Weighted Tardiness Scheduling with Sequence-Dependent Setups: A Benchmark Problem for Soft Computing

Vincent A. Cicirello

Computer Science and Information Systems
The Richard Stockton College of New Jersey
Pomona, NJ 08240
cicirelv@stockton.edu

Summary. In this paper we present a set of benchmark instances and a benchmark instance generator for a single-machine scheduling problem known as the weighted tardiness scheduling problem with sequence-dependent setups. Furthermore, we argue that it is an ideal benchmark problem for soft computing in that it is computationally hard and does not lend itself well to exact solution procedures. Additionally, it has a number of important real world applications.

1 Introduction

In this paper we present a set of benchmark instances and a benchmark instance generator for a single-machine scheduling problem known as the weighted tardiness scheduling problem with sequence-dependent setups. Furthermore, we argue that it is an ideal benchmark problem for soft computing in that it is computationally hard and does not lend itself well to exact solution procedures. Additionally, it has a number of important real world applications.

The weighted tardiness problem is encountered in a number of real-world applications, including turbine component manufacturing [3], the packaging industry [1], among others [15]. It is a scheduling objective function that Morton and Pentico indicate to be very hard even if setups are independent of job sequence [15]. Exact optimal solutions can be found by branch-and-bound for instances of at most 100 jobs (e.g., [17]), but are considered impractical for instances that are larger than this [16]. For larger instances, heuristic or metaheuristic approaches are preferred (e.g., [9, 10, 16, 18, 11]).

E. Avineri, M. Koppen, K. Dahal, Y. Sunitiyoso, R. Roy (Eds.): Applications of Soft Computing: Updating the State of the Art, Advances in Intelligent and Soft Computing 52, pp. 201–210, 2009.

What truly makes the problem a challenge problem is if you consider the case with sequence-dependent setups. *Setup time* refers to a length of time that must be spent preparing a machine prior to processing a job [15]. If all jobs are identical, or if the setup time only depends on the job that the machine is being setup for, but not on the previously processed job, then we can say that the setups are *sequence-independent*. If the setups are sequence-independent, then the problem can be transformed to essentially remove them from the problem (e.g., adding the setup times to the process times in some way). When setup time of a job depends on the job that is processed immediately before it on the machine then the setups are *sequence-dependent*. Allahverdi et al as well as Zhu and Wilhelm offer comprehensive reviews of these and other considerations pertaining to setup costs [2, 22].

Sequence-dependent setups commonly appear in real-world scheduling problems (e.g., [7, 1, 3, 14, 13]). Unfortunately, however, they are often ignored during the development of algorithms. The vast majority of work on sequencing problems assume that setups are sequence-independent, usually without acknowledging the possibility that they may be a factor in the problem. Sen and Bagchi discuss the significance of the challenge that sequence-dependent setups pose for exact solution procedures [21]. Specifically, they discuss how sequence-dependent setups induce a non-order-preserving property of the evaluation function. At the time of their writing, exact solution procedures such as A*, Branch-and-Bound algorithms, or GREC [21] for sequencing problems with sequence-dependent setups were limited to solving instances with no more than approximately 25-30 jobs, even for easier objective functions. Problem instances of larger size require turning to soft computing approaches.

Although there are exact approaches for optimizing weighted tardiness when setups are independent of sequence (or non-existent), all current approaches for the problem when setups are sequence-dependent are either heuristic or metaheuristic. For example, there are some dispatch scheduling heuristics for the sequence-dependent setup version of the weighted tardiness problem such as ATCS [12] as well as the heuristic of [20]. Both of these are rather ad hoc modifications of the well-known R&M dispatch policy [19] for the setup-free version of the problem. Additionally, there have been several recent metaheuristics for the problem. Lee et al suggested a local hill climbing algorithm to apply to the solution given by their ATCS dispatch policy [12]. Cicirello and Smith developed a value-biased stochastic sampling algorithm to expand the search around ATCS; and also benchmarked their approach with several other heuristic search algorithms [8]. Most recently, a permutation-based genetic algorithm using the Non-Wrapping Order Crossover operator [5] and a simulated annealing algorithm [6] have both improved upon a number of the best known solutions to several benchmark instances.

