Parallel Block-Based Simulated Annealing for the Single Machine Total Weighted Tardiness Scheduling Problem

Wojciech Bożejko[1(✉)], Jarosław Pempera[1], Mariusz Uchroński[1], and Mieczysław Wodecki[2]

[1] Department of Control Systems and Mechatronics, Wrocław University of Science and Technology, Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland
{wojciech.bozejko,jaroslaw.pempera,mariusz.uchronski}@pwr.edu.pl

[2] Department of Telecommunications and Teleinformatics, Wrocław University of Science and Technology, Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland
mieczyslaw.wodecki@pwr.edu.pl

Abstract. This paper presents an algorithm based on the simulated annealing technique for the problem of scheduling tasks on one machine with the criterion of minimizing the sum of costs of tasks performed overdue. We propose to use, for the first time in simulated annealing, elimination criteria, the so-called blocks, allowing to not considered non-perspective solutions from the neighborhood. The use of blocks with the parallelization of the algorithm with the use of the MPI library allowed to improve the algorithm's efficiency in the same number of steps. Additionally, we propose to extend the benchmark database with new large test instances for which the block properties are most fully presented.

1 Introduction

Single-machine tasks scheduling problems with cost goal functions have a very long history – over 50 years (the first work of Rinnoy Kan et al. [8] appeared in 1977). Despite the simplicity of the formulation they mostly belong to the class of *NP-hard* problems. They are important both from the point of view of theory and practice. Such problems often constitute a significant part of the more extensive production systems. Different variants of single-machine tasks scheduling are still intensively tested and the obtained results are the inspiration for much more complex research on multi-machine problems. In the description of the considered in this chapter problem we will use some definitions, notations and properties presented in Wodecki's paper [13].

We propose to use the so-called 'block eliminating properties'. For the flow shop scheduling problem such algorithm were presented by Grabowski and Wodecki [2]). Blocks allow its users for both: reduction of calculations time, as well as improving the value of designated solutions. For the considered in this work one-machine problem *TWT*, in the works of Wodecki [13] and [14] there were two types of blocks presented, the so-called early task blocks and tardy task blocks. Below, we briefly present the basic definitions and properties of the problem under consideration.

2 Problem Description

A single-machine scheduling problem with due-dates of tasks completion times (Total Weighted Tardiness Problem, abbreviated as *TWT*) will be defined as follows. Let $\mathcal{J} = \{1, 2, \ldots, n\}$ be a set of tasks, each of which should be performed without interruption on the machine executing at most one task at a given time. For the task $i \in \mathcal{J}$ $(i = 1, \ldots, n)$ we define: p_i – execution time, w_i– penalty for tardiness, and d_i – requested date for completion of execution (so-called due-date).

Let $\pi = \pi(1), \pi(2), \ldots, \pi(n)$ be any permutation of elements from $\mathcal{J}$, and Φ a set of all such permutations. Then, $C_{\pi(i)} = \sum_{j=1}^{i} p_{\pi(j)}$ is a *completion date* of task $\pi(i) \in \mathcal{J}$, executed as i-th in order. If $C_{\pi(i)} \leq d_{\pi(i)}$, then the task is called early, otherwise - tardy.

Tardiness of task $T_{\pi(i)} = \max\{0, C_{\pi(i)} - d_{\pi(i)}\}$, and the *cost* of its execution $f(\pi(i)) = w_{\pi(i)} T_{\pi(i)}$. By

$$\mathcal{F}(\pi) = \sum_{i=1}^{n} w_{\pi(i)} T_{\pi(i)}. \tag{1}$$

we denote the *cost of execution* of all tasks (*weigh* of permutation π).

The problem under consideration (which will be also called a deterministic problem) consists in determining the permutation of $\pi^* \in \Phi$ with the smallest weight i.e. such that

$$\mathcal{F}(\pi^*) = \min\{\mathcal{F}(\pi) : \ \pi \in \Phi\}. \tag{2}$$

In the literature, this problem is denoted by $1||\sum w_i T_i$ and belongs to the class of *NP-hard* problems (Lenstra et al. [5]). Optimal algorithms for solving it based on the dynamic programming method were presented by Sahni [10] (for integer data this algorithm has the complexity $O(n \min\{\sum_j p_j, \sum_j w_j, \max_j\{d_j\}\}))$ and, based on the branch and bound, Potts, Van Wassenhove [7], Villareal, Bulfin [12] and Wodecki [14]. Optimal (exact) algorithms enable effective determination of examples of optimal solutions whose number of tasks does not exceed 80 (parallel algorithm, Wodecki [14]). Therefore, in practice there are usually approximate algorithms (mainly metaheuristics) used (tabu search algorithm, Bożejko et al. [1]). A very effective iterated local-search method has been proposed by Kirkik and Oguz [4]. The key aspect of the method is its ability to explore an exponential-size neighborhood in polynomial time by a dynamic programming technique.

