Journal of Heuristics, 11: 5–34, 2005

Enhancing Stochastic Search Performance by Value-Biased Randomization of Heuristics

VINCENT A. CICIRELLO*
Department of Computer Science, Drexel University, 3141 Chestnut Street, Philadelphia, PA 19104
email: cicirello@cs.drexel.edu

STEPHEN F. SMITH
The Robotics Insitute, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213
email: sfs@cs.cmu.edu

Submitted in October 2003 and accepted by David Woodruff in November 2004 after 1 revision

Abstract

This paper investigates the utility of introducing randomization as a means of boosting the performance of search heuristics. We introduce a particular approach to randomization, called Value-biased stochastic sampling (VBSS), which emphasizes the use of heuristic value in determining stochastic bias. We offer an empirical study of the performance of value-biased and rank-biased approaches to randomizing search heuristics. We also consider the use of these stochastic sampling techniques in conjunction with local hill-climbing. Finally, we contrast the performance of stochastic sampling search with more systematic search procedures as a means of amplifying the performance of search heuristics.

Key Words: stochastic sampling, stochastic search, randomized heuristics, combinatorial optimization, weighted tardiness scheduling, sequence-dependent setups

1. Introduction

In many combinatorial optimization domains, simpler stochastic algorithms often exhibit superior performance as compared to more carefully refined and highly customized approaches for the problem at hand. Stochastic search algorithms are often robust, scalable problem solvers. Examples of the success of such stochastic search algorithms abound: Space Telescope Scheduling (Bresina, 1996), Project Scheduling (Cesta, Oddi, and Smith, 1999, 2002), Solid Model Similarity Assessment (Cicirello, 1999; Cicirello and Regli, 1999, 2001, 2002), Satisfiability (Selman, Kautz, and Cohen, 1996; Prestwich, 2001; Boyan and Moore, 1998), VLSI Channel Routing (Boyan and Moore, 1997,1998), Graph Coloring (Prestwich, 2001), Constraint Satisfaction (Freuder et al., 1995), and N-Queens (Prestwich, 2001)—just to mention a few.

*Corresponding Author

The particular class of stochastic search algorithm that we focus on in this paper is that of stochastic sampling. In a stochastic sampling algorithm, the search iteratively probes from the root of the search-space down to a terminal node, maintaining no history of the search paths taken on previous probes. Each decision in such a framework is made stochastically. The most basic algorithm in this class makes unbiased random decisions and is sometimes referred to as *Iterative Sampling* (Langley, 1992). Of greater interest to us are stochastic sampling algorithms that bias random decisions using some domain heuristic. One example is that of heuristic equivalency (Gomes, Selman, and Crato, 1997, 1998; Gomes et al., 2000) in which all choices with heuristic value within some percentage of the heuristic's preferred choice are considered equivalent; and in which the decision is made unbiased at random from among this set of "heuristically equivalent" choices. Heuristic equivalency has also been referred to by the term *acceptance bands* (Oddi and Smith, 1997; Cesta, Oddi, and Smith, 1999, 2002). Another more general framework is Bresina's *heuristic biased stochastic sampling* (HBSS) framework (Bresina, 1996). In HBSS, decisions are made at random, but biased according to a function of a ranking determined by an ordering of the choices according to the heuristic.

Stochastic sampling can often be an effective method for amplifying heuristic preformance. If we assume that we have a strong domain heuristic, then iteratively randomizing that heuristic can be an effective search technique. For example, for many scheduling problems, a great deal of effort has been made by researchers in the area of operations research into the development of strong domain heuristics (see Morton and Pentico, 1993 for a survey of heuristic scheduling systems). These heuristics, when applied deterministically, can provide near-optimal results in some circumstances. But, as should be expected when dealing with a heuristic method, they are not infallible. Through randomization of a strong domain heuristic, we can expand the search to a stochastic neighborhood of the heuristic's prescribed solution.

This paper investigates the utility of introducing randomization as a means of boosting the performance of search heuristics. We introduce a particular approach to randomization, called Value-biased stochastic sampling (VBSS), which emphasizes the use of heuristic value in determining stochastic bias. We offer an empirical study of the performance of value-biased and rank-biased approaches to randomizing search heuristics. We also consider the use of these stochastic sampling techniques in conjunction with local hill-climbing. Finally, we contrast the performance of stochastic sampling search with more systematic search procedures as a means of amplifying the performance of search heuristics. The experimental evaluations of this paper consider the difficult combinatorial optimization problem known as *Weighted Tardiness Sequencing with Sequence-Dependent Setups*.

The remainder of this paper is organized as follows. Section 2 defines the iterative sampling algorithm and Section 3 defines Bresina's HBSS algorithm. Next, in Section 4, we present our value-biased variation of the HBSS algorithm that we call VBSS. In Section 5, we discuss the issue of choosing a bias function within the VBSS and HBSS frameworks. In Section 6, we describe the problem domain we use to evaluate the approach. Results are presented in Sections 7–9. We conclude the paper in Section 10.

Algorithm 1: Iterative Sampling (IS)

Input: Number of iterations I; an "objective" function; and a search-tree T.
Output: A solution S.

```
IS(I, objective, T)
(1)     bestsofar ← nil
(2)     repeat I times
(3)         S ← root search-node of T
(4)         while S is a decision node of T
(5)             select uniformly at random choice C from S
(6)                 S ← Successor(S,C)
(7)         if AllConstraintsSatisfied(S) and objective(S) is superior to
            objective(bestsofar)
(8)             bestsofar ← S
(9)     return bestsofar
```

2. Iterative sampling

Iterative Sampling (Langley, 1992) is a simple algorithm that begins at the root of the search tree and chooses a branch to follow from the root at random. From this successor node in the search space, it again chooses randomly a branch to follow, and so on and so forth until it either finds a solution or hits a dead-end. If the latter, then it begins again at the top of the search tree and iterates the process. If the former, then if we are simply looking for *some* solution we are finished. Otherwise, if we are looking for the best solution we can find, then the process iterates some number of times until we are satisfied with the quality of the best solution found. It is a rather simple, easy to implement algorithm; but it is naive and does not consider any search/state information, nor does it consider any existing heuristics for the problem. Its ability to find solutions relies entirely on the assumption that there may exist many solutions in the search-space. Furthermore, its ability to find "good" solutions in an optimization context relies entirely on the assumption that there exists a high density of such "good" solutions. In most domains of practical interest, these assumptions tend to be overly optimistic, although some promising results have been obtained for a flow-shop scheduling problem (Watson et al., 1999).[1,2] Algorithm 1 shows the iterative sampling algorithm for a generic search-tree.

3. Heuristic-biased stochastic sampling

Bresina's HBSS (Bresina, 1996) framework provides a general basis for amplifying heuristic performance through randomization. HBSS operates within a global search paradigm, where partial solutions are extended by adding one new decision at each step of the search. Like iterative sampling, a random choice process is invoked to make each decision; but unlike iterative sampling, this process is biased according to a pre-specified heuristic for the problem at hand. Specifically, the heuristic is used to first prioritize the alternatives

that remain feasible at a given decision point, and then a bias function is superimposed over this ranking to stochastically select from this ranked set. Once a complete solution is generated, it is evaluated according to the global optimization criteria. The search process is then repeated some number of times and the best solution generated is taken as the final result.

The specific problem considered by Bresina was telescope observation scheduling. Given a set of potential observation tasks and an objective criterion (e.g., maximize viewing time), the problem is to produce a schedule (i.e., a sequence of tasks) for execution during the next period. Formulated within HBSS, generation of a schedule proceeds in a forward dispatching manner, by repeatedly ranking the subset of tasks that remain "unscheduled", and then choosing the next task to append to the current (partial) schedule.[3] This process iterates until either all potential tasks have been scheduled or the time frame has been exhausted.[4] The resulting schedule is then evaluated globally and the search process is restarted. The HBSS algorithm is illustrated in a general search context in Algorithm 2.

The ability to use different bias functions within HBSS provides a means of placing more or less emphasis on following the advice of the base heuristic. A number of polynomial bias functions of the form r^{-n} and an exponential bias function of the form e^{-r}, are proposed and explored by Bresina, where r is the rank of the choice in question (Bresina, 1996). As pointed out by Bresina, the choice of bias function can and should be made based on overall confidence in the base heuristic. If the heuristic is deemed strong, then it makes sense to follow it more often; if the heuristic is weak, then a more disruptive bias is called for. Heuristics are typically more or less informed in different decision contexts, but the rank-biased approach of HBSS does not provide a means of calibrating the degree of randomness according to this dynamic aspect of problem solving state. This capability requires movement away from an approach to random bias based strictly on rank order and toward an approach based on heuristic valuations.

4. Value-biased stochastic sampling

A basic flaw of the HBSS framework of Bresina is that it ignores the discriminatory power inherent in the heuristic. Its stochastic decisions rank-order the choices from the choice with the highest value of the heuristic to the choice with the lowest value of the heuristic. It then chooses choice c_i according to a roulette wheel with probability:

$$P(c_i) = \frac{\text{bias}(\text{rank}(c_i, \text{heuristic}))}{\sum_j \text{bias}(\text{rank}(c_j, \text{heuristic}))} \tag{1}$$

where bias( ) is a bias function and rank( ) returns the rank assigned to a given choice when the set of choices is sorted by the given heuristic. Other than to sort the choices, the heuristic values are not used and thus not utilized to their full potential. There are, however, occasions when a rank-biased approach may be more beneficial. For example, genetic algorithms sometimes benefit from a rank-biased selection strategy (Goldberg, 1989). If the heuristic values of alternatives are all very close, then a rank-biased approach can provide a better ability to discriminate among choices.

