scheduling problem. *Operations Research*, 24:772–782.

Erschler, J., Roubellat, F., and Vernhes, J. P. (1980). Characterising the set of feasible sequences for n jobs to be carried out on a single machine. *European Journal of Operational Research*, 4:189–194.

Fisher, H. and Thompson, G. L. (1963). Probabilistic learning combinations of local job-shop scheduling rules. In Muth, J. F. and Thompson, G. L., editors, *Industrial Scheduling*, pages 225–251. Prentice Hall, Englewood Cliffs, New Jersey.

Fox, M. S., Sadeh, N., and Baykan, C. (1989). Constrained heuristic search. In *Proceedings of IJCAI-89*, pages 309–316.

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

Harvey, W. D. and Ginsberg, M. L. (1995). Limited discrepancy search. In *Proceedings of IJCAI-95*, pages 607–613.

Lawrence, S. (1984). *Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (Supplement)*. PhD thesis, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania.

Le Pape, C. (1994). Implementation of resource constraints in ilog schedule: A library for the development of constraint-based scheduling systems. *Intelligent Systems Engineering*, 3(2):55–66.

Lhomme, O. (1993). Consistency techniques for numeric CSPs. In *Proceedings of IJCAI-93*, volume 1, pages 232–238.

Muscettola, N. (1992). Scheduling by iterative partition of bottleneck conflicts. Technical Report CMU-RI-TR-92-05, The Robotics Institute, Carnegie Mellon University.

Nuijten, W. P. M. (1994). *Time and resource constrained scheduling: a constraint satisfaction approach*. PhD thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology.

Nuijten, W. P. M., Aarts, E. H. L., van Arp Taalman Kip, D. A. A., and van Hee, K. M. (1993). Randomized constraint satisfaction for job shop scheduling. In *Proceedings of the IJCAI'93 Workshop on Knowledge-Based Production, Scheduling and Control*, pages 251–262.

Sadeh, N. (1991). *Lookahead techniques for micro-opportunistic job-shop scheduling*. PhD thesis, Carnegie-Mellon University. CMU-CS-91-102.

Sadeh, N. and Fox, M. S. (1996). Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem. *Artificial Intelligence Journal*, 86(1).

Sadeh, N., Sycara, K., and Xiong, Y. (1995). Backtracking techniques for the job shop scheduling constraint satisfaction. *Artificial Intelligence*, 76:455–480.

Smith, S. F. and Cheng, C. C. (1993). Slack-based heuristics for constraint satisfaction scheduling. In *Proceedings AAAI-93*, pages 139–144.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64:278–285.

Vaessens, R. J. M., Aarts, E. H. L., and Lenstra, J. K. (1994). Job shop scheduling by local search. Technical Report COSOR Memorandum 94-05, Eindhoven University of Technology. Submitted for publication in INFORMS Journal on Computing.