2 Problem Formulation

The weighted tardiness scheduling problem with sequence-dependent setups can be defined as follows. The problem instance consists of 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 job j if it follows job i on the machine. It is not necessarily the case that $s_{i,j} = s_{j,i}$. The 0-th "job" is the start of the problem ($p_0 = 0$, $d_0 = 0$, $s_{i,0} = 0$, $w_0 = 0$). Its purpose is for the specification of the setup time of each of the jobs if sequenced first.

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

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

where T_j is the tardiness of job j; and c_j, d_j are the completion time and duedate of job j. The completion time of a job 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 the job itself. Specifically, let $\pi(j)$ be the position in the sequence of job j. Define c_j as:

$$c_j = \sum_{i,k \in J, \pi(i) <= \pi(j), \pi(i) = \pi(k)+1} p_i + s_{k,i}. \tag{2}$$

3 Weighted Tardiness Problem Instance Generator

Previously, the author implemented a problem instance generator for the weighted tardiness scheduling problem with sequence-dependent setups [4]. This instance generator is an implementation of a procedure described by Lee et al and used in the analysis of Lee et al's dispatch scheduling policy ATCS [12]. Cicirello's instance generator has since been refined and reimplemented in Java and is available on the web (http://loki.stockton.edu/~cicirelv/benchmarks/).

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. These parameters are defined as follows:

$$\tau = 1 - \frac{\bar{d}}{C_{\max}} \tag{3}$$

$$R = \frac{d_{\max} - d_{\min}}{C_{\max}} \tag{4}$$

$$\eta = \frac{\bar{s}}{\bar{p}} \tag{5}$$

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 optimal 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.

Lee et al provide experimental data for setting the value of β for 4 different size problems (20, 40, 60, and 80 job instances). Cicirello's original implementation of the instance generator was restricted to generating instances of those 4 sizes. One of the refinements of the new Java implementation is fitting a curve to Lee et al's reported data to extrapolate appropriate values of β for other size problem instances. Specifically, a small 3 node feedforward neural net (2 hidden sigmoid nodes, 1 output sigmoid) with the number of jobs, N, as input was fitted to the data through a least-squares fit. The result is the following definition of β as a function of the number of jobs:

$$\beta(N) = \frac{1}{1 + e^{(1.0949132 - 1971.6253 \cdot A(N) - 8.1243637 \cdot B(N))}}, \tag{6}$$

with

$$A(N) = \frac{1}{1 + e^{(7.168150953 + 0.040112027 \cdot N)}} \tag{7}$$

and

$$B(N) = \frac{1}{1 + e^{(-10.58867025 + 2.400027877 \cdot N)}}. \tag{8}$$

The processing times of the jobs of the instances produced by the generator 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 $[0, 10]$.

4 Weighted Tardiness Benchmark Instances

In addition to the instance generator, the set of benchmark problem instances used by [4, 8, 5, 6] among others, are available on the web (http://loki.stockton.edu/~cicirelv/benchmarks/). This benchmark set includes 120 problem instances with 60 jobs each. The problem set is 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, there are 10 problem instances. Generally speaking, these 12 problem sets cover a spectrum from loosely to tightly constrained problem instances.

File Format.

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

```
Problem Instance: <instance number>
Problem Size: <number of jobs>
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: <makespan estimate>
D_bar: <average duedate>
End Generator Parameters
Begin Problem Specification
Process Times:
<process time for job 0>
...
<process time for job n-1>
Weights:
<weight for job 0>
...
<weight for job n-1>
Duedates:
<duedate for job 0>
...
<duedate for job n-1>
Setup Times:
<i> <j> <setup for job j if after i>
// i=-1 indicates setup if j is first
End Problem Specification
```

Current Best Known Solutions.

