Stochastic Procedures for Generating Feasible Schedules

Angelo Oddi*
Dipartimento di Informatica e Sistemistica
Universita' di Roma "La Sapienza"
oddi@assi.dis.uniroma1.it

Stephen F. Smith
The Robotics Institute
Carnegie Mellon University
sfs@cs.cmu.edu

Abstract

In this paper, we investigate the use of stochastic variable and value ordering heuristics for solving job shop scheduling problems with non-relaxable deadlines and complex metric constraints. Previous research in constraint satisfaction scheduling has developed highly effective, deterministic heuristics for this class of problems based on simple measures of temporal sequencing flexibility. However, they are not infallible, and the possibility of search failure raises the issue of how to most productively enlarge the search. Backtracking is one alternative, but such systematicity generally implies high computational cost. We instead design an iterative sampling procedure, based on the intuition that it is more productive to deviate from heuristic advice in cases where the heuristic is less informed, and likewise better to follow the heuristic in cases where it is more knowledgeable. We specify stochastic counterparts to previously developed search heuristics, which are parameterized to calibrate degree of randomness to level of discriminatory power. Experimental results on job shop scheduling CSPs of increasing size demonstrate comparative advantage over chronological backtracking. Comparison is also made to another, recently proposed iterative sampling technique called heuristic-biased stochastic sampling (HBSS). Whereas HBSS assumes a statically specified heuristic bias that is utilized at every application of the heuristic, our approach defines bias dynamically according to how well the heuristic discriminates alternatives.[1]

Introduction

Recent research in constraint satisfaction problem solving (CSP) approaches to scheduling has produced strong scheduling search heuristics with fairly broad applicability (e.g., (Cheng & Smith 1994)). However, these heuristics are not guaranteed to succeed, and one problematic aspect of CSP-based scheduling procedures in practical domains is their fall back reliance on backtracking search. For backtracking search to perform reasonably, solutions must be evenly distributed among the "leaves" of the search tree, and unfortunately, this is typically not the case. One common approach to avoiding the computational bottleneck of backtracking has been to instead relax constraints (typically deadlines) if the search arrives at an inconsistent state. Such approximate solution procedures can be particularly useful as components of a larger schedule optimization process (e.g., (Cheng & Smith 1996)). Even so, performance is entirely a function of the quality of the heuristics that are used, and could benefit from the ability to productively enlarge the search. The interesting question here is what middle ground exists between backtrack-free approximate procedures and complete backtracking search.

One interesting alternative is to restart the search from scratch whenever it arrives at an inconsistent state, and take some number of different decisions on each successive pass through the search tree. The simplest example of this search model is iterative sampling (Langley 1992) which randomly explores different paths through the search tree. However, in and of itself, iterative sampling provides no basis for exploiting knowledge of search heuristics. Least Discrepancy Search (LDS) (Harvey & Ginsberg 1995), alternatively, is a restarting procedure designed with the use of a search heuristic in mind. LDS is based on the intuition that paths through the search tree which vary only slightly from the path advocated by the heuristic represent the most promising alternatives and should be explored first. In its most basic form, LDS defines a systematic procedure wherein the path advocated by the heuristic is explored first, followed by all paths which deviate from the heuristic at just one decision point, followed by all paths which deviate at two decision points, and so on. Heuristic-biased stochastic sampling (HBSS) (Bresina 1996) is a second restarting procedure designed to utilize a search heuristic as the basis for conducting a broader-based search. In this case, a heuristic bias function is used to encode the level of confidence one has in the heuristic's advice and this function is used to modulate a stochastic decision procedure akin to iterative sampling. Both HBSS and

*Present address: IP-CNR, Viale Marx 15, I-00137 Rome, Italy

LDS have been effectively applied to specific scheduling problems.

One common characteristic of LDS and HBSS is that they both assume context independence with regard to the quality of the available search heuristic (or heuristics). In HBSS, for example, the heuristic bias function influences the degree to which the heuristic is to be followed in the same way at every choice point. Similarly, in LDS all paths which deviate in only one choice from those of the heuristic are considered more promising than paths which deviate in two choices, regardless of the specific nature of the respective decisions. In actuality, a given search heuristic will typically provide different amounts of leverage at different choice points in the search. It may be more or less informed at different stages of the search, more or less discriminating when applied to different sets of alternatives, etc. In designing an extended search procedure, one would ideally like to concentrate exploration around those decisions where the heuristic guidance is weakest and most susceptible to error.