Algorithm 2: Heuristic-Biased Stochastic Sampling (HBSS)

Input: Number of iterations I; a "heuristic" function; a "bias" function; an "objective" function; and a search-tree T.
Output: A solution S.

```
HBSS(I, heuristic, bias, objective, T)
(1)     bestsofar ← solution S obtained if "heuristic" is followed from
        T
(2)     repeat I times
(3)         S ← root search-node of T
(4)         while S is a decision node of T
(5)             foreach choice C from S
(6)                 score[C] ← heuristic(C, S)
(7)             sort all choices C according to score[C]
(8)             totalweight ← 0
(9)             foreach choice C from S
(10)                rank[C] ← sort position of C
(11)                weight[C] ← bias(rank[C])
(12)                totalweight ← totalweight + weight[C]
(13)            foreach choice C from S
(14)                prob[C] ← weight[C] / totalweight
(15)            select randomly the choice C biased according to prob[C]
(16)                S ← Successor(S, C)
(17)        if AllConstraintsSatisfied(S) and objective(S) is superior to
            objective(bestsofar)
(18)            bestsofar ← S
(19)    return bestsofar
```

In order to fully utilize the discriminatory power of the heuristic inherent in the heuristic values, we now define what we call value-biased stochastic sampling (VBSS).[5] In VBSS, decisions are again made in a manner analogous to a roulette wheel, but choice c_i is now made with probability:

$$P(c_i) = \frac{\text{bias}(\text{heuristic}(c_i))}{\sum_j \text{bias}(\text{heuristic}(c_j))} \tag{2}$$

VBSS is shown in Algorithm 3.

Consider one decision context in which you have two choices that are preferred almost equivalently by the heuristic (i.e., they have almost, but not quite, equal heuristic values). Now consider a second decision context in which you have two choices, but where one of the choices is much more strongly preferred (i.e., it has a much higher heuristic value than the other choice). In HBSS, both of these decision contexts are regarded equivalently. That is, the choice with the higher heuristic value gets ranked first, the other choice gets ranked second, the bias function is applied, and the decision is made. The first ranked choice has the

	Context 1 Choice 1 (H1 = 10) / Choice 2 (H2 = 11)	Context 2 Choice 1 (H1 = 10) / Choice 2 (H2 = 100)
HBSS with bias(p) = 1/p	Rank(c1) = 2 Rank(c2) = 1 Bias(Rank(c1)) = 0.5 Bias(Rank(c2)) = 1 P(choosing c2) = 1/(1+0.5) = 0.67	Rank(c1) = 2 Rank(c2) = 1 Bias(Rank(c1)) = 0.5 Bias(Rank(c2)) = 1 P(choosing c2) = 1/(1+0.5) = 0.67
VBSS with bias(p) = p	Bias(H1) = 10 Bias(H2) = 11 P(choosing c2) = 11/(11+10) = 0.52	Bias(H1) = 10 Bias(H2) = 100 P(choosing c2) = 100/(100+10) = 0.91

Figure 1. Two example decision contexts—one more discriminating than the other.

same probability of being made in both contexts. In VBSS, alternatively, since the heuristic values are used explicitly in the stochastic decisions rather than a rank imposed by them, the first ranked choice in the more discriminating context is chosen with a much higher probability than it is in the less discriminating context. This can be seen in figure 1 where we see two example decision contexts: one in which the heuristic values are almost the same and a second in which the heuristic values are drastically different. In this second decision context, using a value-biased approach, there is a much higher probability of choosing the heuristic preferred choice as compared to the other decision context.

Computationally, the VBSS algorithm selects a choice in $O(C)$ time where there are C choices from the current search-node. Since it must do this C times (in a sequencing problem), the core sampling procedure has an overall algorithmic complexity of $O(C^2)$. In fact, the complexity of a corresponding deterministic heuristic sequencing procedure is also $O(C^2)$. Hence, VBSS adds only a constant factor to the computational time required for strict deterministic application of the heuristic. If we compare this complexity to that of the HBSS algorithm of Algorithm 2, we can see that VBSS can be asymptotically more efficient. Since HBSS biases its stochastic decisions according to a rank-ordering of the choices, the obvious implementation sorts the choices according to their heuristic values each time a search decision is to be made—an $O(C \log C)$ operation. And with C decisions to be made (in a sequencing problem), the algorithmic complexity of HBSS is $O(C^2 \log C)$.[6]

5. Choosing a bias function

With VBSS (as well as with HBSS), it is necessary to specify a bias function that is applied to the heuristic values (or in the case of HBSS to the ranks). This bias function can be viewed as a system parameter that requires some amount of tuning. There are a couple of issues to consider when selecting a bias function for VBSS[7]:

Algorithm 3: Value Biased Stochastic Sampling (VBSS)

Input: Number of iterations I; a "heuristic" function; a "bias" function; an "objective" function; and a search-tree T.
Output: A solution S.

```
VBSS(I, heuristic, bias, objective, T)
(1)     bestsofar ← solution S obtained if "heuristic" is followed from
        T
(2)     repeat I times
(3)         S ← root search-node of T
(4)         while S is a decision node of T
(5)             foreach choice C from S
(6)                 score[C] ← heuristic(C, S)
(7)             totalweight ← 0
(8)             foreach choice C from S
(9)                 weight[C] ← bias(score[C])
(10)                totalweight ← totalweight + weight[C]
(11)            foreach choice C from S
(12)                prob[C] ← weight[C] / totalweight
(13)            select randomly the choice C biased according to prob[C]
(14)                S ← Successor(S, C)
(15)        if AllConstraintsSatisfied(S) and objective(S) is superior to
            objective(bestsofar)
(16)            bestsofar ← S
(17)    return bestsofar
```

– First, stronger and/or knowledge-rich heuristics should probably be used in conjunction with stronger bias functions, such as polynomials of high degree. The rationale is that if a large amount of effort and expert knowledge went into the heuristic's design, then the search might benefit by following its advice more often. Similarly, weaker and/or knowledge-poor heuristics should probably be used with weaker bias functions, such as low degree polynomials.
– Second, heuristics that generally give values in some very small range, such as very small positive real values less than one, might require a strong bias function to spread out the values used in the roulette wheel decisions. Heuristics that give values in a wider range might already be sufficiently spread to require only a weaker bias function.

With these issues in mind, little (if any) fine-tuning of the bias function choice is generally needed. It is generally sufficient to try out a few in some range decided upon by considering the types of values given by the heuristic and the strength of the heuristic itself. Alternatively, one might employ a more computationally heavy approach to tuning the bias function such as with a meta-optimization approach as is sometimes done to tune genetic algorithms (see Cao and Wu, 1999; Eiben, Hinterding, and Michalewicz, 1999; Cicirello and Smith, 2000 for examples). In the experiments of this paper, no such computational method has

been employed. A handful of bias functions have been tried on a small set of problem instances for a small number of iterations. The bias functions tried in these preliminary tuning runs are chosen according to the rationale presented above, and in most cases give nearly numerically indistinguishable results.

6. Problem domain: weighted tardiness sequencing with sequence-dependent setups

To demonstrate and experimentally evaluate the VBSS framework, we turn to the domain of factory scheduling and the fairly broad body of work in the area of dispatch scheduling heuristics (see Morton and Pentico, 1993). Specifically, we explore the utility of stochastic sampling in a difficult combinatorial optimization domain, and use the domain as a basis for validating the VBSS approach. The scheduling problem tackled is that of the weighted tardiness scheduling problem with sequence-dependent setups and is formalized in Section 6.1. Existing methods for the problem from the literature are presented in Section 6.2. The search space is discussed in Section 6.3. The description of the specific set of problem instances and how they were generated can be found in Section 6.4. Performance criteria that are used in the experimental comparisons are defined in Section 6.5.

6.1. Problem formalization

The Weighted Tardiness Scheduling Problem with Sequence-Dependent Setups is a sequencing problem encountered in a number of real-world problem applications (e.g., turbine component manufacturing (Chiang, Fox, and Ow, 1990), the packaging industry (Adler et al., 1993), among others (Morton and Pentico, 1993)). Specifically, we are given a set of jobs $J = \{j_0, j_1, \ldots, j_N\}$. Each of the jobs j has a weight w_j, duedate d_j, and process time p_j. Furthermore, $s_{i,j}$ is defined as the amount of setup time required immediately prior to the start of processing for job j if it is to follow job i on the machine. It is not necessarily the case that $s_{i,j} = s_{j,i}$. The 0-th job is a non-physical job representing the starting point of the problem ($p_0 = 0, d_0 = 0, s_{i,0} = 0, w_0 = 0$). Its only purpose is to allow for the specification of the setup time of each of the jobs if sequenced in position 1. The sequence-dependent nature of the setup times is a primary source of problem difficulty. The particular version of the problem that we concern ourselves with here is the case where the N jobs must be sequenced on a single machine and where preemption of a job during setup or processing is not permitted. Furthermore, if a machine is processing or setting up, then it cannot do anything else until that operation is completed.

