A Simple, Fast, and Effective Heuristic for the Single-Machine Total Weighted Tardiness Problem

Halil Şen, Kerem Bülbül

Sabancı University, Manufacturing Systems and Industrial Engineering, İstanbul, Turkey
halilsen, bulbul@sabanciuniv.edu

Keywords: Preemptive relaxation, transportation problem, nested schedule.

1 Introduction

We consider the single-machine total weighted tardiness problem (TWT) where a set of n jobs with general weights $w_1, \ldots, w_n$, integer processing times $p_1, \ldots, p_n$, and integer due dates $d_1, \ldots, d_n$ has to be scheduled non-preemptively. If C_j is the completion time of job j then $T_j = \max(0,\ C_j - d_j)$ denotes the tardiness of this job. The objective is to find a schedule S^*_{WT} that minimizes the weighted sum of the tardiness costs of all jobs computed as $\sum_{j=1}^{n} w_j T_j$. This problem is known to be unary $\mathcal{NP}$-hard. Our goal is to design a constructive heuristic for this problem that yields excellent feasible solutions in short computational times by exploiting the structural properties of a preemptive relaxation.

Although the initial studies on TWT were conducted more than five decades ago, e.g., by McNaughton (1959), the topic is still challenging for ongoing research. Studies related to this topic cover both exact algorithms and heuristics. At present, the best exact algorithm is the SSDP algorithm of Tanaka, Fujikuma and Araki (2009); they are able to solve up to 300-job instances in 350 seconds on average and standard benchmark instances with 100 jobs are solved to optimality in an average of 6.42 seconds with a maximum of 39 seconds. In the domain of heuristics, the best contender is the iterated generalized pairwise interchange based dynasearch (GPI-DS) of Grosso, Croce and Tadei (2004). GPI-DS identifies the optimal solutions of 100-job instances in an average of 0.11 seconds with a maximum of 3.91 seconds.

We first develop a family of preemptive lower bounds for TWT and explore its structural properties. Then, we show that the solution corresponding to the least tight lower bound among those investigated features some desirable properties that can be exploited to build excellent feasible solutions to the original non-preemptive problem in short computational times. The aim is to develop a non-parametric, easy to implement, constructive heuristic which is fast, efficient and can form a basis for local search heuristics. Moreover, the structure of the preemptive relaxation allows us to obtain several non-preemptive solutions starting from the same preemptive solution. Thus, we reckon that our approach may be employed to form an initial solution pool for population-based metaheuristics. We present results on standard benchmark instances from the literature.

2 Proposed Algorithm

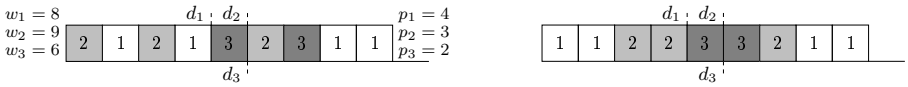
McNaughton (1959, in Theorem 2.2) shows that preemption in the classical sense, which allows that a job is split into any number of parts of arbitrary length and only penalizes the tardiness of the last part of the job, results in a problem as hard as TWT. However, if we prescribe that jobs may *only* be preempted at integer points in time by breaking a job j into p_j unit-jobs, and then assign a tardiness cost to the completion time of each unit-job, then this preemptive relaxation of TWT boils down to a transportation problem (TR). The size of TR is pseudo-polynomial because we have $P = \sum_{j=1}^{n} p_j$ unit-jobs; however, in practice

transportation problems can be solved very efficiently. Similar preemptive lower bounds were proposed for the single-machine earliness/tardiness problem by Bülbül, Kaminsky and Yano (2007), Sourd and Kedad-Sidhoum (2003).

In the presentation below, a feasible solution (schedule) of the transportation problem is denoted by S_{TR}, where an optimal solution is marked by an $*$ in the superscript. $S_{\mathrm{TR}}(t), t = 1, \ldots, P$, represents the job processed in period t and $j(t_1, t_2)$ is the ordered set of all time periods $t_1 \leq t \leq t_2$ so that $S_{\mathrm{TR}}(t) = j$.