In this paper, we develop and evaluate such an extended-search procedure for solving scheduling problems with non-relaxable deadlines. Our starting point is a high-performance, deterministic constraint satisfaction scheduling procedure called SP-PCP (Cheng & Smith 1994), whose effectiveness derives from variable and value ordering heuristics that exploit simple measures of temporal sequencing flexibility. Following the intuition behind the original design of these heuristics, we define stochastic counterparts for use within an iterative sampling search framework which promote randomness in inverse proportion to the amount of discriminatory power that each heuristic provides in any given choice context. We apply this revised search procedure to the extended job-shop problem previously addressed in (Cheng & Smith 1994), which requires attendance to metric bounds on both the durations of individual operations and the separation times between successive operations of the same job, and demonstrate the cost/performance leverage of our approach over the originally developed backtracking search model. We also contrast the performance of our procedure to that of HBSS, to evaluate the advantage of dynamic (or context specific) heuristic bias. We start by defining the scheduling problem of interest and reviewing the deterministic SP-PCP scheduling model.

Preliminary Definitions

The extended Job Shop Deadline Scheduling Problem (JSDSP) considered in (Cheng & Smith 1994) involves synchronizing the use of a set of resources $R = \{r_1 \ldots r_m\}$ to perform a set of jobs $J = \{j_1 \ldots j_n\}$ over time. The processing of a job j_i requires the execution of a sequence of n_i activities $\{a_{i1} \ldots a_{in_i}\}$, and the execution of each activity a_{ij} is subject to the following constraints:

- resource availability - each a_{ij} requires exclusive use of a single resource $r_{a_{ij}}$ for its entire duration.
- processing time constraints - each a_{ij} has a minimum and maximum processing time, $proc_{ij}^{min}$ and $proc_{ij}^{max}$, such that $proc_{ij}^{min} \leq e_{ij} - s_{ij} \leq proc_{ij}^{max}$, where the variables s_{ij} and e_{ij} represent the start and end times respectively of a_{ij}.
- separation constraints - for each pair of successive activities a_{ij} and a_{ij+1}, $j = 1 \ldots n_i - 1$, in job j_i, there is a minimum and maximum separation time, sep_{ik}^{min} and sep_{ik}^{max}, such that $\{sep_{ik}^{min} \leq s_{ik+1} - e_{ik} \leq sep_{ik}^{max} : k = 1 \ldots n-1\}$.
- job release and due dates - Every job j_i has a release date rd_i, which specifies the earliest time that any a_{ij} can be started, and a due date dd_i, which designates the time by which all a_{ij} must be completed.

There are different ways to formulate this problem as a CSP. In (Smith & Cheng 1993), the problem is treated as one of establishing precedence constraints between pairs of activities that require the same resource, so as to eliminate all possible conflicts in resource use. In CSP terms, a decision variable O_{ijr} is defined for each pair of activities a_i and a_j requiring resource r, which can take on one of two values: $a_j\{before\}a_i$ or $a_i\{before\}a_j$.

To support the search for a consistent assignment to this set of decision variables, we can define for any JSDSP a directed graph $G_d(V,E)$, where the set of nodes V represents time points (i.e., the origin point and the start and end time points, s_{a_i} and e_{a_i}, of each activity a_i) and the set of edges E represents temporal distance constraints. For every constraint of the form $a \leq tp_j - tp_i \leq b$ specified in JSDSP, there are two weighted edges in the graph $G_d(V,E)$. The first one is directed from tp_i to tp_j with weight b and the second one is directed from tp_j to tp_i with weight $-a$. The graph $G_d(V,E)$ corresponds to a Simple Temporal Problem (Dechter, Meiri, & Pearl 1991) and its consistency can be efficiently determined via shortest path computations. Thus, a search for a solution to JSDSP can proceed by repeatedly adding posting) new precedence constraints into $G_d(V,E)$ and recomputing shortest path lengths to confirm that $G_d(V,E)$ remains consistent (i.e., no negative weight cycles). Let $d(tp_i, tp_j)$ designate the shortest path length in graph $G_d(V,E)$ from node tp_i to node tp_j.