The objective of this problem is to sequence the set of jobs J on a machine to minimize the total weighted tardiness:

$$T = \sum_{j \in J} w_j T_j = \sum_{j \in J} w_j \max(c_j - d_j, 0), \tag{3}$$

where T_j is the tardiness of job j; and c_j, d_j is the completion time and duedate of job j. The completion time of job j is equal to the sum of the process times and setup times of all

jobs that come before it in the sequence plus the setup time and process time of job j itself. Specifically, let $\pi(j)$ be the position in the sequence of job j. We can now define c_j as:

$$c_j = \sum_{i,k \in J, \pi(i) <= \pi(j), \pi(i) = \pi(k)+1} p_i + s_{k,i}. \tag{4}$$

The weighted tardiness scheduling problem with sequence-dependent setups is NP-hard in the strong sense.

6.2. State-of-the-(ad-hoc)-art solution methods

Most of the work that has been done on the weighted tardiness scheduling problem has been for versions of the problem without sequence-dependent setup times. For the setup-free problem, there exist libraries of benchmark instances and a large number of algorithms with which to compare new approaches.

Unfortunately, although sequence-dependent setups commonly appear in real-world scheduling problems (e.g., Adler et al., 1993; Morley and Schelberg, 1993; Morley, 1996), they are often ignored during the development of algorithms and solution procedures. It is common practice to assume that setup time can be reintroduced to the problem after solving the setup-free version. For example, the dispatch heuristics for the sequence-dependent setup case that we will discuss below are variations of a well-known and state-of-the-art heuristic for the setup-free version of the problem. Terms including setup time have been added to the heuristic in a seemingly ad hoc manner. Yet, at the same time, these ad hoc heuristic procedures (as well as local improvement search algorithms applied to their results) seem about the best one can do for the sequence-dependent setup version of the problem given the current state-of-the-art. For example, complete, guaranteed optimal algorithms are currently limited to solving problems of at most 20–25 jobs in size (Sen and Bagchi, 1996), necessitating the use of heuristic-guided search, or meta-heuristic approaches for larger problems.

In the following subsections, we overview the existing dispatch policies for the sequence-dependent setup, weighted tardiness scheduling problem as well as a local hill-climber designed specifically with the intention of improving upon the solutions produced by one of these heuristics.

6.2.1. Dispatch scheduling policies. In dynamic factory environments, dispatch scheduling heuristics provide a practical, robust basis for managing execution in environments that are often plagued by a large number of dynamic characteristics that can be difficult to accurately model (e.g., McKay provides a model of such a complex dynamic factory environment (McKay, 1993)). Scheduling decisions such as which job to assign to a machine next are made in an online manner only as needed, based on the current state of the factory. Dispatch heuristics make use of information about jobs such as expected processing time, setup time, due date, priority, etc., and are typically designed to optimize a given performance objective. Their virtue is their simplicity and insensitivity to environmental dynamics and for these reasons they are commonly employed. At the same time, the

localized and myopic nature by which decisions are made under such schemes make them inherently susceptible to sub-optimal decision-making and they can even exhibit formally chaotic tendencies (Kempf and Beaumariage, 1994; Beaumariage and Kempf, 1995).

For the problem domain of weighted tardiness scheduling with sequence-dependent setups, the *Apparent Tardiness Cost with Setups (ATCS)* dispatch heuristic (Lee et al., 1997) is perhaps the strongest heuristic available. ATCS builds on earlier research into the weighted tardiness problem and is arguably the current best performing dispatch policy for this class of scheduling problem. ATCS is defined as follows:

$$\text{ATCS}_j(t,l) = \frac{w_j}{p_j} \exp\left(- \frac{\max(d_j - p_j - t, 0)}{k_1 \bar{p}} - \frac{s_{l,j}}{k_2 \bar{s}} \right), \tag{5}$$

where t is the current time (or the sum of the process and setup times of the already sequenced jobs); l is the index of the job just completed (or the last job added to the schedule); $\bar{p}$ is the average processing time of all jobs; $\bar{s}$ is the average setup time. The k_1 and k_2 are parameters for tuning the heuristic. Lee et al. tunes these parameters according to problem instance characteristics (Lee et al., 1997), specifically:

$$k_1 = \begin{cases} 4.5 + R & \text{if } R \leq 0.5 \\ 6.0 - 2R & \text{otherwise} \end{cases}, \tag{6}$$

and

$$k_2 = \frac{\tau}{2\sqrt{\eta}}. \tag{7}$$

R is the due-date range factor; τ is the due-date tightness factor; and η is the setup time severity factor. These parameters are defined as follows:

$$\tau = 1 - \frac{\bar{d}}{C_{\max}} \tag{8}$$

$$R = \frac{d_{\max} - d_{\min}}{C_{\max}} \tag{9}$$

$$\eta = \frac{\bar{s}}{\bar{p}} \tag{10}$$

where $\bar{d}$, $\bar{p}$, and $\bar{s}$ are the average duedate, average process time, and average setup time, $d_{\max}$, $d_{\min}$ are the maximum and minimum duedates, and $C_{\max}$ is the makespan (or completion time of the last job). Given that the makespan depends on the sequence and the $s_{i,j}$, the estimator suggested by Lee et al. is used: $\tilde{C}_{\max} = n(\bar{p} + \beta\bar{s})$ where n is the number of jobs in the problem instance. The next job j added to the schedule using the ATCS heuristic is simply:

$$j = \arg\max_j \text{ATCS}_j(t,l). \tag{11}$$

Before the ATCS heuristic's presentation by Lee et al., the best performing dispatch heuristic for this problem had been that of Raman, Rachamadugu, and Talbot (1989):

$$\text{Raman}_j(t, l) = \frac{w_j}{p_j + s_{l,j}} \exp\left(- \frac{\max(d_j - p_j - s_{l,j} - t, 0)}{k\bar{p}} \right) \tag{12}$$

where the next job j is chosen according to:

$$j = \arg\max_j \text{Raman}_j(t, l). \tag{13}$$

The k is a parameter that again requires tuning. In their comparison of ATCS and Raman, Lee et al. suggest setting $k = 5.5 - \tau - R + \eta$.

One thing that should be noted is that both Raman et al.'s dispatch policy as well as the ATCS dispatch policy are modifications of the R&M dispatch policy (Rachamadugu and Morton, 1982) that was developed for the variation of the problem without sequence-dependent setup constraints:

$$\text{R\&M}_j(t) = \frac{w_j}{p_j} \exp\left(- \frac{\max(d_j - p_j - t, 0)}{k\bar{p}} \right). \tag{14}$$

6.2.2. Dispatch policy as starting configuration for local search. In Lee et al.'s original paper describing the ATCS heuristic, they also present a local hill-climber designed specifically with the intentions of being applied to the solution that results directly from the deterministic application of the dispatch policy ATCS (Lee et al., 1997). The hill-climber (that we will refer to later in our experimental study as "LEE") assumes that the starting configuration is a good one (i.e., in the vicinity of the optimal or a near-optimal solution). Given this assumption, Lee et al.'s hill-climber uses a fairly small operator set. The operator set includes two types of local moves:

1. Swaps: Choose job j' that adds the most to the total weighted tardiness objective (i.e., $j' = \arg\max_j w_j \max(c_j - d_j, 0)$). Next, consider swapping job j' with each of the 20 nearest jobs in the current sequence.
2. Insertions: Choose job j' that adds the most to the total weighted tardiness objective (i.e., $j' = \arg\max_j w_j \max(c_j - d_j, 0)$). Next, consider the removal of job j' followed by the insertion of job j' before each of the 20 nearest jobs in the current sequence.

Two things to note about this operator set is that it does not consider moving jobs large distances and that one of the jobs is fixed to be the job which adds the most to the objective function value. These fall out from the assumption of having a good initial solution. It is assumed that jobs are near their optimal location in the sequence (i.e., moving jobs large distances in the sequence is not considered) and it is assumed that most jobs do not need to be moved at all (i.e., fixing one of the jobs to be the one that adds the most to the objective function). Given this operator set, the hill-climber at each step makes the move that gives

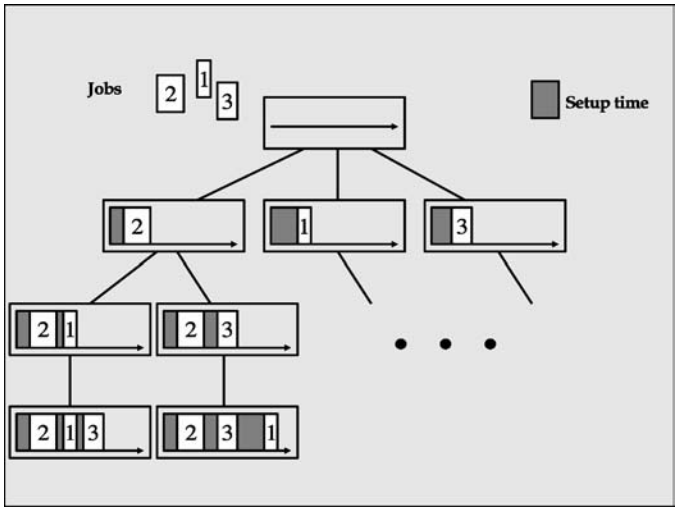

Figure 2. Illustration of the search space for the weighted tardiness scheduling problem. Particularly note the sequence-dependent size of the setup times that are indicated by the size of the gray boxes in the figure.

the greatest improvement in the objective function's evaluation and ends the search when no further improvement can be made given this operator set. The ATCS dispatch policy coupled with the use of this hill-climber is currently in operation within the scheduling system of a number of factories in the packaging industry (Adler et al., 1993).

