Amplification of Search Performance through Randomization of Heuristics

Vincent A. Cicirello and Stephen F. Smith

The Robotics Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
{cicirello, sfs}@cs.cmu.edu

Abstract. Randomization as a means for improving search performance in combinatorial domains has received increasing interest in recent years. In optimization contexts, it can provide a means for overcoming the deficiencies of available search heuristics and broadening search in productive directions. In this paper, we consider the issue of amplifying the performance of a search heuristic through randomization. We introduce a general framework for embedding a base heuristic within an iterative sampling process and searching a stochastic neighborhood of the heuristic's prescribed trajectory. In contrast to previous approaches, which have used rank-ordering as a basis for randomization, our approach instead relies on assigned heuristic value. Use of heuristic value is important because it makes it possible to vary the level of stochasticity in relation to the discriminatory power of the heuristic in different decision contexts, and hence concentrate search around those decisions where the heuristic is least informative. To evaluate the efficacy of the approach, we apply it to a complex, weighted-tardiness scheduling problem. Taking a state-of-the-art heuristic for this scheduling problem as a starting point, we demonstrate an ability to consistently and significantly improve on the deterministic heuristic solution across a broad range of problem instances. Our approach is also shown to consistently outperform a previously developed, rank-ordering based approach to randomizing the same heuristic in terms of percentage of improvement obtained.

1 Introduction

A fundamental design challenge for search procedures in combinatorial domains concerns how to promote good coverage of higher-valued points of the solution space in a computationally efficient manner. Search heuristics provide a basis for managing search efficiency in many domains. But since heuristics are not infallible, an optimizing search process must in many cases balance adherence to their advice against the possibility of missing better solutions.

Randomization provides a natural approach to hedging on this trade-off. One general schema that has gained increasing attention in recent years follows an iterative sampling framework - solutions (or partial solutions) are generated using

a base search procedure with some amount of non-determinism and this procedure is repeatedly re-started to explore different solution paths. In constraint satisfaction problem-solving (CSP) domains, the observed heavy-tailed nature of search runtime distributions across different solution trajectories provides a conceptual rationale for adopting this approach [6]. By cutting off search at specified computational time-limits (in essence, abandoning trials that belong to the heavy-tail) and repeatedly restarting along different solution paths, the overall search process can reach more productive regions of this search space sooner. In practice, this is precisely the behavior observed; the approach has been shown to significantly reduce the run-times of complete search procedures on hard problem instances across a range of problem domains [6, 10].

In the realm of combinatorial optimization, randomization and iterative sampling have similarly yielded productive results [13, 2, 3, 6, 5], although the underlying rationale is somewhat different. The concern in this case is more than minimizing the time required to obtain a feasible solution; it is instead to find the best possible solution as quickly as possible. Given this optimizing search focus, research has tended to place greater reliance on strong, domain-specific heuristics and emphasize randomization of incomplete search approaches. The rationale starts from the assumption that a good heuristic is available and argues that the neighborhood around its preferred solution path is a good one to search.[1] In essence, randomization serves to amplify the performance of the heuristic by allowing search to compensate for its occasional misstep.

One crucial aspect of any iterative sampling search procedure is the approach taken to randomizing the base search heuristic. Conceptually, the goal is to perturb the choice order prescribed by the heuristic at a given decision point in a way that preserves much of the heuristic's original bias. A simple, non-interfering strategy is to simply break ties randomly in those decision contexts where two or more top choices are ranked equivalently (i.e., making random decisions only when the heuristic fails to make a selection). A somewhat more aggressive approach, called heuristic equivalency [6], chooses randomly among all choices having a heuristic value within $H\%$ of the highest. Bresina [2] defined a more general framework called Heuristic-Biased Stochastic Sampling (HBSS), which utilizes a specified bias function as a basis for deviating from the choice order given by the base heuristic. Through use of different bias functions, a wide range of randomization strategies can be realized, including the two basic strategies just mentioned.