The SP-PCP Scheduling Procedure

The SP-PCP scheduling procedure (Shortest Path-based Precedence Constraint Posting) (Cheng & Smith 1994) utilizes shortest path information in $G_d(V,E)$ in two ways to enhance the basic search process sketched above. First, it is possible to define *dominance conditions*, which identify unconditional decisions and promote early pruning of alternatives. For any pair of activities a_i and a_j that are competing for the same

resource, 4 possible cases of conflict are defined:

1. $d(e_{a_i}, s_{a_j}) < 0 \ \wedge \ d(e_{a_j}, s_{a_i}) < 0$
2. $d(e_{a_i}, s_{a_j}) < 0 \ \wedge \ d(e_{a_j}, s_{a_i}) \geq 0$
3. $d(e_{a_j}, s_{a_i}) < 0 \ \wedge \ d(e_{a_i}, s_{a_j}) \geq 0$
4. $d(e_{a_i}, s_{a_j}) \geq 0 \ \wedge \ d(e_{a_j}, s_{a_i}) \geq 0$

Condition 1 represents an *unresolvable conflict*. There is no way to sort a_i and a_j without inducing a negative cycle in graph $G_d(V, E)$, and the search has reached an inconsistent state. Conditions 2, and 3, alternatively, distinguish *uniquely resolvable conflicts*. Here, there is only one feasible ordering of a_i and a_j and the decision of which constraint to post is thus unconditional. In the case of Condition 2, only $a_j\{before\}a_i$ leaves $G_d(V, E)$ consistent and similarly, only $a_i\{before\}a_j$ is feasible in the case of Condition 3. Condition 4 designates a final class of *resolvable conflict*. In this case, both orderings of a_i and a_j remain feasible and it is necessary to make a choice.

It is possible to extend Conditions 2 and 3 above (as originally defined in (Cheng & Smith 1994)) to specify only those circumstances where a_i and a_j can actually overlap in time and thus avoid the posting of redundant constraints. For example, if constraints $a_i\{before\}a_j$ and $a_j\{before\}a_k$ have been posted, then there is no need to post $a_i\{before\}a_k$. The following extended dominance conditions are sufficient to detect such situations:

2. $d(e_{a_i}, s_{a_j}) < 0 \ \wedge \ d(e_{a_j}, s_{a_i}) \geq 0 \ \wedge \ d(s_{a_i}, e_{a_j}) > 0$
3. $d(e_{a_j}, s_{a_i}) < 0 \ \wedge \ d(e_{a_i}, s_{a_j}) \geq 0 \ \wedge \ d(s_{a_j}, e_{a_i}) > 0$

We will assume the use of these stronger dominance conditions in the procedures that are developed and evaluated later in this paper.

The second way in which shortest path information is exploited within SP-PCP is in the definition of variable and value ordering heuristics for selecting and resolving conflicts in the set characterized by Condition 4. In this context, $d(e_{a_i}, s_{a_j})$ and $d(e_{a_j}, s_{a_i})$ provide measures of the degree of sequencing flexibility that remains with respect to a_i and a_j. The SP-PCP variable ordering heuristic attempts to focus first on the conflict with the least amount of sequencing flexibility (i.e., the ordering decision that is closest to being forced). More precisely, the conflict (a_i, a_j) with the overall minimum value of $VarEval(a_i, a_j) = min\{bd_{ij}, bd_{ji}\}$ is always selected for resolution, where:
where

$$bd_{ij} = \frac{d(e_{a_i}, s_{a_j})}{\sqrt{S}}, \quad bd_{ji} = \frac{d(e_{a_j}, s_{a_i})}{\sqrt{S}}$$

and

$$S = \frac{min\{d(e_{a_i}, s_{a_j}), d(e_{a_j}, s_{a_i})\}}{max\{d(e_{a_i}, s_{a_j}), d(e_{a_j}, s_{a_i})\}}$$