Definition 1. *A job j is said to be preempted by job k at time t_1, if there exist two time periods t_1 and t_2 such that $1 \leq t_1 < t_2 < P$, $S_{\mathrm{TR}}(t_1) = j$, $S_{\mathrm{TR}}(t_2) = k$, $\mid j(t_1+1, t_2-1) \mid = 0$, and $\mid j(t_2+1, P) \mid \geq 1$. A feasible schedule S_{TR} for TR is said to be preemptive, if it contains at least one preempted job.*

In other words, if at least one unit-job of job k appears between two successive unit-jobs of job j, then job k is said to preempt job j. Under this definition, job k may preempt job j even if these two jobs are never processed in two adjacent time slots.



(a) (b)

Fig. 1. Structure of preemptions

Our solution approach relies on the idea that the second type of preemptive relaxation of TWT as discussed above has some desirable properties. Both schedules in Fig. 1 are preemptive but the schedule in Fig. 1(b) has a special structure which can be useful for building non-preemptive schedules easily. In the schedule in Fig. 1(a) job 1 preempts job 2 and then gets preempted by job 2. On the contrary, in the other schedule a job does not resume processing until the job that preempts it completes processing. Preemptive schedules with this property are called as *"nested"*. To ensure that the set of optimal solutions of TR includes at least one nested schedule, the objective function cost coefficients in the TR problem must be set diligently.

2.1 Cost Coefficients in the Transportation Problem

When we study the structure of the preemptions in an optimal solution of TR, we observe that a job with a higher priority may preempt jobs with lower priority. This priority is determined by the cost coefficients of the unit-jobs in TR.

To obtain a nested structure similar to that in Fig. 1(b), we need to determine appropriate objective coefficients in TR. Since preemption is related to the priority between two jobs, we have to select the cost coefficients in such a way that, a lower priority job is preempted by a higher priority job at most once. This property can be ensured by selecting cost coefficients that lie on a piecewise linear convex function with a single breakpoint.

The idea underlying our proposed cost structure is intuitive. Each unit-duration portion of a job j has a cost coefficient proportional to the ratio of the unit tardiness weight of job j to its processing time. That is, a unit-job of job j processed during the time interval $(t-1, t]$ incurs a cost of $c_{jt} = \frac{w_j}{p_j} \max(0, t - d_j)$. The proof that the optimal solution of TR with these coefficients provides a lower bound on the optimal objective function value of TWT is provided by Şen (2010, in Theorem 3.4). Alternate cost coefficients were proposed by Bülbül *et. al.* (2007), Sourd and Kedad-Sidhoum (2003). While these coefficients generally result in tighter lower bounds from TR, the set of optimal solutions may not contain any nested solution in these cases. Details are provided by Şen (2010).

2.2 Nesting Algorithm (NA)

In this section, we present the Nesting Algorithm. Şen (2010, in Lemma 3.5) shows that it converts any feasible schedule S_{TR} of TR into a feasible schedule S'_{TR} with no larger cost. Moreover, NA constructs a nested optimal schedule S'_{TR} when applied to any optimal schedule S^*_{TR} (Şen 2010, in Theorem 3.10). A direct corollary of this result is that there exists a nested optimal solution to TR under our cost coefficients.

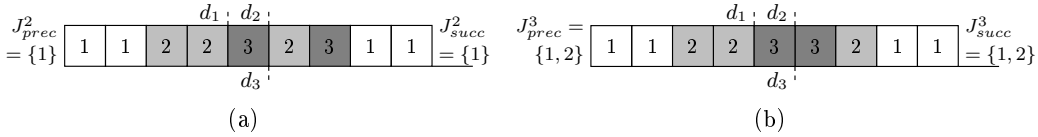


Fig. 2. Nesting algorithm

NA performs two types of tasks for each job j. First, it rearranges the current schedule so that the unit-jobs of job j succeed all unit-jobs of the jobs with no larger due dates over the time periods $1, \ldots, d_j$ (Şen 2010, in steps 3-11 of Algorithm 1). We denote this set of jobs by $J^j_{prec} = \{k|\ k < j\}$, where we assume that the jobs are sorted and re-labeled in non-decreasing order of their due dates in the rest of our presentation. Second, we define $J^j_{succ} = \{k|\ \frac{w_j}{p_j} > \frac{w_k}{p_k}\} \cup \{k < j|\ \frac{w_j}{p_j} = \frac{w_k}{p_k}\}$, and NA ensures that the unit-jobs of the jobs in J^j_{succ} appear following all unit-jobs of job j over the time periods $d_j + 1, \ldots, P$ (Şen 2010, in steps 12-20 of Algorithm 1). When applied to the optimal solution in Fig. 1(a), NA takes no action for the first job. In the iteration for job 2, the unit-jobs of jobs 1 and 2 are re-arranged over the first four time periods as required (see Fig. 2(a)). In the last iteration, the schedule is updated so that the unit-jobs of job 3 precede those of 1 and 2 from period $d_3 + 1$ onward (see Fig. 2(b)).