6.3. The search-space

The search-space that we will explore in the experiments in the remainder of this paper can be viewed as a tree. The root node of this tree represents an empty sequence (or empty schedule). Each search-node has a child for each of the jobs not yet in the sequence. Such a child of a search-node represents adding the given job to the end of the sequence. If a problem instance has N jobs, then the root node has N children (i.e., each represents beginning the sequence with one of the N jobs). Each of the children of the root node has $N-1$ children, and so forth and so on. Each leaf node of the tree represents one of the $N!$ possible sequences of jobs. Every leaf node represents a feasible solution and thus every path from the root is guaranteed to find a feasible solution. An illustration of this search-space is shown in figure 2. Particularly note that the size of the setup times (indicated by the gray boxes) is dependent upon the sequence of jobs.

6.4. The problem set

The problem instances that we consider are generated according to a procedure described by Lee et al. (1997) and used in the analysis of their dispatch scheduling policy ATCS.

Unfortunately, they did not make their actual problem set available so we are unable to compare to the exact problem instances that were used in their study. Our problem set is however generated as they prescribe. We have made these problem instances available on the Internet (Cicirello, 2003). Appendix 10 provides the location, details of the file format, and best known solutions.

Each problem instance is characterized by three parameters: the due-date tightness factor τ; the due-date range factor R; and the setup time severity factor η. We specifically consider problem sets characterized by the following parameter values: $\tau = \{0.3, 0.6, 0.9\}$; $R = \{0.25, 0.75\}$; and $\eta = \{0.25, 0.75\}$. For each of the twelve combinations of parameter values, we generate 10 problem instances with 60 jobs each. Generally speaking, these 12 problem sets cover a spectrum from loosely to tightly constrained problem instances. The processing times are uniformly distributed over the interval [50, 150], with $\bar{p} = 100$. The mean setup time $\bar{s}$ is then determined from η and the setup times are uniformly distributed in the interval $[0, 2\bar{s}]$. The duedate of a job is uniformly distributed over $[\bar{d}(1 - R), \bar{d}]$ with probability τ and uniformly distributed over $[\bar{d}, \bar{d} + (C_{\max} - \bar{d})R]$ with probability $1 - \tau$. The weights of the jobs are distributed uniformly over [0, 10].

6.5. *Performance criteria*

In the experimental results that follow, we use the following performance criteria:

- Average Percent Improvement (API) over the solution given by use of the deterministic heuristic policy. Percent improvement is defined as: $100 * \frac{h-a}{h}$ where h is the objective value for the problem instance given by the deterministic use of the dispatch heuristic ATCS, and a is the objective value given by the search algorithm. This is then averaged across all 120 problem instances to get API. For all stochastic algorithms considered, the API given is also averaged across the results of 10 independent runs on each problem instance. 95% confidence intervals are shown.

  The ATCS dispatch policy is currently the best known for this problem. Lee et al.'s hill-climber which uses ATCS to determine the initial starting configuration is currently the best known heuristic search algorithm for this problem and Lee et al. report their results as the API over ATCS. Ideally, we would like to have a set of problems for which the optimal solutions are known, for which we have bounds, or for which we at least have the current best known solutions. Unfortunately, for the sequence-dependent setup problem, there is not such a benchmark set of problems in existence currently. For this reason, we present comparative results using the criteria set forth in the original discussion of the current best known algorithm for the problem.
- Average CPU Time (TIME) in seconds on a Sun Ultra 10, 300 MHz.
- The average number of search nodes generated (Gen) during the search.[8]
- The average number of search nodes visited (Visits) during the search.
- The average number of solution nodes considered (Sols) during the search.

7. To value-bias or to rank-bias?

Earlier, we saw an example of two decision contexts in which a value-bias approach made better use of the discriminatory power inherent in the heuristic values as compared to a rank-bias approach to stochastic sampling. Here now, we compare VBSS and HBSS experimentally on the weighted tardiness scheduling problem with sequence-dependent setups. The goal is to see how much can be gained by using the heuristic values for bias rather than biasing by rank-order alone.

In order to properly compare HBSS and VBSS on the problem set, we begin by applying each of them with a wide range of bias functions and the ATCS dispatch heuristic presented earlier. Table 1 shows the results of the HBSS algorithm for polynomial bias functions of degrees 1 through 8 (bias $= \frac{1}{\text{rank}^p}$ where p is the degree of the polynomial). Similarly, Table 2

Table 1. HBSS Preliminary Results: A sampling of the results from applying HBSS with various bias functions and 100 iterations. This is a snapshot of the data that was used to choose a bias function for the HBSS algorithm for further experiments.

p	API
1	0.0
2	4.6
3	18.3
4	21.5
5	21.6
6	20.8
7	19.7
8	17.9

Table 2. VBSS Preliminary Results: A sampling of the results from applying VBSS with various bias functions and 100 iterations. This is a snapshot of the data that was used to choose a bias function for the VBSS algorithm for further experiments.

p	API
1	7.3
2	16.7
3	20.0
4	21.6
5	22.7
6	22.9
7	23.2
8	23.1

Table 3. VBSS (with $p = 5$) vs HBSS (with $p = 5$) for various numbers of iterations. ATCS is the deterministic heuristic result. LEE is the hill-climber (non-randomized) of Lee et al.

Algorithm	API	TIME	Gen	Visits	Sols
ATCS	0.0	0.009	1,830	60	1
LEE	12.4 ± 0.15	0.012	1,909	61	2
VBSS (1)	5.0 ± 0.07	0.021	3,660	120	2
HBSS (1)	4.7 ± 0.06	0.159	3,660	120	2
VBSS (10)	16.4 ± 0.15	0.123	20,130	660	11
HBSS (10)	15.5 ± 0.14	1.585	20,130	660	11
VBSS (100)	22.7 ± 0.17	1.122	184,830	6,060	101
HBSS (100)	21.6 ± 0.17	15.467	184,830	6,060	101
VBSS (200)	23.8 ± 0.18	2.234	367,830	12,060	201
HBSS (200)	22.6 ± 0.17	30.932	367,830	12,060	201

shows the results of VBSS with polynomial bias functions (bias = value^p where p is the degree of the polynomial and value is the heuristic value). Many more bias functions were also considered but for brevity have been excluded from these tables. The values in the tables are averages across all problem instances and for 10 runs each.

For the HBSS algorithm, the degree 5 polynomial gives highest average percent improvement over the deterministic heuristic solution (API), though is not statistically significant as compared to either the degree 4 or 6 polynomials. All further comparisons involving HBSS will use the results given by the degree 5 polynomial bias function. For the VBSS results, the best bias function is even less clear. No statistical significance was found among polynomial bias functions of degrees 4 through 8. Further comparisons will use the results given by the degree 5 polynomial bias function.

We now turn to a direct comparison of VBSS and HBSS in Table 3. We compare the algorithms with several numbers of iterations. We also compare to the deterministic heuristic solution listed in the row labeled ATCS and to the hill-climber proposed by Lee et al. (1997) applied to the result of the deterministic heuristic solution listed in the table as LEE. First note that all rows in the table should necessarily do at least as well as ATCS. This is due to the fact that both HBSS and VBSS, as implemented, first compute the ATCS solution; and LEE hill-climbs on the ATCS solution. We can now make the following observations:

– Both VBSS and HBSS begin outperforming LEE when 10 or more iterations are computed. But, for a single iteration of either HBSS or VBSS, the solutions are not as good as that of LEE and require more time than LEE. Therefore, HBSS and VBSS both appear useful for improving upon solutions provided the CPU time is available. If you desire a good solution almost instantaneously, then LEE might be the better choice. Yet in slightly more than 2 seconds using 200 iterations of VBSS, you can achieve almost twice the API as compared to the API of LEE.

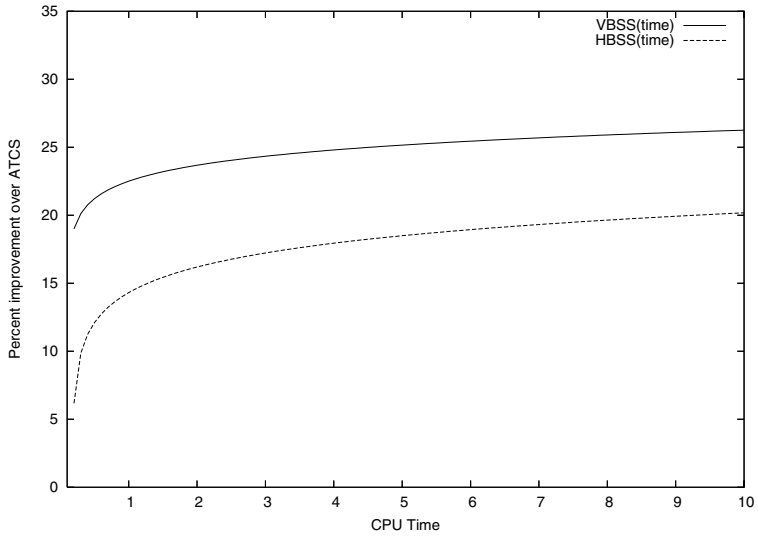

Figure 3. Comparison of the average percent improvement over the heuristic solution given by VBSS and HBSS as a function of CPU time (seconds). VBSS's performance is shown by the solid line. HBSS's performance is shown by the dashed line.