The $\sqrt{S}$ bias is introduced to hedge when the conflict with the overall $min\{d(e_{a_i}, s_{a_j}), d(e_{a_j}, s_{a_i})\}$ has a very large $max\{d(e_{a_i}, s_{a_j}), d(e_{a_j}, s_{a_i})\}$, and a second

SP-PCP($Jsdsp$)
1. **loop**
2. **if** Unresolvable-Conflict($Jsdsp$)
3. **then return**(nil)
4. **else**
5. **if** Uniquely-Resolvable-Conflicts($Jsdsp$)
6. **then** Post-Unconditional-Constraints($Jsdsp$)
7. **else begin**
8. $Conflict$::=Choose-Resolvable-Conflict($Jsdsp$)
9. **if** ($Conflict = NIL$)
10. **then return**(solution)
11. **else begin**
12. $Const$::=
Choose-Prec-Constraint($Jsdsp, Conflict$)
13. Post-Constraint($Jsdsp$,$Const$)
14. **end**
15. **end**
16. **end-loop**

Figure 1: Basic SP-PCP Algorithm

conflict has two shortest path values just slightly larger than this overall minimum. In such situations, it is not clear which conflict has the least sequencing flexibility.

The value ordering heuristic used within SP-PCP to resolve a selected conflict (a_i, a_j) simply chooses the precedence constraint that retains the most sequencing flexibility. Specifically, $a_i\{before\}a_j$ is selected if $bd_{ij} > bd_{ji}$ and $a_j\{before\}a_i$ otherwise.

Figure 1 gives the basic overall SP-PCP solution procedure, which interleaves the application of dominance conditions (Steps 2 and 5) with variable and value ordering (Steps 8 and 12 respectively) and incremental updating of the solution graph $G_d(V, E)$ (Steps 6 and 13) to conduct a single pass through the search tree. It is restructured slightly from the algorithm originally given in (Cheng & Smith 1994) to take advantage of the stronger dominance conditions defined above, but is otherwise equivalent to the one-pass, partial solution procedure that was evaluated in this work. The complete, backtracking search version of SP-PCP defined in (Cheng & Smith 1994) is obtained by instead backtracking whenever possible before returning nil in Step 3.

Random SP-PCP Scheduling Procedures

The experimental study conducted in (Cheng & Smith 1994) has demonstrated the effectiveness of the simple scheduling procedure depicted in Figure 1. This study also indicates the added leverage of embedding this partial procedure within a backtracking search framework in terms of an ability to solve a larger number of larger-sized problem instances. At the same time, it is also clear from this study that this ability to solve

RANDOM-SP-PCP($Jsdsp$, $MaxR$)
1. $Solution$ ::= NIL
2. r ::= 0
3. **while** ($r \leq MaxR$) **and not**($Solution$) **do begin**
4. $Solution$ ::= SP-PCP($Jsdsp$)
5. r ::= $r + 1$
6. Reset-Initial-Solution-State($Jsdsp$)
7. **end-while**

Figure 2: Random-SP-PCP Algorithm

additional problems comes at a considerable increase in computational cost, raising the question of whether there exist more efficient ways than backtracking to enlarge the search when variable and value ordering decisions lead to an unresolvable conflict. In this section, we propose the use of an iterative sampling procedure in which variable and value ordering decisions are made in a stochastic, but heuristically-biased manner.

Assuming that at least one of the *Choose-Resolvable-Conflict* or *Choose-Prec-Constraint* steps of the basic SP-PCP algorithm is handled in a stochastic manner, the algorithm shown in Figure 2 implements a basic iterative sampling procedure. The algorithm attempts to solve an instance of JSDSP by exploring random paths in the search tree until either a feasible solution is found or a maximum number of paths have been examined. Every time the algorithm explores a new path in the search tree it starts from scratch (i.e., from an initial solution state where no constraints have yet been posted between pairs of activities requiring the same resource).

Below, we first define stochastic counterparts to SP-PCP's deterministic heuristics which vary the probability that the heuristic's advice will be followed according to its discriminatory power in a given choice context. Then we define a static-bias approach based on Heuristic-Biased Stochastic Sampling (HBSS) (Bresina 1996).