In this paper, we propose an alternative approach to randomizing search heuristics within an iterative sampling framework. Our approach has much in common with the HBSS approach of Bresina [2], but with one important difference. Instead of operating with respect to the rank ordering of choices that is implied by application of the heuristic, our approach biases selection on the actual values assigned by the heuristic to each possible choice. Like Oddi and Smith [10], we assume that the heuristic can be more or less discriminating in

[1] This rationale also underlies the design of some recent deterministic search procedures such as discrepancy search [7, 12].

different decision contexts and argue that the heuristic's degree of preference for one choice over another should impact the random selection process. Our approach is inspired by a stochastic model of how wasp colonies self-organize (or order themselves) into dominance hierarchies. However, the technique is lifted from the Biological context and reformulated as a general framework for randomizing heuristics. Borrowing from HBSS, we designate our approach Wasp beHavior Inspired STochastic sampLING (WHISTLING).

We demonstrate the advantage of this "value-based" approach to randomization through application to a class of sequence-dependent, weighted-tardiness scheduling problems. This domain is interesting in two respects. On one hand, it presents a difficult combinatorial optimization challenge. On the other, it is a practical problem and significant prior effort has gone into the development of good heuristics. Taking a state-of-the-art heuristic for this problem class as a starting point, we show that our technique consistently improves the quality of the solution obtained by straight deterministic application of the heuristic, and consistently outperforms HBSS in terms of percentage of improvement obtained. Further, as a consequence of relying directly on the values assigned to different choices by the heuristic, our approach is an inherently more efficient computational process. Before describing our approach, we first summarize the earlier developed HBSS framework.

2 Heuristic-Biased Stochastic Sampling

As just indicated, Bresina's HBSS 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. A random choice process is invoked to make each decision, and 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 overall 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 the scheduling of observations on a telescope. 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.[2] This process iterates until either all potential tasks have been scheduled or the time

[2] 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

Algorithm 1: Heuristic-Biased Stochastic Sampling (HBSS)
Input: Number of iterations I; a "heuristic" function; a "bias" function; an "objective" function; and a set of tasks T_j to schedule.
Output: A schedule S.

```
HBSS(I, heuristic, bias, objective, tasks T_j)
    bestsofar ← DispatchScheduling(T_j, heuristic)
    repeat I times
        S ← the empty schedule
        while not all tasks scheduled
            foreach unscheduled task T
                score[T] ← heuristic(T, S)
            sort all tasks T according to score[T]
            foreach unscheduled task T
                rank[T] ← sort position of T
                weight[T] ← bias(rank[T])
                totalweight ← totalweight + weight[T]
            foreach unscheduled task T
                prob[T] ← weight[T] / totalweight
            select randomly the next task biased according to prob[T]
                add this selected task to the candidate schedule S
                remove this selected task from the set of unscheduled tasks
        evaluate[S] ← objective(S)
        if evaluate[S] is superior to evaluate[bestsofar]
            bestsofar ← S
    return bestsofar
```

Fig. 1. The HBSS Algorithm

frame has been exhausted.[3] The resulting schedule is then evaluated globally and the search process is restarted. The HBSS algorithm is illustrated in the context of this scheduling problem in Algorithm 1.

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. In Bresina's HBSS framework [2] 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, where r is the rank of the choice in question. As pointed out by Bresina [2], 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. The potential problem is that heuristics are typically more or less informed in different decision contexts; and it is not possible to calibrate the degree of randomness allowed according to this dynamic aspect of problem solving state.

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.

[3] In most cases, it is not possible to schedule all desired observations in Bresina's domain within the allotted time window.

This capability requires movement away from an approach to random bias based strictly on rank order. We introduce such an approach in the next section.

3 Wasp Behavior Inspired Stochastic Sampling (WHISTLING)

Our stochastic search framework makes its random decisions using a value-based bias rather than a rank-based bias. Random decisions are made in a manner equivalent to spinning a roulette wheel, where each choice is given a section of the wheel proportional in size to the value given by the heuristic to making that choice.

Our approach to computing this roulette wheel decision derives from a naturally-inspired computational model of the self-organization that takes place within a colony of wasps [11]. In nature, a hierarchical social order among the wasps of the colony is formed through interactions among individual wasps of the colony. This emergent social order is a succession (or prioritization) of wasps from the most dominant to the least dominant. In the model of Theraulaz *et al.*, the results of these interactions are determined stochastically based on the "force" variables of the wasps involved. The probability of wasp 1 winning a dominance contest against wasp 2 is defined based on the force variables, F_1 and F_2, of the wasps as:

$$P(F_1, F_2) = \frac{F_1^2}{F_1^2 + F_2^2} \ . \tag{1}$$

After such an interaction, the value of the force variable of the winner is increased and the value of the force variable of the loser is decreased. Over time, through many such interactions, a hierarchical social order is formed.