– Consider any number of iterations (1, 10, 100, or 200) and note that VBSS using the heuristic values as bias gives better results than HBSS using rank bias with that same number of iterations.
– Upon examining CPU Times, it should be noted that for these 60 job problems, VBSS exhibits a greater than 10-fold speedup over HBSS. This is due to the need of HBSS to sort the choices according to the heuristic.[9] Given this, one can compare 100 iterations of VBSS to 10 iterations of HBSS; and similarly 10 iterations of VBSS to 1 iteration of HBSS. With this in mind, you can clearly find significantly better results by value-biasing in a fraction of the time required to rank-bias the stochastic search. The plot in figure 3 illustrates the trend over time. The performance of VBSS clearly dominates HBSS.

8. Value-biasing local search starting configurations

In the previous section, we observed that LEE requires a tiny fraction of the CPU time required to perform multiple iterations of VBSS, but also observed that if we had the extra time, we could find significantly better solutions than LEE with VBSS. Perhaps, VBSS could be used as a means of extending the Lee et al. hill-climber to a multi-start algorithm. Here we examine the use of VBSS as a mechanism for seeding the starting solutions of Lee et al.'s local hill-climber. We refer to this VBSS seeded hill-climber as VBSS-HC—the first iteration of which hill-climbs on the ATCS heuristic solution and thus necessarily performs at least as well as LEE in terms of solution quality. The plot in figure 4 shows that the VBSS seeded hill-climber dominates the basic VBSS approach. The added cost of the hill-climb

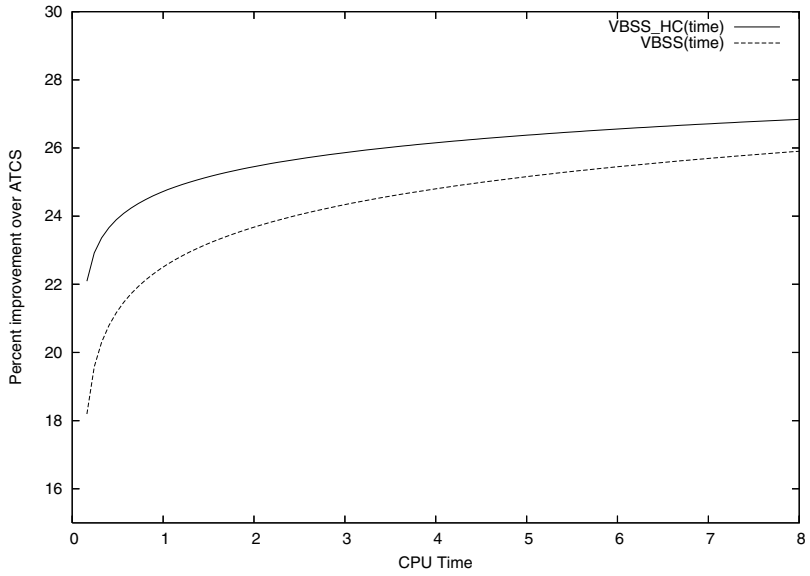

Figure 4. Comparison of the average percent improvement over the heuristic solution given by VBSS-HC and VBSS as a function of CPU time (seconds). VBSS-HC's and VBSS's performance are shown by the solid and dashed lines, respectively.

on each iteration in terms of CPU time is outweighed by the extra improvement in solution quality.

As an added point of comparison, we also consider beginning a simulated annealing search at the solution given by the ATCS dispatch policy. Simulated annealing attempts to escape local optima by allowing dis-improving moves. Such "bad" moves are made stochastically with probability that increases as the degree of dis-improvement decreases and also decreases as the search progresses, eventually settling into a local hill-climb. We have implemented a computationally intense simulated annealing procedure that uses a modified Lam schedule (see Boyan (1998) for description of the modified Lam schedule) and that is allowed to run for a total of 20 million evaluations. The simulated annealing approach is clearly dominated by HBSS-HC as shown in the plot of average percent improvement over time in figure 5.

In Table 4 we compare this VBSS seeded hill-climber (VBSS-HC), the hill-climber with unbiased random starting configurations (IS-HC), LEE, simulated annealing (SA), and the deterministic dispatch policy ATCS. We make the following observations:

– The unbiased hill-climber never finds a solution better than ATCS. In fact, in most cases (though not shown here) the best solution given by the unbiased hill-climber on any problem instance is far worse than that of the ATCS solution. The primary reason for this is that the Lee et al. hill-climber assumes a good starting configuration

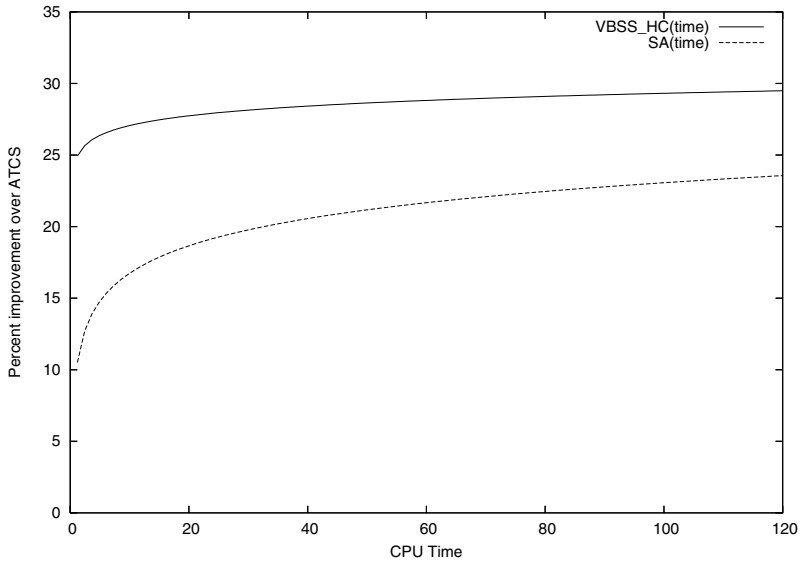

Figure 5. Comparison of the average percent improvement over the heuristic solution given by VBSS-HC and SA as a function of CPU time (seconds). VBSS-HC's and SA's performance are shown by the solid and dashed lines, respectively.

with its very limited operator set. Unbiased random starting solutions are highly unlikely to be such good starting solutions for the problem instances of this problem set.

– Using VBSS to seed the starting configurations requires approximately half the CPU time for the search as compared to using unbiased random starting configurations. Though it takes more time to generate starting solutions with VBSS (evident in the higher number of search nodes generated by VBSS-HC), the underlying assumption of Lee et al.'s hill-climber, that it starts at a good solution holds in the case of VBSS. Therefore, the hill-climb is short (note the lower numbers of search nodes and solutions nodes visited by VBSS-HC as compared to IS-HC). In contrast, it takes nearly negligible time to generate unbiased random starting solutions, but those solutions are far from local optima, so the hill-climb takes significantly more time.
– VBSS-HC requires only a fraction of the time required by our simulated annealing algorithm to produce solutions that are comparable or significantly better. Our implementation of simulated annealing uses a cooling schedule designed to allow the theoretically optimal schedule of non-improving moves that are accepted and is allowed to run for a significant amount of time. It is designed to provide an aggressive search in terms of solution quality, beginning from the heuristic solution. VBSS-HC offers a more cost-effective approach compared to this aggressive simulated annealing approach to expanding the search from the heuristic solution.

Table 4. VBSS plus a local hill-climb (VBSS-HC) vs Iterative Sampling plus a local hill-climb (IS-HC). VBSS-HC uses a polynomial bias function of degree 5.

Algorithm	API	TIME	Gen	Visits	Sols
ATCS	0.0	0.009	1,830	60	1
LEE	12.4 ± 0.15	0.012	1,909	61	2
SA (1)	25.3 ± 0.19	224.60	20,001,830	7,838,400	7,838,341
SA (3)	28.1 ± 0.19	673.10	60,001,830	23,515,080	23,515,021
VBSS-HC (1)	15.1 ± 0.16	0.021	3,791	122	4
VBSS-HC (10)	21.1 ± 0.18	0.123	20,725	667	22
VBSS-HC (100)	24.7 ± 0.19	1.122	190,076	6,122	202
VBSS-HC (200)	25.5 ± 0.19	2.236	378,243	12,183	402
VBSS-HC (500)	26.6 ± 0.19	5.570	942,744	30,366	1,002
VBSS-HC (1000)	27.4 ± 0.19	11.121	1,883,579	60,671	2,002
VBSS-HC (2500)	28.1 ± 0.19	27.742	4,706,084	151,586	5,002
VBSS-HC (5000)	28.7 ± 0.19	55.674	9,410,259	303,111	10,002
VBSS-HC (10000)	29.3 ± 0.19	114.140	18,818,609	606,161	20,002
IS-HC (1)	0.0	0.023	716	75	16
IS-HC (10)	0.0	0.226	7,163	754	164
IS-HC (100)	0.0	2.272	71,633	7,541	1,641
IS-HC (200)	0.0	4.543	143,266	15,082	3,282
IS-HC (500)	0.0	11.351	358,165	37,305	8,205
IS-HC (1000)	0.0	22.700	716,330	75,410	16,410
IS-HC (2500)	0.0	56.713	1,790,825	186,525	41,025
IS-HC (5000)	0.0	113.446	3,581,650	373,050	82,050
IS-HC (10000)	0.0	226.902	7,163,300	754,100	164,100

Both algorithms perform the hill-climb on the results of each of its iterations. ATCS is the deterministic heuristic result. LEE is the hill-climber (single-start, non-randomized) of Lee et al. SA is simulated annealing for the specified number of restarts beginning with ATCS solution.