Stochastic Variable and Value Ordering

Our design of stochastic versions of SP-PCP's variable and value ordering heuristics follows from the simple intuition that makes more sense to follow a heuristic's advice when it clearly distinguishes one alternative as superior and it makes less sense to follow its advice when several choices are judged to be equally good.

Let us consider first the case of **variable ordering**. As previously discussed, SP-PCP's variable ordering heuristic selects the conflict (a_i, a_j) with the overall minimum value of $VarEval(a_i, a_j) = min\{bd_{ij}, bd_{ji}\}$. If $VarEval(a_i, a_j)$ is << than $VarEval(a_k, a_l)$ for all other pending conflicts (a_k, a_l), then the selected conflict (a_i, a_j) is clearly distinguished. However, if other $VarEval(a_k, a_l)$ values are instead quite "close" to $VarEval(a_i, a_j)$, then the preferred choice is not clear and selection of any of these conflicts may be reasonable. We formalize this notion by defining an *acceptance band* β with respect to the set of pending resolvable conflicts and expanding the *Choose-Resolvable-Conflict* step of SP-PCP to:

1. Calculate the overall minimum value $\omega = min\{VarEval(a_i, a_j)\}$ as before
2. Determine the subset of resolvable conflicts $SC = \{(a_i, a_j) : \omega \leq VarEval((a_i, a_j)) \leq \omega(1 + \beta)\}$
3. Randomly select a conflict (a_i, a_j) in the set SC.

Thus, β defines a range around the minimum heuristic evaluation within which any differences in evaluations are assumed to be insignificant and non-informative. The smaller the value of β, the higher the assumed discriminatory power of the heuristic. In the experiments reported later, we consider two values for β: 0.05 (in which case all evaluations within 5% of the minimum are considered to be equally good next choices) and 0.1 (in which case all evaluations within 10% are considered equivalent).

A similar approach can be taken for **value ordering** decisions. Let $pc(a_i, a_j)$ be the deterministic value ordering heuristic used by SP-PCP. As previously noted, $pc(a_i, a_j) = a_i\{before\}a_j$ when $bd_{ij} > bd_{ji}$ and $a_j\{before\}a_i$ otherwise. Recalling the definition of bd, in cases where $S = \frac{min\{d(e_{a_i}, s_{a_j}), d(e_{a_j}, s_{a_i})\}}{max\{d(e_{a_i}, s_{a_j}), d(e_{a_j}, s_{a_i})\}}$ is ≈ 1, and hence bd_{ij} and bd_{ji} are $\approx$ equal, $pc(a_i, a_j)$ does not give clear guidance (both choices appear equally good). Accordingly, we define the following randomized version of *Choose-Prec-Constraint*:

$$rpc(a_i, a_j) = \begin{cases} \overline{pc(a_i, a_j)} & : \; U[0,1] + \alpha < S \\ pc(a_i, a_j) & : \; otherwise \end{cases}$$

where α represents a threshold parameter, $U[0,1]$ represents a random value in the interval $[0,1]$ with uniform distribution function and $\overline{pc(a_i, a_j)}$ is the complement of the choice advocated by $pc(a_i, a_j)$. Under this random selection method, it is simple to demonstrate that probability of deviating from the choice of SP-PCP's original value ordering heuristic pc is $(S - \alpha)$ when $S \geq \alpha$ and 0 otherwise. If α is set at 0.5, then each ordering choice can be seen to be equally likely in the case where $S = 1$ (i.e., the case where the heuristic gives the least information). Below, we examine performance with $\alpha = 0.5$ and $\alpha = 0.75$ (where there is a stronger bias toward deterministic behavior).

Heuristic-Biased Stochastic Sampling

An alternative to the "dynamic-bias" approach to variable and value ordering just described is HBSS, which can be viewed as a "static-bias" approach. HBSS operates by applying a given (deterministic) heuristic to first produce a ranking of alternative choices in any decision context, and then super-imposing a statically defined *bias function* over this ranking to define random selection probabilities.