We can map the above model to the problem of randomizing heuristic choices by associating a wasp as a proxy for each possible choice in a decision context, and defining the force of a given wasp to be a function of the heuristic value assigned to its corresponding choice.

More precisely, we define the force of a wasp as:

$$F_w = \text{bias}(\text{heuristic}(\text{Choice}_w)) + D \ , \tag{2}$$

where Choice_w is the choice represented by wasp w, heuristic() is a function that returns the heuristic value of the given choice, and bias() is a bias function (as in the HBSS framework). D is a variable that is initialized to 0 and varies according to the results of dominance contests during a run of the algorithm (see below). It mimics the fluctuations that occur in the force variable values as dominance contests between real wasps are won and lost in nature.

Given that our definition of F_w includes an explicit bias factor, we are able to generalize, as well as simplify, the stochastic rule for choosing the winner of a dominance competition (Equation 1) as follows:

$$P(F_1, F_2) = \frac{F_1}{F_1 + F_2} \ . \tag{3}$$

Algorithm 2: Wasp beHavior Inspired STochastic sampLING (WHISTLING)
Input: Number of iterations I; a "heuristic" function; a "bias" function; an "objective" function; and a set of tasks T_j to schedule.
Output: A schedule S.

```
WHISTLING(I, heuristic, bias, objective, tasks T_j)
    bestsofar ← DispatchScheduling(T_j, heuristic)
    repeat I times
        S ← the empty schedule
        while not all tasks scheduled
            foreach unscheduled task T
                force[T] ← bias(heuristic(T, S))
            WinnerSoFar ← arbitrary task T from the unscheduled tasks
            Challengers ← the set of unscheduled tasks − WinnerSoFar
            foreach task T in the set Challengers
                with probability P(force[T], force[WinnerSoFar]) (see Eq. 3)
                    force[T] ← force[T] + force[WinnerSoFar]
                    WinnerSoFar ← T
                otherwise
                    force[WinnerSoFar] ← force[WinnerSoFar] + force[T]
            Add WinnerSoFar to the candidate schedule S
            Remove WinnerSoFar from the set of unscheduled tasks
        evaluate[S] ← objective(S)
        if evaluate[S] is superior to evaluate[bestsofar]
            bestsofar ← S
    return bestsofar
```

Fig. 2. The WHISTLING Algorithm

By coupling this new definition in Equation 3 with an appropriate bias function in the force definition of Equation 2 (i.e., p^2) we can express the original rule given in Equation 1. However, like HBSS, our reformulation allows the expression of a range of bias functions. Bonabeau *et al.* generalize their definition as $P(F_1, F_2) = \frac{F_1^a}{F_1^a + F_2^a}$ where a is a parameter [1]. However, their motivation is somewhat different. They discuss how this formula models the behavior of real insect societies and how different values for a may more closely model a particular society's behavior. Furthermore, force is a variable in their model that adapts according to wins and losses of dominance contests and is not explicitly defined functionally as we do here. Our definition allows for the spectrum of polynomials as does Bonabeau *et al.*'s; but it also allows expression of many others (e.g., exponentials and logarithms) that cannot be expressed by Bonabeau *et al.*'s generalization.

Given the above definitions of $P(F_1, F_2)$ and F_w, we can make the sampling algorithm concrete by defining a specific competition structure. In what follows, we assume the following style of tournament. An initial wasp (choice) is selected arbitrarily and proceeds to engage in successive competitions with all other wasps (other choices). The initial wasp continues to compete as long

as it wins. If it is defeated, the new winner takes its place and proceeds to face the remaining candidates. Upon winning a competition, the winning wasp's D variable is updated to accumulate the force variable value of the loser, and the loser drops out. The wasp (choice) remaining at the end of the tournament is returned as the final selection.

It can be shown that this computation is equivalent to selecting according to a standard roulette wheel decision, with each possible choice Choice_w taking a chunk of the wheel proportional to $\text{bias}(\text{heuristic}(\text{Choice}_w))$. The advantage of this method of computing a roulette wheel decision over the usual one is that a single pass through the set of choices is required. The alternative would be to make one initial pass to compute $\sum_j \text{bias}(\text{heuristic}(\text{Choice}_j))$ followed by a second pass to choose w with probability $\frac{\text{bias}(\text{heuristic}(\text{Choice}_w))}{\sum_j \text{bias}(\text{heuristic}(\text{Choice}_j))}$. A proof of the equivalence of the tournament of dominance contests to a roulette wheel decision is provided in Appendix A.