3 Block Elimination Properties

Let $\pi \in \Phi$ be n-element permutation - solution to the *TWT*. The sequence the following one after another elements from π

$$\alpha = (\pi(a), \pi(a+1), \ldots, \pi(b)),$$

where $1 \leq a \leq b \leq n$ is called *subpermutation* of permutation π.

Definition 1. *Subpermutation of tasks $\pi^{\mathcal{T}}$ in permutation $\pi \in \Phi$ is called a* $\mathcal{T}$-block*, if:*

a) *each task $j \in \pi^{\mathcal{T}}$ is early and $d_j \geq C_{last}$, where C_{last} is the date of completion of the last task from $\pi^{\mathcal{T}}$,*
b) *$\pi^{\mathcal{T}}$ is a maximum subpermuation meeting the constraint (a).*

It is easy to see that if $\pi^{\mathcal{T}}$ is $\mathcal{T}$-block, then there is fulfilled the inequality

$$\min\{d_j : j \in \pi^{\mathcal{T}}\} \geq C_{last}.$$

Definition 2. *Subpermutation of tasks $\pi^{\mathcal{D}}$ in permutation $\pi \in \Phi$ is called of* $\mathcal{D}$-block*, if:*

a′) *each task $j \in \pi^{\mathcal{D}}$ is tardy and $d_j < S_{first} + p_j$, where S_{first} is the date of starting the first task from $\pi^{\mathcal{D}}$,*
b′) *$\pi^{\mathcal{D}}$ is the maximum subpermutation satisfying the constraint (a′).*

In this case, in any permutation of elements from $\pi^{\mathcal{D}}$ each task (belonging to $\pi^{\mathcal{D}}$), even swapped to the first position, is tardy in permutation π.

In the paper [13] there was a theorem proved.

Theorem 1. *For any permutation $\pi \in \Phi$ there is a division of π into subpermutations such, that each of them is:*

i) $\mathcal{T}$-block or
ii) $\mathcal{D}$-block.

Algorithm partitioning n-element permutation into blocks was described by [13] and has computational complexity of $O(n)$.

Definition 3. *Let $\mathcal{B} = [B_1, B_2, \ldots, B_s]$ be partitioning of permutation π into blocks. If in D-blocks the tasks occur according to relation*

$$\frac{w_{\pi(i-1)}}{p_{\pi(i-1)}} \geq \frac{w_{\pi(i)}}{p_{\pi(i)}}, \quad i = a, a+1, \ldots, b, \tag{3}$$

then π will be called D*-optimal and will be in short denoted by* D*-opt.*

The following theorem applies to the specific properties of blocks in solutions to the TWT problem and provides the basis for elimination (through an intermediate review) of certain elements of solutions space. It is successfully used in the algorithms based on the local search method.

Theorem 2 (Wodecki [13]). *For each ordered permutation $\pi \in \Phi$ if a permutation β was obtained from π by any interchange of its elements and $\mathcal{F}(\beta) < \mathcal{F}(\pi)$ then in the permutation β at least one task a block (T or D-opt) of π was moved before the first or the last task of this block.*

In order to eliminate from the neighborhood some solutions, generated by *'insert'* type moves there were 'block eliminating properties' (Theorem 2) used.

Let B_k $(k = 1, 2, \ldots, s)$ be the k-th block in permutation $\pi \in \Phi$. For job $j \in B_k$ by $\mathcal{N}_k^{bg}(j)$ let us denote a set of permutations created by moving job j to the beginning of block B_k (before first job in block). Analogously, for job $j \in B_k$ by $\mathcal{N}_k^{ed}(j)$ let us denote a set of permutations created by moving job j to the end of the block B_k (after the last job in block). The neighborhood of the solution $\pi \in \Phi$:

$$\mathcal{N}(\pi) = \bigcup_{j=1}^{s} \bigcup_{j \in B_k} (\mathcal{N}_k^{bg}(j) \cup \mathcal{N}_k^{ed}(j)). \tag{4}$$

4 Parallel Simulated Annealing Metaheuristic

Simulated Annealing (SA) is a stochastic heuristic algorithm which explores the solution space using randomized search procedure. The method was first applied to combinatorial problems by Kirkpatrick et al. [3]. In each iteration the SA algorithm a random perturbation is mode to the current solution $\pi \in \Phi$, giving rise to the set $\mathcal{N}(\pi)$ of neighbors. The neighbor $\beta \in \mathcal{N}(\pi)$ is accepted as the next configuration with probability function $\Psi(\pi, \beta, T)$. The $\Psi(\pi, \beta, T)$ is known as *accepting function* and depends on control parameter T (*temperature*). Its value changes suitably chosen intervals. In practice the accepting function is chose in such way that solutions corresponding to large increases in cost have a small probability of being accepted, whereas solutions corresponding small increases in cost have a larger probability of being accepted. The main objective is to escape from local optima keeping the convergence of the whole searching process. Change of temperature follows according to the cooling scheme. Initial T_0 is determined experimentally. The most common acceptance function in SA algorithm is Boltzman function $\Psi(\pi, \beta, T) = exp([F(\beta) - F(\pi)]/T$ (where $\pi \in \Phi$ and $\beta \in \mathcal{N}(\pi)$), geometrical cooling scheme $T \leftarrow c \cdot T$ $(0 < c < 1)$ and insert moves.