Tables 1(a), 1(b), 2(a), 2(b), 3(a), 3(b), 4(a), 4(b), 5(a), 5(b), 6(a), and 6(b) list the current best known solutions to the benchmark instances. Specifically, each table lists the instance number, the current best known solution (for the weighted tardiness objective), and the algorithm that first found that solution. Note that other algorithms may also have found that solution. We simply list here the first one that did. Algorithms are abbreviated as follows:

- SA-H: A recent simulated annealing algorithm [6].

Table 1. Current best known solutions for instances with: (a) Loose Duedates, Narrow Duedate Range, Mild Setups; and (b) Loose Duedates, Narrow Duedate Range, Severe Setups

(a)

Instance	Best	First Found By
1	790	SA-H
2	5824	SA-H
3	1936	GA
4	6840	SA-H
5	5017	SA-R
6	7824	SA-H
7	3933	SA-H
8	298	SA-R
9	7059	SA-H
10	2125	SA-H

(b)

Instance	Best	First Found By
11	5088	SA-H
12	0	LDS
13	6147	VBSS-HC
14	3761	SA-R
15	2039	SA-R
16	5559	SA-R
17	387	SA-R
18	1918	SA-R
19	239	SA-H
20	3805	SA-R

- SA-R: A second version of that recent simulated annealing algorithm [6].
- GA: A permutation-based genetic algorithm using the Non-Wrapping Order Crossover operator and an insertion mutator [5].
- VBSS-HC: A multistart hill climber seeded with starting configurations generated by Value Biased Stochastic Sampling [8].
- VBSS: Value Biased Stochastic Sampling [8].
- LDS: Limited Discrepancy Search truncated after exhausting all search trajectories with 2 or less discrepancies from the ATCS heuristic solution [8]. It was used in the original empirical evaluation of VBSS and VBSS-HC.
- SA: A simulated annealing algorithm also used in the original empirical evaluation of VBSS and VBSS-HC [8].

The problem instances and the best known solutions to them are organized according to the 12 classes of instances as follows:

- Table 1(a): Loose Duedates, Narrow Duedate Range, Mild Setups
- Table 1(b): Loose Duedates, Narrow Duedate Range, Severe Setups
- Table 2(a): Loose Duedates, Wide Duedate Range, Mild Setups
- Table 2(b): Loose Duedates, Wide Duedate Range, Severe Setups
- Table 3(a): Moderate Duedates, Narrow Duedate Range, Mild Setups
- Table 3(b): Moderate Duedates, Narrow Duedate Range, Severe Setups
- Table 4(a): Moderate Duedates, Wide Duedate Range, Mild Setups
- Table 4(b): Moderate Duedates, Wide Duedate Range, Severe Setups
- Table 5(a): Tight Duedates, Narrow Duedate Range, Mild Setups
- Table 5(b): Tight Duedates, Narrow Duedate Range, Severe Setups
- Table 6(a): Tight Duedates, Wide Duedate Range, Mild Setups
- Table 6(b): Tight Duedates, Wide Duedate Range, Severe Setups

Table 2. Current best known solutions for instances with: (a) Loose Duedates, Wide Duedate Range, Mild Setups; and (b) Loose Duedates, Wide Duedate Range, Severe Setups

(a)

Instance	Best	First Found By
21	0	LDS
22	0	LDS
23	0	LDS
24	1092	SA-H
25	0	SA
26	0	LDS
27	57	SA-R
28	0	GA
29	0	LDS
30	215	SA-R

(b)

Instance	Best	First Found By
31	0	LDS
32	0	LDS
33	0	LDS
34	0	LDS
35	0	LDS
36	0	LDS
37	1008	SA-R
38	0	LDS
39	0	LDS
40	0	LDS

Table 3. Current best known solutions for instances with: (a) Moderate Duedates, Narrow Duedate Range, Mild Setups; and (b) Moderate Duedates, Narrow Duedate Range, Severe Setups

(a)

Instance	Best	First Found By
41	71242	SA-H
42	59493	SA-H
43	147737	SA-H
44	36265	SA-H
45	59696	SA-R
46	36175	SA-R
47	74389	SA-H
48	65129	SA-R
49	79656	SA-R
50	32777	SA-R

(b)