The WHISTLING algorithm is presented in Algorithm 2, expressed in the same scheduling problem context used earlier to describe HBSS. Line 6 of the algorithm is the initial definition of the force of the scheduling wasps with $D = 0$ (see Equation 2). Line 11 and line 14 represent the increase in the value of D in Equation 2 for the winning wasp. The dominance contests (see Equation 3) take place in line 10 of the algorithm.

Computationally, the WHISTLING algorithm selects the next task to schedule in $O(T)$ time where there are T tasks to be scheduled. Since it must do this T times, the core sampling procedure has an overall algorithmic complexity of $O(T^2)$. In fact, the complexity of a corresponding deterministic dispatch scheduling procedure is also $O(T^2)$. Hence, WHISTLING 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 1, we can see that WHISTLING is asymptotically more efficient. Since HBSS biases its stochastic decisions according to a rank-ordering of the tasks, it necessarily sorts the tasks according to their heuristic values each time a task is scheduled – an $O(T \log T)$ operation.[4] And with T tasks to schedule, the

[4] 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 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. One problem with this 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. This assumption can cause one or more choices to be ranked differently than they should be as well as it can cause choices which should be ranked equivalently to be ranked differently. Furthermore, the linear time algorithm for the selection operation itself is 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 [4].

algorithmic complexity of HBSS is $O(T^2 \log T)$. In the next Section, we will see how WHISTLING and HBSS compare experimentally in a particular scheduling domain.

4 Experiments

4.1 Experimental Design

To demonstrate and experimentally evaluate the WHISTLING framework, we turn to the domain of factory scheduling and the fairly broad body of work in the area of dispatch scheduling heuristics (c.f., [9]). In dynamic factory environments, dispatch scheduling heuristics provide a practical, robust basis for managing execution. 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.

We focus specifically on a weighted tardiness scheduling problem with sequence-dependent setups. The objective of this problem is to sequence the set of jobs J on a machine so as to minimize the total weighted tardiness:

$$T = \sum_{j \in J} w_j T_j = \sum_{j \in J} w_j \max(f_j - d_j, 0) \ , \tag{4}$$

where T_j is the tardiness of job j; w_j, f_j, d_j are the weight, finish time, and due-date of job j. The problem is complicated by the fact that it takes variable amounts of time to reconfigure (or setup) the machine when switching between any two jobs.

To address this problem within our stochastic framework, we take as our starting point the *Apparent Tardiness Cost with Setups (ATCS)* dispatch heuristic [8]. 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:

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

where t is the current time; l is the index of the job just completed (or the last job added to the schedule); w_j, p_j, d_j are the weight, processing time, and due-date of job j (the job for which we are computing the heuristic value), respectively; $\bar{p}$ is the average processing time of all jobs; $\bar{s}$ is the average setup time; s_{lj} is the amount of setup time needed if job j is sequenced after job l.

Table 1. Comparison of average percent improvement of 1 iteration of WHISTLING to 1 iteration of HBSS over the deterministic heuristic solution of ATCS for different classes of problem instance. n is the number of instances in the problem class. "Lee" shows the results of Lee *et al.*'s hill climber.