The proposed parallel SA algorithm has been parallelized using the MPI library using an idea of multiple independent search processes of different starting points (MSSS due to Voß [9] classification) On the supercomputer cluster platform, there were parallel processes implemented with mechanism of diversifying of starting solutions, based on the Scatter Search algorithm's idea, with using `MPI_Scatter` procedure. A parallel reduction mechanism (`MPI_Reduce`) was used to collect the data. The idea of the parallelization is given in Fig. 1. Both parallel SA with and without block-based neighborhood were implemented.

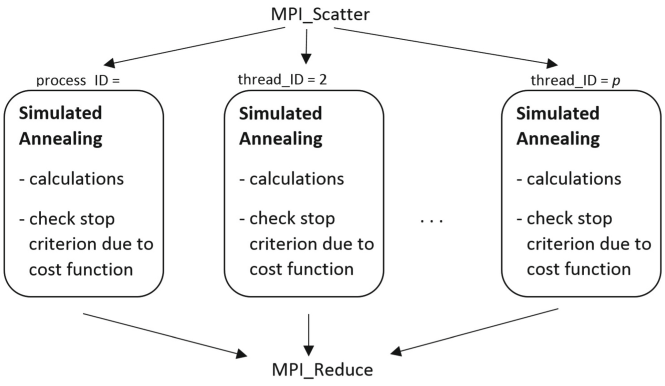

Fig. 1. Simulated annealing parallelization scheme

5 Computational Experiments

The parallel simulated annealing algorithm has been implemented in C++ and parallelized with MPI library. The calculations were performed on the BEM cluster located in the Wrocław Center for Networking and Supercomuting.

Calculations were made using two data sets of various sizes and degrees of difficulty. The first data set includes 375 examples of three different sizes ($n = 40, 50, 100$). This data set with the best solutions is placed on the OR-Library [6] website. The second data set was generated by authors in following way:

- p_i – random integer from range $[1, 100]$ with uniform distribution,
- w_i – random integer from range $[1, 10]$ with uniform distribution,
- d_i – random integer from range $[P(1 - T_F - R_{DD}/2), P(1 - T_F + R_{DD}/2)]$ with uniform distribution.

Where $P = \sum_{i=1}^{v} p_i$, $R_{DD} = 0.2, 0.4, 0.6, 0.8, 1.0$ (relative range of due dates) and $T_F = 0.2, 0.4, 0.6, 0.8, 1.0$ (average tardiness factor). For each of the 25 pairs of values of R_{DD} and T_F five instances were generated. For each value of $n = 200, 500, 1000$ 125 instances were generated – 325 in total. This data set was published on web page [11].

Computation was performed of two variants of simulated annealing algorithm (with blocks – SA_b and without blocks – SA) using test instances from previously mentioned data sets. Fore each solution the percentage relative error was determined with a formula $PRD = \frac{F_{ref} - F_{alg}}{F_{ref}} \cdot 100\%$, where:

- F_{ref} – value of the reference solution (from OR-Library for $wt40$, $wt50$, $wt100$ test instances and obtained with META (the best from SWPT, EDD, AU and COVERT rules, see [7]) algorithm for $wt200$, $wt500$, $wt1000$ test instances),
- F_{alg} – the value of the solution determined by tested algorithm (SA or SA_b).

Table 1, 2 and 3 presents results (PRD values) for parallel simulated annealing algorithm with and without blocks for test instances with sizes $n = \{40, 50, 100, 200, 500, 100\}$. Experiments have been conducted for number of processors $p = \{8, 16, 32, 64\}$. PRD values from Table 1 have been calculated using cost functions reference values from OR-Library. For test instances sizes $n = \{40, 50, 100\}$ there is no significant profit from using blocks – only for $n = 100$ results were little better for algorithm with blocks. Increasing number of processors results in smaller PRD values. PRD values from Table 2 and 3 have been calculated using cost functions values obtained with META algorithm (that is why they are negative). For bigger problem sizes $n = \{200, 500, 1000\}$ impact of using blocks on solution quality is more visible especially for $n = 200$ and 500. Similar as for previous data set increasing number of processors results in better solution quality.