9. Value-biased hill-climber versus systematic heuristic search

Harvey and Ginsberg consider the idea that for some constraint satisfaction problems, successor ordering heuristics in a tree search could lead directly to a solution; and that in the cases where a heuristic fails to lead directly to a solution, it may have succeeded had it not made a few mistakes along its path through the search tree. *Limited Discrepancy Search* (LDS) is designed with this rationale in mind (Harvey and Ginsberg, 1995). LDS begins by following the successor ordering heuristic through the search tree to a leaf. If the leaf is a solution, the search ends. Otherwise, it systematically considers all paths through the search tree with at most 1 "discrepancy" with the ordering heuristic (i.e., at most one decision is made contrary to the choice of the heuristic). If it fails to find a solution, then it considers all paths with at most 2 discrepancies with the heuristic, and then at most three, and so forth until either a solution is found or the search space is exhausted. *Improved Limited Discrepancy Search* (ILDS) eliminates much of the redundancy of the LDS algorithm by ensuring that each leaf node is visited at most once (Korf, 1996).

Walsh has similar motivation in the design of his *Depth-bounded Discrepancy Search* (DDS) (Walsh, 1997). Walsh, however, acknowledges the idea that search heuristics are often less-informed at the top of the search tree and often very-informed near the leafs of the tree. To deal with this, he combines aspects of LDS with aspects of iterative deepening search (Korf, 1985). On iteration 0, DDS follows the heuristic's advice to a leaf. On iteration $i+1$, DDS considers all discrepancies at depth i or less systematically in order of increasing discrepancy.

Although LDS and DDS are motivated by some of the same rationale as that of VBSS, their goals are somewhat different and are not well-suited to the types of problems for which we are concerned. LDS and DDS are concerned with finding a *feasible* solution as opposed to finding an *optimal* or *near-optimal* solution. Consider a problem where every leaf node is a solution but with drastically differing objective values. LDS or DDS would not have any idea of when to stop searching in this instance. Also, although they have the advantage of being complete search procedures (i.e., guaranteed to find a solution if one exists), this advantage can be a disadvantage when scalability to very large problem instances is considered due to its worst-case exponential complexity. Often, non-systematic search procedures are the ones that scale best to larger combinatorial problems (Langley, 1992; Freuder et al., 1995; Prestwich, 2001).

We now consider a computational comparison of VBSS and VBSS-HC with various versions of discrepancy search. The purpose of this comparison is to explore the power of stochastic sampling in combinatorial domains where systematic search gets bogged down considering a large number of un-promising solutions. For example, limited discrepancy search is required to systematically exhaust every solution path containing a single discrepancy from the heuristic's advice before it ever considers a solution path with two discrepancies. During this search, there might be several decision contexts in which the heuristic very strongly prefers one choice over the others, or very strongly prefers against one or more choices in some decision contexts. LDS has no way of skipping over the single discrepancy solution paths where it is clear to the heuristic that one or more choices are bad and thus systematically considers them anyway. Similarly, depth-bounded discrepancy search is unable to consider a solution with discrepancies at depth 4 or greater before it exhausts all solution paths with all possible discrepancy combinations from the root through and including depth 3. Although its assumption of a heuristic's tendency to be most fallible at the start of the search is a good one in most cases, at the same time if the search-space has a very large branching factor, then it is still likely to be considering a large number of solution paths that might be clearly un-promising to the heuristic. Our hypothesis is that stochastic sampling approaches are able to better tune the heuristic's advice to the context and though incomplete are capable of finding better solutions given limited time than the systematic approaches of LDS and DDS in domains where the branching factor is very high.

Specifically, we will consider the following variations of discrepancy search:

- LDS-all-single: Limited Discrepancy Search[10] considering all single discrepancy (or less) solution paths. There are 1, 771 such solution states in each of these 60 job problem instances.

Table 5. VBSS and VBSS-HC as compared to discrepancy search procedures.

Algorithm	API	TIME	Gen	Visits	Sols
VBSS (100)	22.7 ± 0.17	1.12	184,830	6,060	101
VBSS (500)	25.1 ± 0.18	4.44	916,830	30,060	501
VBSS (1000)	26.0 ± 0.18	8.87	1,831,830	60,060	1001
VBSS-HC (100)	24.7 ± 0.19	1.12	190,076	6,122	202
VBSS-HC (500)	26.6 ± 0.19	5.57	942,744	30,366	1,002
VBSS-HC (1000)	27.4 ± 0.19	11.12	1,883,579	60,671	2,002
VBSS-HC (10000)	29.3 ± 0.19	114.14	18,818,609	606,161	20,002
LDS-all-single	24.3 ± 0.17	6.04	1,601,615	70,270	1,771
LDS-all-two	28.8 ± 0.19	3902.04	912,864,816	48,219,521	1,533,116
DDS-depth-2	23.6 ± 0.18	20.94	6,056,940	205,320	3,540
DDS-depth-3	26.5 ± 0.18	911.72	339,393,960	11,703,240	205,320

– LDS-all-two: Limited Discrepancy Search[11] considering all two discrepancy (or less) solution paths. There are 1, 533, 116 such solution states.
– DDS-depth-2: Depth-Bounded Discrepancy Search considering all discrepancies through depth 2. There are 3, 540 solution states considered.
– DDS-depth-3: Depth-Bounded Discrepancy Search considering all discrepancies through depth 3. There are 205, 320 solution states considered.

The results of this comparison can be found in Table 5. The following observations are made:

– VBSS on average finds better solutions than LDS-all-single in less time (with 500 iterations of VBSS) and even better solutions with slightly more time than LDS (1000 iterations of VBSS).
– In approximately one twentieth of the CPU time, the VBSS seeded hill-climber (VBSS-HC with 100 iterations) finds solutions better than DDS (DDS-depth-2).
– In less than two minutes, the VBSS seeded hill-climber (VBSS-HC with 10000 iterations), finds solutions that are on average better than solutions found by allowing LDS to run for over an hour per instance (LDS-all-two) and that are on average better than allowing DDS to run for 15 minutes (DDS-depth-3).
– VBSS-HC can find better solutions while generating and visiting far fewer search nodes than LDS or DDS and while visiting far fewer solution nodes than LDS or DDS. In particular compare VBSS-HC with 10,000 iterations to LDS-all-two and DDS-depth-3. For example, in considering all solution paths with two or less discrepancies from the heuristic's advice, LDS generates nearly a billion search nodes, visiting almost 50 million of them, and evaluating over a million and a half solution nodes; while 10,000 iterations

of VBSS-HC generates just under 19 million search nodes visiting approximately 600 thousand, and evaluating only 20 thousand solution nodes.

The overall theme of these observations is that in far less CPU time, VBSS can find significantly better solutions on average as compared to the systematic discrepancy search procedures. This would clearly not be the case in all domains, but for this problem domain with the large branching factor, it is. Though often used in problems of optimization, LDS and DDS are better suited to constrained problem domains—for example, constraint satisfaction domains where we are instead looking for any feasible solution.

10. Conclusion

In this paper, we presented a stochastic sampling algorithm called VBSS. VBSS is a value-biased alternative to the rank-biased stochastic sampling algorithm known as HBSS. The power of the VBSS algorithm, as compared to HBSS, is its ability to make better use of the discriminatory power inherent in search heuristics within the stochastic sampling framework. This superior search performance was demonstrated when we showed that the VBSS framework is able to find significantly better solutions to weighted tardiness sequencing problems with sequence-dependent setups in significantly less computation time as compared to HBSS. The advantages of value-biasing decisions in a stochastic sampling search as compared to rank-biasing the decisions include:

- *Exploiting the inherent discriminatory power of heuristics*.
  In using the actual heuristic values within the stochastic decisions of the sampling algorithm, VBSS is better able to leverage the inherent discriminatory power of a search heuristic. This is both theoretically and experimentally valid. In theory, HBSS uses only the order of the choices given by the heuristic; while VBSS instead uses the heuristic values directly, incorporating the complete information available within the heuristic. We experimentally demonstrated this improved search performance on a weighted tardiness scheduling problem.
- *Improved decision-making efficiency*.
  Since the need to rank-order the choices at each step is alleviated in the value-biased approach of VBSS, the stochastic decisions can be computed in an efficient $O(n)$ time (small constant multiplier), rather than the $O(n \log n)$ time required by the obvious implementation of the rank-biased approach of HBSS. Though we do not rule out the possibility of developing an $O(n)$ rank-biased decision scheme, such a scheme would necessarily operate with a much larger constant multiplier compared to the value-biased approach.

We also showed that VBSS can be used to seed the starting solution configurations of a multistart hill-climber to enhance the search performance of that local search algorithm. This is somewhat contrary to popular belief that multistart hill-climbers perform better when

the starting configurations are random and unbiased. However, for the problems considered in this paper, we have strong heuristics at our disposal. These heuristics appear to be very well-informed in a large number of cases. Therefore, when using a randomization of these strong heuristics within a multistart hill-climber, we do see a benefit over unbiased random starts since each of these stochastic samples is likely to start the search at or near an already good solution. For weighted tardiness sequencing with sequence-dependent setups, we saw that such an approach has the ability to find better solutions as compared to truncated systematic heuristic search algorithms (such as LDS or DDS) in significantly less CPU time. Part of the reason for this result is the size of the problem space in this domain. LDS and DDS are forced to explore many heuristically unattractive solution trajectories (i.e., solution trajectories with only 1 or 2 discrepancies, but where the 1 or 2 discrepancies are large in heuristic value) due to their systematic nature, while VBSS can avoid such solution trajectories.