Problem				Lee	HBSS Bias Function		WHISTLING Bias Function		
τ	R	η	n		p^{-4}	p^{-5}	p^{6}	p^{7}	p^{8}
0.3	0.25	0.25	10	**19.60**	10.21	7.36	15.07	4.60	8.94
0.3	0.25	0.75	10	**24.26**	7.92	9.48	22.29	23.81	11.83
0.3	0.75	0.25	10	**50.09**	27.23	14.54	21.81	20.18	26.97
0.3	0.75	0.75	10	**40.15**	14.09	8.92	21.98	18.89	28.34
0.6	0.25	0.25	10	0.17	2.01	1.47	**2.95**	1.96	1.81
0.6	0.25	0.75	10	1.12	3.16	0.98	2.39	3.05	**3.51**
0.6	0.75	0.25	10	0.23	0.30	0.65	**1.31**	1.12	0.52
0.6	0.75	0.75	10	0.65	0.40	2.04	1.88	5.36	**6.06**
0.9	0.25	0.25	10	0.00	**0.39**	0.33	0.00	0.00	0.00
0.9	0.25	0.75	10	0.00	**0.25**	0.03	0.16	0.22	0.00
0.9	0.75	0.25	10	0.04	0.00	**0.35**	0.00	0.12	0.00
0.9	0.75	0.75	10	0.00	0.21	0.04	0.00	0.29	**0.65**
0.3	*	*	40	**33.53**	14.86	10.08	20.29	16.87	19.02
0.6	*	*	40	0.54	1.47	1.29	2.13	2.87	**2.98**
0.9	*	*	40	0.01	**0.21**	0.19	0.04	0.16	0.16
*	0.25	*	60	**7.53**	3.99	3.28	7.14	5.61	4.35
*	0.75	*	60	**15.19**	7.04	4.42	7.83	7.66	10.42
*	*	0.25	60	**11.69**	6.69	4.12	6.86	4.66	6.37
*	*	0.75	60	**11.03**	4.34	3.58	8.12	8.60	8.40
*	*	*	120	**11.36**	5.51	3.85	7.49	6.63	7.39

k_1 and k_2 are parameters for tuning the heuristic. In the experiments reported below, we set the values of these parameters according to Lee *et al.*'s original recommendations [8]. When applied deterministically, the next job j added to the schedule using the ATCS heuristic is simply:

$$j = \arg\max_j \mathrm{ATCS}_j(t, l) \ . \tag{6}$$

The problem instances that we consider are generated according to Lee *et al.*'s procedure [8]. That is, 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.

With regard to configuration of the WHISTLING and HBSS search procedures, we consider runs with polynomial bias functions of degree $d = 1 \ldots 8$ and

Table 2. Comparison of average percent improvement of 10 iterations of WHISTLING to 10 iterations of HBSS over the deterministic heuristic solution of ATCS for different classes of problem instance. n is the number of instances in the problem class. "Lee" shows the results of Lee *et al.*'s hill climber.

Problem				Lee	HBSS Bias Function		WHISTLING Bias Function		
τ	R	η	n		p^{-4}	p^{-5}	p^6	p^7	p^8
0.3	0.25	0.25	10	19.60	18.57	25.34	28.85	25.46	**32.09**
0.3	0.25	0.75	10	24.26	32.07	29.00	**44.74**	41.47	40.30
0.3	0.75	0.25	10	**50.09**	41.84	48.41	49.68	40.80	49.22
0.3	0.75	0.75	10	40.15	44.36	53.18	54.85	53.60	**58.96**
0.6	0.25	0.25	10	0.17	3.63	4.84	3.63	4.20	**5.21**
0.6	0.25	0.75	10	1.12	9.21	7.00	8.85	**13.04**	11.19
0.6	0.75	0.25	10	0.23	3.03	1.86	**4.51**	3.68	4.21
0.6	0.75	0.75	10	0.65	6.99	11.88	11.07	12.91	**12.92**
0.9	0.25	0.25	10	0.00	**1.25**	0.86	0.30	0.39	1.14
0.9	0.25	0.75	10	0.00	1.26	1.08	0.97	0.73	**1.49**
0.9	0.75	0.25	10	0.04	**0.64**	0.48	0.00	0.13	0.23
0.9	0.75	0.75	10	0.00	1.01	1.10	0.87	**1.26**	0.76
0.3	*	*	40	33.53	34.21	38.98	44.53	40.33	**45.14**
0.6	*	*	40	0.54	5.72	6.40	7.02	**8.46**	8.38
0.9	*	*	40	0.01	**1.04**	0.88	0.54	0.63	0.91
*	0.25	*	60	7.53	11.00	11.35	14.56	14.22	**15.24**
*	0.75	*	60	15.19	16.31	19.49	20.16	18.73	**21.05**
*	*	0.25	60	11.69	11.49	13.63	14.50	12.44	**15.35**
*	*	0.75	60	11.03	15.82	17.21	20.23	20.50	**20.94**
*	*	*	120	11.36	13.66	15.42	17.36	16.47	**18.14**

number of iterations $I = 1, 10, 100$.[5] The heuristic and objective are as indicated above. We benchmark both WHISTLING and HBSS in terms of average percent improvement over the deterministic heuristic solution of ATCS.