2.3 Non-preemptive Heuristic (NPH)

A nested schedule may be denoted by a parentheses structure, where an opening and a closing parenthesis precede and succeed the first and last unit-jobs of a job, respectively. For instance, the representation of the schedule in Fig. 2(b) is (1 (2 (3) 2) 1). "Parenthesis j" refers to the partial schedule marked by the first and last unit-jobs of job j.

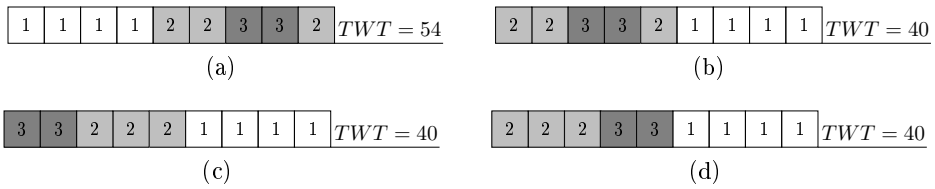


Fig. 3. Non-preemptive heuristic

In its basic form, when NPH detects a preempted job j, then it either schedules all unit-jobs of job j contiguously to either precede or follow all other unit jobs in parenthesis j. For each of these options, we sequence jobs in non-decreasing order of the completion times of their last unit-jobs, construct a non-preemptive schedule, and compute the total weighted tardiness. NPH continues iterating starting from the better alternative until it eventually provides us with a non-preemptive schedule. In Fig. 3, NPH is applied to the nested schedule in Fig. 2(b). For the preempted job 1, the two options are depicted in Fig. 3(a)-3(b). Starting from the better preemptive schedule in Fig. 3(b), NPH forms the two alternatives in Fig. 3(c)-3(d) for job 2 and terminates with two non-preemptive schedules with a total weighted tardiness of 40. Clearly, NPH may be embedded into a tree search leading to several non-preemptive solutions starting from a single nested preemptive solution. Thus, it may be employed to form an initial population in metaheuristics.

3 Computational Results and Discussion

In order to evaluate the performance of the proposed constructive heuristic, we solved the standard benchmark instances in the OR-Library[1] and the instances that were generated by Tanaka *et. al.* (2009)[2]. The algorithms were implemented in C++ using `CPLEX 12.1` Concert Technology. The runs were performed on a single core of a Windows PC with a 2.33 GHz Intel Core2 Quad CPU and 3.46 GB RAM.

For each setting of the number of jobs n, we report the average performance measures over 125 instances in Table 1. The corresponding worst case figures appear in parentheses. The column "# Opt." indicates the number of optimal solutions identified by NPH. The number of instances with optimality gaps larger than 1% and 10% appear in the last two columns, respectively.

Table 1. Summary of the results

n	CPU time*			# Opt.	%Gap		# %Gap	
	TR	NA	NPH	NPH	TR	NPH	>1%	>10%
40†	0.5 (0.9)	0.005 (0.016)	0.001 (0.016)	54	-14 (-96)	1.20 (42)	14	5
50†	0.7 (1.3)	0.009 (0.016)	0.002 (0.032)	47	-13 (-98)	0.42 (13)	9	1
100†	3 (6.3)	0.047 (0.079)	0.010 (0.188)	40	-10 (-99)	0.21 (8)	4	0
150‡	12 (20)	0.127 (0.188)	0.051 (0.500)	38	-8 (-91)	0.58 (56)	4	1
200‡	23 (41)	0.259 (0.407)	0.143 (1.750)	34	-7 (-98)	0.30 (13)	4	1
250‡	48 (101)	0.440 (0.656)	0.175 (2.391)	35	-6 (-95)	0.43 (41)	4	1
300‡	78 (154)	0.712 (1.047)	0.348 (4.031)	28	-6 (-91)	0.58 (28)	5	3

*: In seconds. †: OR-Library. ‡: Tanaka *et. al.* (2009).

The average optimality gap of NPH is less than 1% across the board except for $n = 40$. From the last two columns, we conclude that our solution approach consistently provides excellent feasible solutions. Instances for which the performance is less than stellar are quite few. The striking fact is that these non-preemptive solutions are obtained from relatively poor lower bounds. One potential issue is that the CPU times increase rapidly with more jobs and longer processing times. We revisit this issue further below.