Appendix A. Weighted tardiness scheduling with sequence-dependent setups: A benchmark set

.1. Overview

This Appendix details the set of problem instances used in the experiments of this paper. These problem instances are available online at either of:

- http://www.cs.drexel.edu/~cicirello/benchmarks.html
- http://www.ozone.ri.cmu.edu/benchmarks.html

The file format for a problem instance file is described in Section .2. The current best known solutions to these instances are listed in Section .3.

.2. Instance file format

Each instance of the benchmark library is stored in a separate file according to the following file format:

```
Problem Instance: <instance number>
Problem Size: <number of jobs in instance>
Begin Generator Parameters
Tau: <tau>
R: <R>
Eta: <eta>
P_bar: <average process time>
P_MIN: <minimum process time>
P_MAX: <maximum process time>
S_bar: <average setup time>
```

```
MAX_WEIGHT: <maximum weight value>
C_max: <target makespan>
D_bar: <average duedate>
End Generator Parameters
Begin Problem Specification
Process Times:
<process time for job 0>
...
<process time for job n>
Weights:
<weight for job 0>
...
<weight for job n>
Duedates:
<duedate for job 0>
...
<duedate for job n>
Setup Times:
<job i> <job j> <setup time for j if it follows i>
// i=-1 indicates the setup time if j is first job
End Problem Specification
```

.3. *Best known solutions*

In this list of best known solutions, we list the problem instance number, best found objective value for the instance, and the algorithm that found it. Algorithms are referred to as follows:

- LDS-all-two: LDS (Limited Discrepancy Search) allowed to run long enough to consider all two-discrepancy solutions.
- HBSS,<iterations>,<bias>: HBSS with the specified number of iterations and a bias function equal to $\text{rank}^{-1*\text{bias}}$.
- VBSS,<iterations>,<bias>: VBSS with the specified number of iterations and a bias function equal to $\text{value}^{\text{bias}}$.
- VBSS-HC,<iterations>,<bias>: Random-restart hill-climber using VBSS to seed the starting solutions for the specified number of iterations and a bias function equal to $\text{value}^{\text{bias}}$.
- SA,<totalEvals>: Simulated annealing with a modified Lam schedule for the specified number of total evaluations. Search begins with ATCS solution.

Other algorithms considered (including DDS, other variations of LDS, IS, Lee et al.'s single-start hill-climber, the deterministic ATCS policy) also found some of these best known solutions but did not exclusively find any best known solutions. The best known solutions are as follows:

Problem Instance	Objective Value	Algorithm
1	978	VBSS-HC,10000,5
2	6489	VBSS-HC,10000,5
3	2348	VBSS-HC,5000,5
4	8311	SA,20000000
5	5606	VBSS-HC,1000,5
6	8244	VBSS,5000,5
7	4347	VBSS-HC,10000,5
8	327	VBSS-HC,5000,5
9	7598	LDS-all-two
10	2451	VBSS-HC,10000,5
11	5263	VBSS-HC,2500,5
12	0	LDS-all-two
13	6147	VBSS-HC,2500,5
14	3941	VBSS,2500,5
15	2915	VBSS-HC,5000,5
16	6711	VBSS-HC,10000,5
17	462	VBSS-HC,5000,5
18	2514	VBSS,10000,5
19	279	VBSS-HC,2500,5
20	4193	VBSS-HC,10000,5
21	0	LDS-all-two
22	0	LDS-all-two
23	0	LDS-all-two
24	1791	SA,20000000
25	0	SA,20000000
26	0	LDS-all-two
27	229	SA,20000000
28	72	SA,20000000
29	0	LDS-all-two
30	575	SA,20000000
31	0	LDS-all-two
32	0	LDS-all-two
33	0	LDS-all-two
34	0	LDS-all-two
35	0	LDS-all-two
36	0	LDS-all-two
37	2407	VBSS-HC,10000,5
38	0	LDS-all-two
39	0	LDS-all-two
40	0	LDS-all-two
41	73176	SA,20000000
42	61859	VBSS-HC,2500,5
43	149990	LDS-all-two
44	38726	SA,20000000
45	62760	VBSS,100,9
46	37992	SA,20000000
47	77189	SA,20000000
48	68920	LDS-all-two

(*Continued on next page.*)

(*Continued*).

49	84143	SA,20000000
50	36235	SA,20000000
51	58574	VBSS-HC,5000,5
52	105367	VBSS-HC,5000,5
53	95452	VBSS-HC,10000,5
54	123558	VBSS,2500,5
55	76368	VBSS,5000,5
56	88420	SA,20000000
57	70414	VBSS,10000,5
58	55522	VBSS-HC,10000,5
59	59060	VBSS-HC,10000,5
60	73328	VBSS-HC,10000,5
61	79884	SA,20000000
62	47860	SA,20000000
63	78822	VBSS,200,23
64	96378	SA,20000000
65	134881	SA,20000000
66	64054	SA,20000000
67	34899	SA,20000000
68	26404	SA,20000000
69	75414	SA,20000000
70	81200	SA,20000000
71	161233	VBSS-HC,1000,5
72	56934	VBSS-HC,5000,5
73	36465	LDS-all-two
74	38292	VBSS-HC,10000,5
75	30980	LDS-all-two
76	67553	VBSS-HC,5000,5
77	40558	SA,20000000
78	25105	SA,20000000
79	125824	VBSS,5000,5
80	31844	VBSS,10000,5
81	387148	VBSS,200,17
82	413488	VBSS,200,17
83	466070	SA,20000000
84	331659	VBSS,100,20
85	558556	SA,20000000
86	365783	VBSS,200,20
87	403016	SA,20000000
88	436855	VBSS,200,11
89	416916	VBSS-HC,5000,5
90	406939	LDS-all-two
91	347175	VBSS,100,14
92	365779	VBSS,100,11
93	410462	VBSS,100,19
94	336299	VBSS,100,10
95	527909	VBSS,100,9
96	464403	LDS-all-two
97	420287	VBSS,10000,5
98	532519	VBSS,100,12

(*Continued on next page.*)

(*Continued*).

99	374781	VBSS,100,12
100	441888	VBSS-HC,500,5
101	355822	LDS-all-two
102	496131	LDS-all-two
103	380170	VBSS,100,18
104	362008	VBSS,200,15
105	456364	VBSS,200,17
106	459925	LDS-all-two
107	356645	VBSS,200,15
108	468111	VBSS,200,21
109	415817	VBSS,10,13
110	421282	LDS-all-two
111	350723	VBSS,5000,5
112	377418	VBSS-HC,10000,5
113	263200	VBSS,100,5
114	473197	VBSS-HC,5000,5
115	460225	VBSS,100,10
116	540231	LDS-all-two
117	518579	VBSS-HC,10000,5
118	357575	LDS-all-two
119	583947	VBSS,200,13
120	399700	VBSS-HC,10000,5

Acknowledgments

This work has been funded in part by the Department of Defense Advanced Research Projects Agency and the U.S. Air Force Rome Research Laboratory under contracts F30602-97-2-0066 and F30602-00-2-0503 and by NASA under contract NCC2-1243. The views and conclusions contained in this document should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of NASA, ARPA, the Air Force or the U.S. Government.

Notes

1. Iterative sampling outperformed a few highly-customized problem-specific algorithms as real-world problem structure was added to the problem. The problem-specific algorithms had been over-fitted to the benchmark set.
2. It should be noted that in Watson et al.'s study, although (unbiased) iterative sampling showed promise, Watson et al.'s main point was that simple randomization of heuristics can outperform more customized algorithms.
3. Re-ranking is necessary at each step because the state (e.g., the position of the telescope and current time), and thus the heuristic ordering, change each time a new task is added to the tentative schedule. Also contributing to the context-dependent nature of the ranking is the fact that some observation tasks are only schedulable within specific time windows and thus are not always feasible choices.
4. In most cases, it is not possible to schedule all desired observations in Bresina's domain within the allotted time window.
5. This paper is an extended and updated version of the paper in which we first introduced VBSS, although in that paper we referred to the algorithm as WHISTLING (Wasp Behavior Inspired Stochastic Sampling)

(Cicirello and Smith, 2002). WHISTLING got its name from the particular mode of computation chosen for the stochastic decisions. In the present paper, we use the obvious method of computation for the roulette wheel decisions; but the results of VBSS and WHISTLING are numerically indistinguishable.

6. It has been pointed out that the worst-case complexity of choosing the i-th largest element from an unsorted list is $O(n)$. Thus, in theory, this stochastic selection operation can also be done in linear time. Under an assumption that the n choices have ranks 1 through n, we can select the winning rank without looking at the actual elements themselves. And then use the winning rank and the linear time selection algorithm to make the decision. However, the linear time algorithm for the selection operation itself is likely to be more of a theoretical interest than of practical interest. It has a large constant factor that results in the $O(n \log n)$ sort-first-then-select algorithm dominating for all but very large problem instances (Cormen et al., 1990). A second problem is that the assumption that the n elements have ranks 1 through n does not hold if more than one element may have the same heuristic value and thus the same rank. It may be possible (if one is clever enough) to devise a linear time method for finding all ties in heuristic value. However, such a method is certainly non-obvious and probably also non-trivial, adding to the already significant constant factor. Thus, although it may be possible to compute the HBSS decisions in linear time, it is probably not desirable to actually do so unless your search-space has a particularly large branching factor.
7. These issues are not necessarily the same as those involved with selecting a bias function for HBSS.
8. By nodes generated, we mean search nodes that have been generated but not necessarily explored. For example, when making a decision at depth d, VBSS and HBSS must compute the heuristic value associated with adding each of the $N - d$ nodes to the current sequence. In doing so, $N - d$ search nodes have been generated. Only one of these will subsequently be visited during this iteration of the search.
9. Note, however, that if it is possible to compute the HBSS decisions in linear time (i.e., if sorting is not necessary) then the speedup may be less than the 10-fold seen in this experiment. However, depending on the significance of the constant factor in such a linear time version in this problem domain with a branching factor of 60, the best HBSS can do in terms of CPU time might still be the $O(n \log n)$ results reported here.
10. The improved version (ILDS) of Korf (1996) is used.
11. Again, the improved version (ILDS) of Korf (1996) is used.