4.2 Results

The average percent improvement of HBSS and WHISTLING over the deterministic dispatch scheduling policy of ATCS is shown in Table 1, Table 2, and Table 3 for one iteration, ten iterations, and 100 iterations of HBSS, respectively. The tables show the results obtained with the best-performing bias functions for each approach (i.e., p^{-4} and p^{-5} for HBSS; p^6, p^7, and p^8 for WHISTLING).[6]

[5] Negative polynomials are used in the case of HBSS since bias here is defined with respect to rank (where a lower rank is preferred). For WHISTLING, we use positive polynomials since bias is instead defined with respect to heuristic value (where a higher value is preferred).

[6] The authors can be contacted for the equivalent data for all bias functions considered during experimentation - lack of space prohibits their inclusion here.

Table 3. Comparison of average percent improvement of 100 iterations of WHISTLING to 100 iterations of HBSS over the deterministic heuristic solution of ATCS for different classes of problem instance. n is the number of instances in the problem class. "Lee" shows the results of Lee *et al.*'s hill climber.

Problem				Lee	HBSS Bias Function		WHISTLING Bias Function		
τ	R	η	n		p^{-4}	p^{-5}	p^{6}	p^{7}	p^{8}
0.3	0.25	0.25	10	19.60	37.19	31.43	37.67	**38.92**	35.78
0.3	0.25	0.75	10	24.26	49.52	52.39	55.04	**55.93**	54.85
0.3	0.75	0.25	10	50.09	59.15	59.02	61.37	61.70	**63.97**
0.3	0.75	0.75	10	40.15	63.66	62.61	**65.28**	64.83	63.61
0.6	0.25	0.25	10	0.17	7.42	7.62	7.82	**9.18**	8.24
0.6	0.25	0.75	10	1.12	12.27	14.37	15.56	**17.18**	15.94
0.6	0.75	0.25	10	0.23	7.01	8.13	8.19	9.16	**9.32**
0.6	0.75	0.75	10	0.65	16.20	16.46	19.03	19.40	**19.94**
0.9	0.25	0.25	10	0.00	1.68	**1.76**	0.70	1.28	1.66
0.9	0.25	0.75	10	0.00	2.36	2.11	2.15	2.52	**3.01**
0.9	0.75	0.25	10	0.04	1.10	**1.27**	0.44	0.73	1.07
0.9	0.75	0.75	10	0.00	2.19	2.08	1.45	2.29	**2.58**
0.3	*	*	40	33.53	52.38	51.36	54.84	**55.35**	54.55
0.6	*	*	40	0.54	10.73	11.65	12.65	**13.73**	13.36
0.9	*	*	40	0.01	1.83	1.81	1.19	1.71	**2.08**
*	0.25	*	60	7.53	18.41	18.28	19.82	**20.84**	19.91
*	0.75	*	60	15.19	24.89	24.93	25.96	26.35	**26.75**
*	*	0.25	60	11.69	18.93	18.21	19.37	**20.16**	20.01
*	*	0.75	60	11.03	24.37	25.00	26.42	**27.03**	26.66
*	*	*	120	11.36	21.65	21.60	22.89	**23.59**	23.33

We also show, as a basis for comparison, the average percent improvement over the deterministic solution obtained by the hill climber designed by Lee *et al.* to improve the solutions found by their heuristic [8]. After sequencing the jobs according to the heuristic, this hill climber greedily selects the job that contributes the most to the weighted tardiness objective. It then considers swapping this job with each of the twenty nearest jobs in the sequence.[7] If any of these swaps improve the objective score of the sequence, it takes the one that improves it the most. The algorithm continues until no further improvement can be made through this method. In the tables, the results of this method are listed as "Lee".

The first 12 rows of each table show the results for each the 12 problem classes individually. The next three rows show the results of aggregating these classes according to the τ (due-date tightness) parameter. This is followed in the next two rows by the results of aggregating the classes according to the R (due-date range) parameter. The next two rows show the results of aggregating the

[7] It considers only the twenty nearest jobs rather than all jobs because through experimentation Lee *et al.* found that considering jobs sequenced further apart by the heuristic for this swap did not buy them much.

problem classes by the η (setup time severity) parameter. The final row of each table then gives the average results across all problem instances. The first four columns of each table describe the problem class by the three aforementioned parameters as well as the number of problem instances in the class. For each of the 12 classes there are 10 instances (the aggregate classes have more instances).

