Weighted Tardiness Scheduling with Sequence-Dependent Setups: A Benchmark Library

Vincent A. Cicirello

cicirello@cs.cmu.edu

February 2003

Intelligent Coordination and Logistics Laboratory
The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

This report details a set of benchmark problem instances that we have generated for the weighted tardiness scheduling with sequence-dependent setups problem. The version of the problem without setup times is NP-Hard. The problem is further complicated by the sequence-dependent nature of the setups. Given that the weighted tardiness objective and the sequence-dependent setups constraint are both common real-world problem characteristics, and given the difficulty in solving this problem, we feel that it is currently under-represented in the research literature. With few algorithms currently available for the problem, this first benchmark problem set and initial results obtained via a large number of heuristic algorithms will hopefully encourage others to explore additional solutions to the problem.

1 Introduction

This report details a set of benchmark problem instances that we have generated for the weighted tardiness scheduling with sequence-dependent setups problem. The version of the problem without setup times is NP-Hard. The problem is further complicated by the sequence-dependent nature of the setups. Given that the weighted tardiness objective and the sequence-dependent setups constraint are both common real-world problem characteristics, and given the difficulty in solving this problem, we feel that it is currently underrepresented in the research literature. With few algorithms currently available for the problem, this first benchmark problem set and initial results obtained via a large number of heuristic algorithms will hopefully encourage others to explore additional solutions to the problem. The problem set can be obtained at one of:

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

The remainder of this report details the formalism of the problem, the method of instance generation, the file format used for the problem instances. Also contained in this report is a description of several heuristic search algorithms used to find initial solutions to these problems to be used as a benchmark for any further algorithm development as well as a list of the current best known solutions to the instances.

Any further improvement you may obtain upon these solutions can be sent to: vincent+wtsbench@cs.cmu.edu. All correspondence should be in plain text format only, either in the body of the e-mail or as an attachment. It should include:

1. the problem id
2. the objective value of the new best solution
3. the sequence of job numbers that obtains the new best solution
4. a description of the algorithm used to obtain this new best
5. complete citations of (and/or webpage links to) any papers describing the algorithm used

You may also send citations to any papers that use this benchmark set regardless of whether or not you improve upon our solutions and we will maintain a bibliography of algorithms for the problem with the benchmarks.

2 Problem Formalism

This benchmark library consists of a set of weighted tardiness scheduling problems 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:

$$T = \sum_{j \in J} w_j T_j = \sum_{j \in J} w_j \max\left(c_j - d_j, 0\right), \tag{1}$$

where T_j is the tardiness of job j; w_j, c_j, d_j are the weight, completion 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. The completion time c_j of a job can be defined formally as:

$$c_j = \sum_{i \in \text{Predecessors}(j) \cup j} p_i + s_{\text{Previous}(i),i} \tag{2}$$

where p_i, $s_{k,i}$ are the process time of job i and the setup time of job i if it immediately follows job k, respectively. Predecessors(j) is the set of all jobs that come before job j in the sequence and Previous(i) is the single job that immediately precedes job i.

3 Instance Generation

The problem instances are generated according to Lee et al.'s procedure (Lee, Bhaskaran, & Pinedo, 1997).[1] 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.[2] 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.

4 Instance File Format

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

[1] Unfortunately, Lee et al. did not make their original problem instances available publicly.

[2] See Lee et al.'s paper for a defi nition of these parameters.

```
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
```

5 Benchmark Solvers

5.1 Dispatch Heuristics

In dynamic factory environments, dispatch scheduling heuristics provide a practical, robust basis for managing execution (Morton & Pentico, 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 can even exhibit formally chaotic tendencies (Kempf & Beaumariage, 1994; Beaumariage & Kempf, 1995).

To address the weighted tardiness scheduling with sequence-dependent setups problem, we take the *Apparent Tardiness Cost with Setups (ATCS)* dispatch heuristic (Lee et al., 1997). 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:

$$\text{ATCS}_j(t,l) = \frac{w_j}{p_j} \exp\left(-\frac{\max\left(d_j - p_j - t, 0\right)}{k_1 \bar{p}} - \frac{s_{lj}}{k_2 \bar{s}}\right), \tag{3}$$

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. k_1 and k_2 are parameters for tuning the heuristic. In our experiments, we set the values of these parameters according to Lee et al.'s original recommendations (Lee et al., 1997). When applied deterministically, the next job j added to the schedule using the ATCS heuristic is simply:

$$j = \arg\max_j \text{ATCS}_j(t,l)\ . \tag{4}$$

The algorithms that follow that require a search heuristic all use ATCS.