Table 1. Results for parallel simulated annealing ($it = 100$)

Number of processors	$wt40$		$wt50$		$wt100$	
	SA	SA_b	SA	SA_b	SA	SA_b
8	0.21	0.13	0.29	0.29	0.11	0.16
16	0.08	0.40	0.12	0.17	0.08	0.04
32	0.05	0.10	0.12	0.01	0.07	0.07
64	0.04	0.03	0.04	0.10	0.06	0.02
Average	0.09	0.17	0.14	0.14	0.08	0.07

Table 2. Results for parallel simulated annealing ($it = 100$), META as a reference

Number of processors	$wt40$		$wt50$		$wt100$	
	SA	SA_b	SA	SA_b	SA	SA_b
8	-7.08	-7.15	-7.32	-7.33	-8.71	-8.67
16	-7.18	-6.91	-7.47	-7.42	-8.73	-8.76
32	-7.21	-7.16	-7.47	-7.54	-8.74	-8.74
64	-7.21	-7.22	-7.21	-7.22	-8.75	-8.77
Average	$-7,17$	$-7,11$	$-7,37$	$-7,38$	$-8,73$	$-8,74$

Table 3. Results for parallel simulated annealing ($it = 100$), META as a reference

Number of processors	$wt200$		$wt500$		$wt1000$	
	SA	SA_b	SA	SA_b	SA	SA_b
8	−10.04	−10.06	−10.12	−10.16	−10.39	−10.38
16	−10.06	−10.10	−10.12	−10.23	−10.09	−10.08
32	−10.10	−10.12	−10.22	−10.25	−10.10	−10.09
64	−10.10	−10.12	−10.24	−10.27	−10.10	−10.09
Average	−10, 08	−10, 10	−10, 18	−10, 23	−10, 17	−10, 16

6 Conclusions

Using blocks in parallel simulated annealing algorithm results in better solutions for bigger problem sizes in comparison with algorithm without blocks. Also increasing number of used parallel processors for computation results in increasing of obtained solutions (in sense of cost function value). We propose new large test examples for the problem under consideration, with sizes $n = 200, 500$ and 1000, because the ones used so far are already too small for modern algorithms.

Acknowledgments. The paper was partially supported by the National Science Centre of Poland, grant OPUS no. 2017/25/B/ST7/02181, grant no. 8211104160 K34W04D03 of Wrocław University of Science and Technology, and computational grant No. 96 of Wrocław Centre for Networking and Supercomputing.

References

1. Bożejko, W., Grabowski, J., Wodecki, M.: Block approach-Tabu search algorithm for single machine total weighted tardiness problem. Comput. Ind. Eng. **50**(1/2), 1–14 (2006)
2. Grabowski, J., Wodecki, M.: A very fast Tabu search algorithm for the permutation flow shop problem with makespan criterion. Comput. Oper. Res. **31**, 1891–1909 (2004)
3. Kirkpatrick, S., Gellat, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **26**, 53–67 (1983)
4. Kirlik, G., Oguz, C.: A variable neighborhood search for minimizing total weighted tardiness with sequence dependent setup times on single machine. Comput. Oper. Res. **39**, 1506–1520 (2012)
5. Lenstra, J.K., Rinnoy Kan, A.H.G., Brucker, P.: Complexity of machine scheduling problems. Ann. Discrete Math. **1**, 343–362 (1977)
6. OR Library: http://people.brunel.ac.uk/~mastjjb/jeb/info.html
7. Potts, C.N., Van Wassenhove, L.N.: Algorithms for scheduling a single machine to minimize the weighted number of late tasks. Manage. Sci. **34**(7), 843–858 (1988)
8. Rinnoy Kan, A.H.G., Lageweg, B.J., Lenstra, J.K.: Minimizing total costs in one-machine scheduling. Oper. Res. **25**, 908–927 (1975)

9. Voß, S.: Tabu search: applications and prospects. In: Du, Z., Pardalos, P.M. (eds.) Network Optimization Problems: Algorithms, Applications and Complexity, pp. 333–353. World Scientific Publishing Co, Singapore (1993)
10. Sahni, S.K.: Algorithms for scheduling independent tasks. J. Assoc. Comput. Match **23**, 116–127 (1976)
11. Uchroński, M.: Test instances for a single-machine total weighted tardiness scheduling problem. https://zasobynauki.pl/zasoby/51561
12. Villareal, F.J., Bulfin, R.L.: Scheduling a single machine to minimize the weighted number of tardy tasks. IEE Trans. **15**, 337–343 (1983)
13. Wodecki, M.: A block approach to earliness-tardiness scheduling problems. Adv. Manuf. Technol. **40**, 797–807 (2009). https://doi.org/10.1007/s00170-008-1395-7
14. Wodecki, M., Bound, A.B.: Parallel algorithm for single-machine total weighted tardiness problem. Adv. Manuf. Technol. **37**, 996–1004 (2008)