Interpreting these results, we can make several observations:

- At all number of iterations considered, WHISTLING exhibits a greater overall average percentage improvement over the deterministic ATCS strategy than does HBSS. In fact, all three top WHISTLING configurations outperform the best HBSS configuration over all problem instances (see last row of tables).
- The hill climber of Lee *et al.* outperforms both a single iteration of HBSS as well as WHISTLING. But for 10 and 100 iterations, the stochastic sampling algorithms begin finding better solutions than the hill climber. It may be interesting to investigate using WHISTLING to generate starting points for a random restart variation of Lee *et al.*'s hill climber, but this has not been done.
- All three WHISTLING configurations equal or outperform the best HBSS configuration in all aggregated problem categories (rows 13 - 19) except for the $\tau = 0.9$ class (row 15). In this one particular problem category, only for 100 iterations does WHISTLING do better than HBSS. However, for all other categories, WHISTLING dominates.
- Perhaps the strongest comparative advantage can be seen on the 100 iteration runs (see Table 3). Here, the WHISTLING configuration with bias function p^7 alone produces a greater percent improvement than both HBSS configurations for every problem class (individual and aggregated) except 3. These results are statistically significant according to paired t-tests.

All of these results can be strengthened by considering computation time. Recall that the HBSS algorithm requires a rank-ordering step that adds to the computational complexity. For the problems considered here with 60 jobs, on a Sun Ultra 1, HBSS requires on average 0.16 seconds for one iteration, 1.59 seconds for ten iterations, and 15.46 seconds for 100 iterations; while WHISTLING only requires 0.016 seconds for one iteration, 0.16 seconds for ten iterations, and 1.50 seconds for 100 iterations. So ten iterations of WHISTLING can be performed in approximately the same amount of CPU time as one iteration of HBSS (and similarly 100 iterations of WHISTLING in the same CPU time as ten iterations of HBSS).

5 Conclusion

In this paper, we have presented a new iterative sampling framework for searching a stochastic neighborhood of an ordering heuristic. This algorithm, which we call WHISTLING, is defined in much the same spirit as the earlier developed

HBSS framework - to provide a general means for overcoming the potential misstep that any given search heuristic is prone to take. However, the WHISTLING approach differs from that of HBSS in one crucial respect: it bases its stochastic decisions on assigned heuristic values instead of a rank-ordering of alternatives. This value-based approach allows the WHISTLING procedure to take into account the discriminating power of the heuristic in different decision contexts, and to better concentrate search around those decisions where the heuristic provides less guidance. As a beneficial side-effect of the use of heuristic value rather than rank order, there is no longer the need to sort the set of choices at each step of the search, and hence WHISTLING is also an inherently more efficient computational process than HBSS.

In our experimental analysis of the performance of WHISTLING, we have focused on a weighted tardiness scheduling problem with sequence-dependent setups. Taking a state-of-the-art dispatch heuristic (ATCS) for this class of problem as a starting point, we tested several configurations of WHISTLING and HBSS over a broad range of problem instances. With relatively few iterations, WHISTLING was shown to consistently and significantly improve on the deterministic ATCS solution. HBSS is also seen to produce significantly better solutions than ATCS alone. However, on runs covering varying numbers of iterations and over problem classes of varying degrees of difficulty and constraint tightness, WHISTLING's performance is seen to dominate HBSS's in terms of average percentage of improvement. The advantage for stochastic search of an ability to vary the degree of randomness as a function of the discriminatory power of the heuristic (or the lack thereof) in specific decision contexts is clearly evident in this domain.

These results suggest several directions for future research. On one front, we are interested in exploring application of the WHISTLING framework in other combinatorial optimization contexts. Another interest is in exploiting the observed anytime characteristics of our stochastic search framework to enhance online decision-making in dynamic domains such as scheduling. Finally, we are interested in developing a better understanding of how aspects of problem structure affect the ability of stochastic search to amplify heuristic performance in optimization domains.

Acknowledgements

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 the CMU Robotics Institute. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force or U.S. Government.