5.2 Hill-Climber

During the development of the heuristic ATCS, Lee et al. noted that the sequence that resulted from using the heuristic could often be improved significantly by moving jobs

short distances within the sequence. Given this, they applied a hill-climbing algorithm to the heuristic solution that considered this. The operator set consisted of taking the job that added the most to the objective value and considering swapping its position in the sequence with each of the 20 nearest jobs. It also consisted of considering removing this job and inserting it in front of the each of the 20 nearest jobs. From each search state, it evaluates these 40 possible moves and accepts the one that improves the objective value the most. It continues until it reaches a local optima.

5.3 Discrepancy Search

Harvey and Ginsberg considered 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) was designed with this rationale in mind (Harvey & 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 had similar motivation in the design of his Depth-bounded Discrepancy Search (DDS) (Walsh, 1997). Walsh, however, acknowledged the idea that search heuristics are often less-informed at the top of the search tree and often very-informed near the leaves of the tree. To deal with this, he combined 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 discrepancies at depth i or less in a manner as in LDS.

5.4 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 then we are done. Otherwise, if we are looking for the best solution we can find,

Algorithm 1: Iterative Sampling (IS)
Input: Number of iterations I; an "objective" function; and a set of tasks T_j to schedule.
Output: A schedule S.
IS(I, objective, tasks T_j)
(1) **repeat** I times
(2) $S \leftarrow$ the empty schedule
(3) **while** not all tasks scheduled
(4) **select** uniformly at random the next task T
(5) add this selected task to the candidate schedule S
(6) remove this selected task from the set of unscheduled tasks
(7) **if** objective(S) is superior to objective(bestsofar)
(8) bestsofar $\leftarrow S$
(9) **return** bestsofar

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. Algorithm 1 shows the iterative sampling algorithm for a generic scheduling problem.

5.5 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 the context of this scheduling problem 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. 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.

5.6 Value-Biased Stochastic Sampling

A fundamental 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:

$$P(c_i) = \frac{\text{bias}(\text{rank}(c_i, \text{heuristic}))}{\sum_j \text{bias}(\text{rank}(c_j, \text{heuristic}))} \tag{5}$$

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

[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 specifi c 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.

Algorithm 2: 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)
(1)    bestsofar ← DISPATCHSCHEDULING(T_j, heuristic)
(2)    repeat I times
(3)       S ← the empty schedule
(4)       while not all tasks scheduled
(5)          foreach unscheduled task T
(6)             score[T] ← heuristic(T, S)
(7)          sort all tasks T according to score[T]
(8)          foreach unscheduled task T
(9)             rank[T] ← sort position of T
(10)            weight[T] ← bias(rank[T])
(11)            totalweight ← totalweight + weight[T]
(12)         foreach unscheduled task T
(13)            prob[T] ← weight[T] / totalweight
(14)         select randomly the next task biased according to prob[T]
(15)            add this selected task to the candidate schedule S
(16)            remove this selected task from the set of unscheduled tasks
(17)      if objective(S) is superior to objective(bestsofar)
(18)         bestsofar ← S
(19)   return bestsofar
```

values are not used and thus not utilized to their full potential.

In order to fully utilize the discriminatory power of the heuristic inherent in the heuristic values, we have previously presented what we call value biased stochastic sampling (VBSS) (Cicirello & Smith, 2002, 2001). In VBSS, decisions are again made in a manner analogous to a roulette wheel, but choice c_i is now made with probability:

$$P(c_i) = \frac{\text{bias}(\text{heuristic}(c_i))}{\sum_j \text{bias}(\text{heuristic}(c_j))} \tag{6}$$

VBSS is detailed 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

Algorithm 3: Value-Biased Stochastic Sampling (VBSS)
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.