Under this scheme, *Choose-Resolvable-Conflict* is defined to proceed in three steps:

1. Create a set C of possible choices by sorting all pending conflicts according to SP-PCP's $VarEval$ heuristic and selecting the first c conflicts.
2. Map the elements of C to the natural values $r = 1,2,3\ldots$ (rank), so that the conflict with minimum heuristic evaluation has rank $r = 1$.
3. Randomly select an element in C, where the probability $P(r)$ to get the choice with rank r is:

$$P(r) = \frac{Fb(r)}{\sum_{i=1}^{c} Fb(i)}$$

Following the experimental results obtained in (Bresina 1996), we assume that Fb is one of the following bias functions:[2]

1. Poly(2):
$$Fb(r) = \frac{1}{r^2}$$
2. Poly(3):
$$Fb(r) = \frac{1}{r^3}$$
3. Exponential:
$$Fb(r) = exp(-r)$$

We also assume that the set of elements C is of fixed size c and that heuristic information is used only to rank elements in C. In the experiments following, we have chosen $c = 20$, and, in fact, for $r = 20$ we have: $1/20^2 = 0.0025$; $1/20^3 = 0.000125$ and $exp(-20) = 2.2E-9$. So, for values of r greater than 20 the values of the functions Fb are in practice 0 anyway.

We define an analogous *Choose-Prec-Constraint* procedure in the following straightforward manner. The decision proposed by $pc(a_i, a_j)$ (SP-PCP's value ordering heuristic) is assigned the value $r = 1$, the complementary choice is assigned $r = 2$, and the probability of selecting the choice with rank r is defined as:

$$P(r) = \frac{Fb(r)}{Fb(1) + Fb(2)}$$

where Fb is one of the previous bias functions.

Experimental Evaluation

In this section, we evaluate the Random SP-PCP procedure together with the stochastic variable and value ordering heuristics developed above on a set of randomly generated job shop deadline scheduling problems. We contrast the performance of various configurations of Random SP-PCP with that of the original deterministic SP-PCP procedures described in (Cheng & Smith 1994), to assess the performance gain over both the simple one-pass partial solution procedure and the full procedure with chronological backtracking. We also examine the performance impact of restricting random decision making to variable ordering alone, value ordering alone or some combination of both. Finally, we compare the performance differences following from dynamic versus static specification of heuristic bias.

Following the same experimental design used in (Cheng & Smith 1994), we considered scheduling problems of size $N \times M$, where the structure of each problem instance is as follows. There are N jobs to be scheduled. Each job requires operations to be performed on each of M different resources, and the order in which each job must visit each resource is random. Using precisely the same problem generation scheme and parameters for specifying release dates, due dates, processing times and separation constraints as used in (Cheng & Smith 1994), we generated problem sets of 50 instances at each of three different sizes: 16×5, 20×5 and 25×5.

All procedures evaluated were implemented in Allegro CommonLisp, and (with one exception noted below) experiments were run on a SUN Sparc 10 workstation. In the case of deterministic SP-PCP, a limit of 2000 nodes was imposed as an upper bound on the amount of backtracking allowed in solving any one problem instance, and performance was measured in terms of number of problems solved and average solution time (over all instances). For the Random SP-PCP procedures tested, a maximum of 10 solution attempts (or starts) was allowed for any instance and the first attempt was always non-random (using the deterministic SP-PCP heuristics). In these experiments, performance was measured in terms of number of problems solved, average number of starts (considering only those problems that were solved), and average solution time.

Table 1 shows the performance results obtained on the smallest 16×5 problem set. Configurations of Random SP-PCP with the "dynamic-bias" search heuristics were tested with the two different settings mentioned earlier for both α (value ordering) and β (variable ordering) parameters, both in isolation and in combination. For the "static-bias", HBSS configurations of Random SP-PCP, three different bias functions were tested in conjunction with the random variable ordering, random value ordering and their combination. In both cases (static-bias and dynamic-bias), if a particular test configuration did not incorporate either random heuristic, then its deterministic SP-PCP counterpart was utilized.