References

1. E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems.* Santa Fe Institute Studies in the Sciences of Complexity. Oxford University Press, 1999.
2. J. L. Bresina. 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*, pages 271–278. AAAI Press, 1996.
3. A. Cesta, A. Oddi, and S. F. Smith. An iterative sampling procedure for resource constrained project scheduling with time windows. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1022–1029. Morgan Kaufmann, 1999.
4. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* McGraw-Hill, 1990.
5. M. Dorigo and G. Di Caro. The ant colony optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization.* McGraw-Hill, 1999.
6. C. Gomes, B. Selman, and H. Kautz. Boosting combinatorial search through randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference*, pages 431–437. AAAI Press, 1998.
7. W. Harvey and M. Ginsberg. Limited discrepency search. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 607–613. Morgan Kaufmann, 1995.
8. Y. H. Lee, K. Bhaskaran, and M. Pinedo. A heuristic to minimize the total weighted tardiness with sequence-dependent setups. *IIE Transactions*, 29:45–52, 1997.
9. T. E. Morton and D. W. Pentico. *Heuristic Scheduling Systems: With Applications to Production Systems and Project Management.* John Wiley and Sons, 1993.
10. A. Oddi and S. F. Smith. Stochastic procedures for generating feasible schedules. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference*, pages 308–314. AAAI Press, 1997.
11. G. Theraulaz, S. Goss, J. Gervet, and J. L. Deneubourg. Task differentiation in polistes wasp colonies: A model for self-organizing groups of robots. In *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pages 346–355. MIT Press, 1991.
12. T. Walsh. Depth-bounded discrepency search. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence.* Morgan Kaufmann, 1997.
13. J. P. Watson, L. Barbulescu, A. E. Howe, and L. D. Whitley. 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)*, pages 688–695. AAAI Press, 1999.

A Proof: Dominance Tournament = Roulette Wheel Decision

Dominance Tournament = Roulette Wheel Decision: The structure of the tournament of dominance contests within the WHISTLING algorithm is such

that Choice_w is chosen with probability $\frac{\text{bias(heuristic(Choice}_w))}{\sum_{j=1}^{k} \text{bias(heuristic(Choice}_j))}$. That is, the tournament is such that each Choice_w is chosen according to a roulette wheel decision where each Choice_w takes a chunk of the wheel proportional to $\text{bias(heuristic(Choice}_w))$.

Proof (by induction): In this proof, $F_w = \text{bias(heuristic(Choice}_w))$ is the initial value of the force of Choice_w. Further F'_w is the force of Choice_w including the accumulation of the force variables of other choices which it has defeated in the tournament.

Base Case: Dominance tournament of two choices.

- Choice_1 wins with probability $\frac{F_1}{F_1+F_2}$. If it wins, its new force is $F'_1 = F_1 + F_2$.
- Choice_2 wins with probability $\frac{F_2}{F_1+F_2}$. If it wins, its new force is $F'_2 = F_1 + F_2$.
- Therefore, for the base case of two choices we have the equivalent of a roulette wheel decision.

Inductive Step:

- Assume that for the case of a tournament of dominance contests of $k-1$ choices that this tournament is equivalent to a roulette wheel decision.
- Further assume that upon the completion of this tournament that the winning choice Choice_w has a force value of $F'_w = \sum_{j=1}^{k-1} F_j$.
- Now consider the addition of a k-th choice Choice_k at the end of the list of choices with an initial force F_k. The winner Choice_w of the sub-tournament consisting of the first $k-1$ choices would then compete in a dominance contest against Choice_k.
- Choice_k wins this dominance contest and thus the tournament of contests among the k choices with probability $\frac{F_k}{F_k+F'_w} = \frac{F_k}{F_k+\sum_{j=1}^{k-1} F_j} = \frac{F_k}{\sum_{j=1}^{k} F_j}$. If it wins, its new force is $F'_k = F_k + F'_w = \sum_{j=1}^{k} F_j$.
- Choice_w wins this dominance contest against Choice_k with probability $\frac{F'_w}{F_k+F'_w}$. Further, since it had a probability of $\frac{F_w}{F'_w}$ of winning the sub-tournament of the first $k-1$ choices, it therefore wins the overall tournament of k choices with probability $\frac{F_w}{F'_w}\frac{F'_w}{F_k+F'_w} = \frac{F_w}{\sum_{j=1}^{k} F_j}$. Its new force if it wins is $F''_w = F_k + F'_w = \sum_{j=1}^{k} F_j$.
- Therefore, for the case of a tournament of k choices, we have the equivalent of a roulette wheel decision.