Instance	Best	First Found By
51	54707	SA-H
52	100793	SA-H
53	94394	SA-R
54	123558	VBSS
55	72420	SA-R
56	80258	SA-H
57	68535	SA-H
58	46978	SA-R
59	56181	SA-H
60	68395	SA-R

In this set of benchmark instances, loose duedates, moderate duedates, and tight duedates refer to values of the duedate tightness factor τ of 0.3, 0.6, and 0.9, respectively. Narrow duedate range and wide duedate range refer to values of the duedate range factor of 0.25 and 0.75, respectively. Mild setups and severe setups refer to values of the setup time severity factor η of 0.25 and 0.75, respectively.

5 Conclusions

In this paper, we presented a set of benchmark instances and a problem instance generator for a computationally hard scheduling problem known as the

Table 4. Current best known solutions for instances with: (a) Moderate Duedates, Wide Duedate Range, Mild Setups; and (b) Moderate Duedates, Wide Duedate Range, Severe Setups

(a)

Instance	Best	First Found By
61	76769	SA-R
62	44781	SA-H
63	76059	SA-H
64	93079	SA-H
65	127713	SA-R
66	59717	SA-H
67	29394	SA-R
68	22653	SA-R
69	71534	SA-H
70	76140	SA-R

(b)

Instance	Best	First Found By
71	155036	SA-R
72	49886	SA-H
73	30259	SA-H
74	32083	SA-R
75	21602	SA-R
76	57593	SA-H
77	35380	SA-H
78	21443	SA-H
79	121434	SA-H
80	20221	SA-H

Table 5. Current best known solutions for instances with: (a) Tight Duedates, Narrow Duedate Range, Mild Setups; and (b) Tight Duedates, Narrow Duedate Range, Severe Setups

(a)

Instance	Best	First Found By
81	385918	SA-H
82	410550	SA-H
83	459939	SA-R
84	330186	SA-R
85	557831	SA-R
86	364474	SA-R
87	400264	SA-R
88	434176	SA-R
89	411810	SA-H
90	403623	SA-R

(b)

Instance	Best	First Found By
91	344428	SA-R
92	363388	SA-R
93	410462	VBSS
94	334180	SA-H
95	524463	SA-R
96	464403	LDS
97	418995	SA-H
98	532519	VBSS
99	374607	SA-R
100	441888	VBSS-HC

weighted tardiness scheduling problem with sequence-dependent setups. Two characteristics of the problem make it ideal for benchmarking soft computing algorithms. First, the sequencing jobs on a single machine to optimize this objective function is NP-Hard even in the setup-free version. Exact approaches are currently limited to instances with no more than 100 jobs; and are infeasible beyond that size. Second, sequence-dependent setups greatly magnify the problem difficulty. It has been shown by others that sequence-dependent setups, even for easier objective functions than weighted tardiness, induce a non-order-preserving property for exact approaches, largely limiting them to solving instances smaller than approximately 30 jobs. Even moderately sized instances require metaheuristics and other soft computing approaches.

Table 6. Current best known solutions for instances with: (a) Tight Duedates, Wide Duedate Range, Mild Setups; and (b) Tight Duedates, Wide Duedate Range, Severe Setups

(a)

Instance	Best	First Found By
101	353575	SA-R
102	495094	SA-H
103	380170	VBSS
104	358738	SA-R
105	450806	SA-R
106	457284	SA-H
107	353564	SA-H
108	462675	SA-R
109	413918	SA-H
110	419014	SA-R

(b)

Instance	Best	First Found By
111	348796	SA-H
112	375952	SA-H
113	261795	SA-H
114	471422	SA-H
115	460225	VBSS
116	537593	SA-H
117	507188	SA-H
118	357575	LDS
119	581119	SA-H
120	399700	VBSS-HC