We can see from Table 1 that 36 problems were solved by straight application of the deterministic heuristics of (Cheng & Smith 1994), leaving only 14 problem instances that require additional search for solution. Chronological backtracking solves an additional 12 problems within the search limits imposed (a

[2] In (Harvey 1994) a very similar approach was taken to variable and value ordering with poor results. However, this study restricted attention to very simple (i.e., constant) bias functions.

Table 1: Performance on 16×5 problem set.

Strategy	Parameters		Nbr. Solved	Nbr. of Starts (avg.)	CPU Time (sec.)
SP-PCP	1 pass		36	1.0	
	chron. bktrk		48	-	236
Random SP-PCP (Dynamic)	α	β			
	0.5	-	49	2.5	129
	0.75	-	49	3.2	146
	-	0.05	50	2.4	108
	-	0.1	49	2.5	120
	0.75	0.05	50	2.2	94
	0.75	0.1	50	3.2	129
	0.5	0.05	50	3.3	137
	0.5	0.1	49	3.1	146
Random SP-PCP (Static)	Bias Fn	Dec.			
	$\frac{1}{r^2}$	var	50	3.5	148
		val	40	5.5	275
		both	39	5.7	292
	$\frac{1}{r^3}$	var	50	2.7	114
		val	39	4.3	296
		both	43	5.8	258
	$exp(-r)$	var	50	2.8	120
		val	39	6.0	289
		both	39	6.6	301

total of 48). All configurations of Random SP-PCP with dynamic-bias heuristics outperform the backtracking procedure on this problem set; each solves at least one additional problem and all have smaller solution times. Of the Random SP-PCP configurations using static-bias heuristics, only the variable ordering heuristic used in isolation (i.e., with deterministic value ordering) yielded results better than fully deterministic backtracking search. However, this variable ordering heuristic performed uniformly well regardless of the bias function that was used. Contrasting the relative performance of dynamic-bias and static-bias approaches to value ordering gives some indication of the advantage of using the choice context (and the degree of information provided by the heuristic) to dynamically determine heuristic bias.

Table 2 displays results obtained on the intermediate sized, 20×5 problem set. All configurations of Random SP-PCP that solved all 50 instances in the 16×5 set were applied in this case, plus the top performing value ordering heuristic configuration of Random SP-PCP ($\alpha = 0.5$). For this problem set, 35 problems are solvable by the deterministic SP-PCP heuristics and chronological backtracking solved an additional 8 instances. At this larger problem size, the performance differential between Random SP-PCP and the backtracking search model widens; the slowest performing stochastic configuration solves 49 of 50 problems with an average solution time less than half that of the backtracking procedure. The strongest performing configuration on this problem set is dynamic-bias variable ordering only (with $\beta = 0.05$). The static-bias variable ordering configuration (with function $\frac{1}{r^3}$) performs quite well also, as do several combined dynamic-bias variable and value ordering configurations.

Performance results obtained for the largest 25×5 problem set are given in Table 3.[3] At this problem size, the relative effectiveness of stochastic variable and value ordering heuristics is most apparent. The deterministic (1 pass) procedure was able to solve only 23 of the 50 problem instances. Yet Random SP-PCP with dynamic-bias variable ordering ($\beta = 0.05$) solved all 50 with only slightly greater than 2 restarts on average, and all configurations of Random SP-PCP tested solve more than twice the number solved by the simple greedy procedure. As was the case at smaller problem sizes, dynamic-bias variable ordering continued to show some performance advantage over static-bias alternatives.

Concluding Remarks

In this paper, we have investigated the use of stochastic search as a means of efficiently solving scheduling problems with non-relaxable deadlines and complex metric constraints. Building from prior research which has developed strong variable and value ordering heuristics for this class of constraint satisfaction scheduling problems, we have focused on the design of stochastic counterparts to these heuristics, which are applicable within an iterative sampling search framework and offer a more cost-effective basis for extended search than does deterministic backtracking. The key idea underlying our approach is to heuristically bias random choices in a *dynamic* fashion, according to how well (or how poor) the available search heuristics discriminate among alternatives in any given choice context. We defined stochastic variants of the scheduling search heuristics developed in (Cheng & Smith 1994) which operate in this manner, and embedded them within an iterative sampling procedure called Random SP-PCP. In an experimental study on randomly generated scheduling problems of increasing scale, the stochastic procedure was found to significantly outperform its deterministic, backtracking-search counterpart.