References

Adler, L., N.M. Fraiman, E. Kobacker, M. Pinedo, J.C. Plotnitcoff, and T.P. Wu. (1993). "BPSS: A Scheduling System for the Packaging Industry." *Operations Research* 41, 641–648.

Beaumariage, T. and K. Kempf. (1995). "Attractors in Manufacturing Systems with Chaotic Tendencies." Presentation at INFORMS-95, New Orleans, http://www.informs.org/Conf/NewOrleans95/ TALKS/TB07.3.html.

Boyan, J.A. (1998). "Learning Evaluation Functions for Global Optimization." Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania.

Boyan, J.A. and A.W. Moore. (1997). "Using Prediction to Improve Combinatorial Optimization Search." In *Proceedings of the Sixth International Workshop on Artificial Intelligence and Statistics (AISTATS-6)*.

Boyan, J.A. and A.W. Moore. (1998). "Learning Evaluation Functions for Global Optimization and Boolean Satisfiability." In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*.

Bresina, J.L. (1996). "Heuristic-Biased Stochastic Sampling." In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference, Volume One*, AAAI Press, pp. 271–278.

Cao, Y.J. and Q.H. Wu. (1999). "Optimization of Control Parameters in Genetic Algorithms: A Stochastic Approach." *International Journal of Systems Science* 30(5), 551–559.

Cesta, A., A. Oddi, and S.F. Smith. (1999). "An Iterative Sampling Procedure for Resource Constrained Project Scheduling with Time Windows." In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, pp. 1022–1029.

Cesta, A., A. Oddi, and S.F. Smith. (2002). "A Constraint-Based Method for Project Scheduling with Time Windows." *Journal of Heuristics* 8, 109–136.

Chiang, W.Y., M.S. Fox, and P.S. Ow. (1990). "Factory Model and Test Data Descriptions: OPIS Experiments." Technical Report CMU-RI-TR-90-05, The Robotics Institute, Carnegie Mellon University.

Cicirello, V.A. (1999). "Intelligent Retrieval of Solid Models." Master"s thesis, Department of Mathematics and Computer Science, Drexel University, Philadelphia, PA.

Cicirello, V.A. (2003). "Weighted Tardiness Scheduling with Sequence-Dependent Setups: A Benchmark Library." Technical report, Intelligent Coordination and Logistics Laboratory, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA. http://www.ozone.ri.cmu.edu/benchmarks.html.

Cicirello, V.A. and W.C. Regli. (1999). "Resolving Non-Uniqueness in Design Feature Histories." In W.F. Bronsvoort and D. C. Anderson (eds.), *Fifth ACM/SIGGRAPH Symposium on Solid Modeling and Applications*. MI: ACM Press Ann Arbor, pp. 76–84.

Cicirello, V.A. and W.C. Regli. (2001). "Machining Feature-Based Comparison of Mechanical Parts." In *International Conference on Shape Modeling and Applications*. Genova, Italy: IEEE Computer Society Press, pp. 176–185.

Cicirello, V.A. and W.C. Regli. (2002). "An Approach to Feature-based Comparison of Solid Models of Machined Parts." *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 16(5), 385–399.

Cicirello, V.A. and S.F. Smith. (2000). "Modeling GA Performance for Control Parameter Optimization." In D. Whitley, D. Goldberg, E. Cantü-Paz, L. Spector, I. Parmee, and H. Beyer (eds.), *GECCO-2000: Proceedings of the Genetic and Evolutionary Computation Conference*. Las Vegas, NV: Morgan Kaufmann Publishers, pp. 235–242.

Cicirello, V.A. and S.F. Smith. (2002). "Amplification of Search Performance through Randomization of Heuristics." In P. Van Hentenryck (ed.), *Principles and Practice of Constraint Programming—CP 2002: 8th International Conference, Proceedings*, vol. LNCS 2470 of *Lecture Notes in Computer Science*. Springer-Verlag. Ithaca, NY, pp. 124–138.

Cormen, T.H., C.E. Leiserson, and R.L. Rivest. (1990) *Introduction to Algorithms*. McGraw-Hill.

Eiben, A.E., R. Hinterding, and Z. Michalewicz. (1999). "Parameter Control in Evolutionary Algorithms." *IEEE Transactions on Evolutionary Computation* 3(2), 124–141.

Freuder, E.C., R. Dechter, M.L. Ginsberg, B. Selman, and E. Tsang. (1995). "Systematic Versus Stochastic Constraint Satisfaction." In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, pp. 2027–2032.

Goldberg, D.E.. (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison Wesley.

Gomes, C.P., B. Selman, and N. Crato. (1997). "Heavy-Tailed Distributions in Combinatorial Search." In *Principles and Practices of Constraint Programming (CP-97)*. Springer-Verlag, pp. 121–135.

Gomes, C., B. Selman, and H. Kautz. (1998). "Boosting Combinatorial Search through Randomization." In *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference*. AAAI Press, pp. 431–437.

Gomes, C.P., B. Selman, N. Crato, and H. Kautz. (2000). "Heavy-Tailed Phenomena in Satisfiability and Constraint Satisfaction Problems." *Journal of Automated Reasoning* 24, 67–100.

Harvey, W. and M. Ginsberg. (1995). "Limited Discrepency Search." In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, pp. 607–613.

Kempf, K. and T. Beaumariage. (1994). "Chaotic Behavior in Manufacturing Systems." In *AAAI-94 Workshop Program, Reasoning About the Shop Floor, Workshop Notes*. AAAI Press, pp. 82–96.

Korf, R. (1985). "Depth-First Iterative-Deepening: An Optimal Admissible Tree Search." *Artificial Intelligence* 27(1), 97–109.

Korf, R. (1996). "Improved Limited Discrepancy Search." In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference, vol. 1*. AAAI Press, pp. 286–291.

Langley, P. (1992). "Systematic and Nonsystematic Search Strategies." In *Artificial Intelligence Planning Systems: Proceedings of the First International Conference*. pp. 145–152.

Lee, Y.H., K. Bhaskaran, and M. Pinedo. (1997). "A Heuristic to Minimize the Total Weighted Tardiness with Sequence-dependent Setups." *IIE Transactions* 29, 45–52.

McKay, K.N. (1993). "The Factory from Hell—a Modelling Benchmark." In *Proceedings of the NSF Workshop on Intelligent, Dynamic Scheduling for Manufacturing Systems*. pp. 99–114.

Morley, D. (1996). "Painting Trucks at General Motors: The Effectiveness of a Complexity-Based Approach." In *Embracing Complexity: Exploring the Application of Complex Adaptive Systems to Business*. The Ernst and Young Center for Business Innovation, pp. 53–58.

Morley, D. and C. Schelberg. (1993). "An Analysis of a Plant-Specific Dynamic Scheduler." In *Proceedings of the NSF Workshop on Intelligent, Dynamic Scheduling for Manufacturing Systems*. pp. 115–122.

Morton, T.E. and D.W. Pentico. (1993) *Heuristic Scheduling Systems: With Applications to Production Systems and Project Management*. John Wiley and Sons.

Oddi, A. and S.F. Smith. (1997). "Stochastic Procedures for Generating Feasible Schedules." In *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference*. AAAI Press, pp. 308–314.

Prestwich, S. (2001). "Local Search and Backtracking vs Non-Systematic Backtracking." In *Using Uncertainty Within Computation: Papers from the 2001 AAAI Fall Symposium, Technical Report FS-01-04*.AAAI Press, pp. 109–115.

Rachamadugu, R.V. and T.E. Morton. (1982). "Myopic Heuristics for the Single Machine Weighted Tardiness Problem." Working Paper 30-82-83, GSIA, Carnegie Mellon University, Pittsburgh, PA.

Raman, N., R.V. Rachamadugu, and F.B. Talbot. (1989). "Real Time Scheduling of an Automated Manufacturing Center." *European Journal of Operational Research* 40, 222–242.

Selman, B., H. Kautz, and B. Cohen: (1996). "Local Search Strategies for Satisfiability Testing." In D.S. Johnson and M.A. Trick (eds.), *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, October 11-13, 1993*, vol. 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society.

Sen, A.K. and A. Bagchi. (1996). "Graph Search Methods for Non-Order-Preserving Evaluation Functions: Applications to Job Sequencing Problems." *Artificial Intelligence* 86(1), 43–73.

Walsh, T. (1997). "Depth-Bounded Discrepency Search." In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, pp. 1388–1395.

Watson, J.P., L. Barbulescu, A.E. Howe, and L.D. Whitley. (1999). "Algorithm Performance and Problem Structure for Flow-shop Scheduling." In *Proceedings, Sixteenth National Conference on Artificial Intelligence (AAAI-99), Eleventh Innovative Applications of Artificial Intelligence Conference (IAAI-99)*. AAAI Press, pp. 688–695.