```
VBSS(I, heuristic, bias, objective, tasks T_j)
(1)   bestsofar ← DISPATCHSCHEDULING(T_j, heuristic)
(2)   repeat I times
(3)     S ← the empty schedule
(4)     while not all tasks scheduled
(5)       foreach unscheduled task T
(6)         score[T] ← heuristic(T, S)
(7)         weight[T] ← bias(score[T])
(8)         totalweight ← totalweight + weight[T]
(9)       foreach unscheduled task T
(10)        prob[T] ← weight[T] / totalweight
(11)      select randomly the next task biased according to prob[T]
(12)        add this selected task to the candidate schedule S
(13)        remove this selected task from the set of unscheduled tasks
(14)    if objective(S) is superior to objective(bestsofar)
(15)      bestsofar ← S
(16)  return bestsofar
```

the choices is much more heavily 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 same probability of being made in both contexts. Whereas, in VBSS, 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-based approach, there is a much higher probability of choosing the heuristic preferred choice as compared to the other decision context; while the rank-based approach treats both decision contexts in the exact same way.

Computationally, the VBSS algorithm selects the next task to schedule in $O(T)$ time

	Context 1		Context 2	
	Choice 1	Choice 2	Choice 1	Choice 2
	H1 = 10	H2 = 11	H1 = 10	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.

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, 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, we can see that VBSS is asymptotically more efficient than Bresina's approach. 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[5]. And with T tasks to schedule, the algorithmic complexity of HBSS is $O(T^2 \log T)$.

[5]It has been pointed out that the worst-case complexity of choosing the i-th largest element from an unsorted list is $O(n)$. 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-fi rst-then-select algorithm dominating for all but very large problem instances (Cormen, Leiserson, & Rivest, 1990).

5.7 Random-Restart Hill-Climbing

We have taken Lee et al.'s hill-climber and created a multi-start version by using VBSS and the ATCS heuristic to generate random starting solutions which are then improved upon by the local hill-climb.

6 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: This is LDS (Limited Discrepancy Search) allowed to run long enough to consider all two-discrepancy solutions.

- HBSS,<iterations>,<bias>: This is HBSS with the specified number of iterations and a bias function equal to $\text{rank}^{-1*\text{bias}}$.

- VBSS,<iterations>,<bias>: This is VBSS with the specified number of iterations and a bias function equal to $\text{value}^{\text{bias}}$.

- VBSS-HC,<iterations>,<bias>: This is the random-restart hill-climber using VBSS to seed the starting solutions for the specified number of iterations and a bias function equal to $\text{value}^{\text{bias}}$.

The 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	8978	VBSS-HC,10000,5
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	2217	VBSS-HC,5000,5
25	27	VBSS-HC,5000,5
26	0	LDS-all-two
27	271	VBSS-HC,5000,5
28	384	VBSS-HC,10000,5
29	0	LDS-all-two
30	638	VBSS,200,8

Problem Instance	Objective Value	Algorithm
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	77242	VBSS-HC,10000,5
42	61859	VBSS-HC,2500,5
43	149990	LDS-all-two
44	40096	VBSS-HC,2500,5
45	62760	VBSS,100,9
46	39214	VBSS-HC,10000,5
47	79039	VBSS-HC,5000,5
48	68920	LDS-all-two
49	84811	VBSS,2500,5
50	37437	VBSS-HC,5000,5
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	90332	LDS-all-two
57	70414	VBSS,10000,5
58	55522	VBSS-HC,10000,5
59	59060	VBSS-HC,10000,5
60	73328	VBSS-HC,10000,5

Problem Instance	Objective Value	Algorithm
61	81837	LDS-all-two
62	50632	VBSS-HC,2500,5
63	78822	VBSS,200,23
64	100534	HBSS,200,4
65	135170	VBSS-HC,1000,5
66	66415	VBSS-HC,10000,5
67	36172	VBSS-HC,2500,5
68	27379	VBSS-HC,1000,5
69	79300	VBSS-HC,2500,5
70	82192	LDS-all-two
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	44343	VBSS-HC,5000,5
78	28839	VBSS-HC,5000,5
79	125824	VBSS,5000,5
80	31844	VBSS,10000,5
81	387148	VBSS,200,17
82	413488	VBSS,200,17
83	468131	VBSS,100,18
84	331659	VBSS,100,20
85	563884	VBSS,100,16
86	365783	VBSS,200,20
87	404206	VBSS-HC,5000,5
88	436855	VBSS,200,11
89	416916	VBSS-HC,5000,5
90	406939	LDS-all-two

Problem Instance	Objective Value	Algorithm
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
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

References

Beaumariage, T., & Kempf, K. (1995). Attractors in manufacturing systems with chaotic tendencies.. Presentation at INFORMS-95, New Orleans, http://www.informs.org/Conf/NewOrleans95/ TALKS/TB07.3.html.

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*, pp. 271–278. AAAI Press.

Cicirello, V. A., & Smith, S. F. (2001). Randomizing dispatch scheduling policies. In *Using Uncertainty Within Computation: Papers from the 2001 AAAI Fall Symposium, Technical Report FS-01-04*, pp. 30–37. AAAI Press. North Falmouth, Massachusetts.

Cicirello, V. A., & Smith, S. F. (2002). Amplifi cation of search performance through randomization of heuristics. In Van Hentenryck, P. (Ed.), *Principles and Practice of Constraint Programming – CP 2002: 8th International Conference, Proceedings*, Vol. LNCS 2470 of *Lecture Notes in Computer Science*, pp. 124–138. Springer-Verlag. Ithaca, NY.

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

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

Kempf, K., & Beaumariage, T. (1994). Chaotic behavior in manufacturing systems. In *AAAI-94 Workshop Program, Reasoning About the Shop Floor, Workshop Notes*, pp. 82–96. AAAI Press.

Korf, R. (1985). Depth-fi rst 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, Volume 1*, pp. 286–291. AAAI Press.

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., Bhaskaran, K., & Pinedo, M. (1997). A heuristic to minimize the total weighted tardiness with sequence-dependent setups. *IIE Transactions, 29*, 45–52.

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

Walsh, T. (1997). Depth-bounded discrepency search. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pp. 1388–1395. Morgan Kaufmann.