Perhaps more interesting, comparison was also made to another, similar iterative sampling procedure based on the static-bias approach of HBSS (Bresina 1996). HBSS utilizes a *static*, pre-determined bias function each time the search heuristic is applied, and hence does not take into account any information about how strong (or weak) the heuristic's guidance is in specific choice contexts. Some configurations of this statically

[3] These largest experiments were run on an UltraSparc 170, which accounts for the apparent discrepancy with respect to the cpu times obtained on the smaller problem sets.

biased procedure were also found to perform quite well. Nonetheless, the best performing configuration of Random SP-PCP on all problem sets tested was one that utilized dynamic-bias search heuristics, indicating the benefit of dynamically adjusting heuristic bias to concentrate search around those scheduling decisions for which the heuristic does not provide strong advice.

Table 2: Performance on 20×5 problem set.

Strategy	Parameters		Nbr. Solved	Nbr. of Starts (avg.)	CPU Time (sec.)
SP-PCP	1 pass		35	1.0	
	chron. bktrk		43	-	605
Random SP-PCP (Dynamic)	α	β			
	0.5	-	49	3.1	285
	-	0.05	50	2.3	204
	0.75	0.05	50	2.6	224
	0.75	0.1	50	3.1	256
	0.5	0.05	50	3.4	272
Random SP-PCP (Static)	Bias Fn	Dec.			
	$\frac{1}{r^2}$	var	49	4.2	363
	$\frac{1}{r^3}$	var	50	2.5	216
	$exp(-r)$	var	49	3.2	299

Table 3: Performance on 25×5 problem set.

Strategy	Parameters		Nbr. Solved	Nbr. of Starts (avg.)	CPU Time (sec.)
SP-PCP	1 pass		23	1.0	
Random SP-PCP (Dynamic)	α	β			
	0.5	-	49	4.23	312
	-	0.05	50	3.26	235
	0.75	0.05	49	2.96	231
	0.75	0.1	49	3.66	290
	0.5	0.05	48	3.5	289
Random SP-PCP (Static)	Bias Fn	Dec.			
	$\frac{1}{r^2}$	var	47	4.1	356
	$\frac{1}{r^3}$	var	49	3.2	258
	$exp(-r)$	var	50	3.78	277

Acknowledgements

The work described in this paper was sponsored in part by the Advanced Research Projects Agency and Rome Laboratory, Air Force Material Command, USAF, under grant number F30602-95-1-0018 and the CMU Robotics Institute. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Advanced Research Projects Agency and Rome Laboratory or the U.S. Government.

Angelo Oddi's work was also supported by Agenzia Spaziale Italiana (ASI), CNR Committee 12 on Information Technology (Project SARI), CNR Committee 04 on Biology and Medicine, and Ministero dell'Universita' e della Ricerca Scientifica e Tecnologica (MURST). He is currently supported by a scholarship from CNR Committee 12 on Information Technology.

References

Bresina, J. 1996. Heuristic-biased stochastic sampling. In *Proceedings AAAI-96*.

Cheng, C., and Smith, S. 1994. Generating feasible schedules under complex metric constraints. In *Proceedings AAAI-94*.

Cheng, C., and Smith, S. 1996. A constraint satisfaction approach to makespan scheduling. In *Proceedings 4th Int. Conf. on AI Planning Systems*.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61–95.

Harvey, W., and Ginsberg, M. 1995. Least discrepancy search. In *Proceedings IJCAI-95*.

Harvey, W. 1994. Search and job shop scheduling. Technical Report CIRL TR 94-1, Computational Intelligence Research Laboratory, University of Oregon, Eugene, Oregon.

Langley, P. 1992. Systematic and non-systematic search strategies. In *Proceedings 1st Int. Conf. on AI Planning Systems*.

Smith, S., and Cheng, C. 1993. Slack-based heuristics for constraint satisfaction scheduling. In *Proceedings AAAI-93*.