References

1. L. Adler, N. M. Fraiman, E. Kobacker, M. Pinedo, J. C. Plotnitcoff, and T. P. Wu. BPSS: a scheduling system for the packaging industry. *Operations Research*, 41:641–648, 1993.
2. A. Allahverdi, J.N.D. Guptab, and T. Aldowaisan. A review of scheduling research involving setup considerations. *Omega: International Journal of Management Science*, 27:219–239, 1999.
3. W. Y. Chiang, M. S. Fox, and P. S. Ow. Factory model and test data descriptions: OPIS experiments. Technical Report CMU-RI-TR-90-05, The Robotics Institute, Carnegie Mellon University, March 1990.
4. Vincent A. Cicirello. *Boosting Stochastic Problem Solvers Through Online Self-Analysis of Performance*. PhD thesis, The Robotics Institute, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 21 July 2003. Also available as technical report CMU-RI-TR-03-27.
5. Vincent A. Cicirello. Non-wrapping order crossover: An order preserving crossover operator that respects absolute position. In M. Keijzer et al, editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'06)*, volume 2, pages 1125–1131. ACM Press, 8-12 July 2006.
6. Vincent A. Cicirello. On the design of an adaptive simulated annealing algorithm. In *Proceedings of the CP 2007 First International Workshop on Autonomous Search*, 23 September 2007.
7. Vincent A. Cicirello and Stephen F. Smith. Wasp-like agents for distributed factory coordination. *Journal of Autonomous Agents and Multi-Agent Systems*, 8(3):237–266, May 2004.
8. Vincent A. Cicirello and Stephen F. Smith. Enhancing stochastic search performance by value-biased randomization of heuristics. *Journal of Heuristics*, 11(1):5–34, January 2005.
9. Vincent A. Cicirello and Stephen F. Smith. The max k-armed bandit: A new model of exploration applied to search heuristic selection. In M. M. Veloso and

S. Kambhampati, editors, *The Proceedings of the Twentieth National Conference on Artificial Intelligence*, volume 3, pages 1355–1361. AAAI Press, 9-13 July 2005.

10. R. K. Congram, C. N. Potts, and S. L. van de Velde. An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, 14(1):52–67, Winter 2002.
11. H. A. J. Crauwels, C. N. Potts, and L. N. Van Wassenhove. Local search heuristics for the single machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, 10(3):341–350, Summer 1998.
12. 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.
13. D. Morley. Painting trucks at general motors: The effectiveness of a complexity-based approach. In *Embracing Complexity: Exploring the Application of Complex Adaptive Systems to Business*, pages 53–58. The Ernst and Young Center for Business Innovation, 1996.
14. D. Morley and C. Schelberg. An analysis of a plant-specific dynamic scheduler. In *Proceedings of the NSF Workshop on Intelligent, Dynamic Scheduling for Manufacturing Systems*, pages 115–122, June 1993.
15. T. E. Morton and D. W. Pentico. *Heuristic Scheduling Systems: With Applications to Production Systems and Project Management*. John Wiley and Sons, 1993.
16. V. Narayan, T. Morton, and P. Ramnath. X-Dispatch methods for weighted tardiness job shops. GSIA Working Paper 1994-14, Carnegie Mellon University, Pittsburgh, PA, July 1994.
17. C. N. Potts and L. N. van Wassenhove. A branch and bound algorithm for the total weighted tardiness problem. *Operations Research*, 33(2):363–377, 1985.
18. C. N. Potts and L. N. Van Wassenhove. Single machine tardiness sequencing heuristics. *IIE Transactions*, 23(4):346–354, December 1991.
19. R. V. Rachamadugu and T. E. Morton. Myopic heuristics for the single machine weighted tardiness problem. Working Paper 30-82-83, GSIA, Carnegie Mellon University, Pittsburgh, PA, 1982.
20. N. Raman, R. V. Rachamadugu, and F. B. Talbot. Real time scheduling of an automated manufacturing center. *European Journal of Operational Research*, 40:222–242, 1989.
21. A. K. Sen and A. Bagchi. Graph search methods for non-order-preserving evaluation functions: Applications to job sequencing problems. *Artificial Intelligence*, 86(1):43–73, September 1996.
22. X. Zhu and W. E. Wilhelm. Scheduling and lot sizing with sequence-dependent setup: A literature review. *IIE Transactions*, 38(11):987–1007, 2006.