For benchmarking purposes, we also implemented the dynasearch algorithm GPI-DS by Grosso *et. al.* (2004) mentioned in Section 1. In Table 2, we report the performance measures for GPI-DS on the same set instances as in Table 1. We did not have access to the original code by Grosso *et. al.* (2004), and the results in Table 2 are based on our own implementation and may differ slightly from those in the original paper. GPI-DS incorporates randomness, and it was run on each instance 15 times independently. In column "# of Iter.", we specify the iteration limit for GPI-DS, and in the next column "# of Opt." we report the number of optimal solutions identified for 125 instances averaged over 15 sets of runs. Columns 4-7 of Table 2 present statistics on the optimality gaps and the total solution times. The average/maximum number of iterations performed and the average/maximum time elapsed until the best solution is identified during the execution of GPI-DS appear in the last four columns.

The results in Table 2 attest to the excellent performance of GPI-DS. Up until 150 jobs, it almost always provides an optimal solution very quickly. For larger n, the solution quality diminishes a little, and the solution times increase rapidly. In general, we observe that GPI-DS yields better results compared to our algorithm. However, this should not

[1] http://people.brunel.ac.uk/~mastjjb/jeb/info.html
[2] http://turbine.kuee.kyoto-u.ac.jp/~tanaka/SiPS/SiPS.html

Table 2. Results of iterated GPI-DS

	# of		% Gap		Total CPU Time*		Best Iter. #		Best CPU Time*	
n	Iter.	Opt.	Mean	Max	Mean	Max	Mean	Max	Mean	Max
40†	200	125.0	0	0	0.19	0.30	2.8	177	0.003	0.20
50†	450	125.0	0	0	0.74	1.08	5.2	252	0.009	0.27
100†	500	124.9	2.8E-06	0.005	4.85	7.66	14.3	395	0.179	5.16
150‡	500	124.5	3.1E-04	0.537	14.80	23.97	31.8	487	1.225	22.31
200‡	500	119.5	2.8E-02	12.698	33.12	54.53	58.6	498	5.133	49.52
250‡	500	115.2	2.3E-02	40.972	62.83	99.78	78.4	499	13.286	95.99
300‡	500	111.9	1.1E-03	0.157	106.49	177.30	88.9	498	25.611	167.52

*: In seconds. †: OR-Library. ‡: Tanaka *et. al.* (2009).

be held against our constructive algorithm for several reasons. First, observe that solving TR amounts to 98% of our solution time on average. Therefore, the focus of our ongoing research is on developing a scalable special-purpose algorithm for TR that exploits the structure in the cost matrix. We reckon that the solution time of TR can be slashed to a large extent which would render our algorithm significantly more competitive. (See Sourd and Kedad-Sidhoum (2003) for a similar effort for the single-machine earliness/tardiness problem.) Then, it would also be conceivable to feed our solution into GPI-DS - or into any other iterative heuristic for that matter - as the initial feasible solution to enhance performance. Second, our approach is very simple to implement compared to sophisticated local search heuristics and provides very good results except in a few cases. Third, the structural properties that we reveal for the preemptive relaxation may pave the way for further research. In particular, we note that the parentheses structure of a nested preemptive schedule corresponds to a time-based decomposition of the original instance. It may be possible to exploit this decomposition to limit the search effort in local search heuristics.

References

Bülbül K., P. Kaminsky and C. Yano, 2007, "Preemption in single machine earliness/tardiness scheduling", *Journal of Scheduling*, Vol. **10**(4-5), pp. 271-292.

Grosso A., F. D. Croce and R. Tadei, 2004, "An enhanced dynasearch neighborhood for the single-machine total weighted tardiness scheduling problem", *O. R. Letters*, Vol. **32**(1), pp. 68-72.

McNaughton R., 1959, Scheduling with deadlines and loss functions", *Management Science*, Vol. **6**(1), pp. 1-12.

Şen H., 2010, *A simple, fast, and effective heuristic for the single-machine total weighted tardiness problem*, Master's thesis, Sabancı University.

Sourd F. and S. Kedad-Sidhoum, 2003, "The one-machine problem with earliness and tardiness penalties", *Journal of Scheduling*, Vol. **6**(6), pp. 533-549.

Tanaka S., S. Fujikuma and M. Araki, 2009, "An exact algorithm for single-machine scheduling without machine idle time", *Journal of Scheduling*, Vol. **12**(6), pp. 575-593.