Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computers & Industrial Engineering xx (2006) 1–14
computers & industrial engineering
www.elsevier.com/locate/dsw

# Block approach—tabu search algorithm for single machine total weighted tardiness problem

Wojciech Bożejko [a], Józef Grabowski [a,*], Mieczysław Wodecki [b]

[a] *Institute of Engineering Cybernetics, Wrocław University of Technology, Janiszewskiego 11-17, 50-372 Wrocław, Poland*
[b] *Institute of Computer Science, University of Wrocław, Przesmyckiego 20, 51-151 Wrocław, Poland*

Received 25 May 2005; received in revised form 3 November 2005; accepted 9 December 2005

**Abstract**

Problem of scheduling on a single machine to minimize total weighted tardiness of jobs can be described as follows: there are $n$ jobs to be processed, each job has an integer processing time, a weight and a due date. The objective is to minimize the total weighted tardiness of jobs. The problem belongs to the class of NP-hard problems. Some new properties of the problem associated with the blocks have been presented and discussed. These properties allow us to propose a new fast local search procedure based on a tabu search approach with a specific neighborhood which employs blocks of jobs and a compound moves technique. A compound move consists in performing several moves simultaneously in a single iteration of algorithm and allows us to accelerate the convergence to good solutions. In the algorithm, we use an idea which decreases the complexity for the search of neighborhood from $O(n^3)$ to $O(n^2)$. Additionally, the neighborhood is reduced by using some elimination criteria. The method presented in this paper is deterministic one and has not any random element, as distinct from other effective but non-deterministic methods proposed for this problem, such as tabu search of Crauwels, H. A. J., Potts, C. N., & Van Wassenhove, L. N. (1998). Local search heuristics for the single machine total weighted tardiness Scheduling Problem. *INFORMS Journal on Computing*, *10*(3), 341–350, iterated dynasearch of Congram, R. K., Potts C. N., & Van de Velde, S. L. (2002). An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, *14*(1), 52–67 and enhanced dynasearch of Grosso, A., Della Croce, F., & Tadei, R. (2004). An enhanced dynasearch neighborhood for single-machine total weighted tardiness scheduling problem. *Operations Research Letters*, *32*, 68–72. Computational experiments on the benchmark instances from OR-Library (http://people.brunel.ac.uk/mastjjb/jeb/info.html) are presented and compared with the results yielded by the best algorithms discussed in the literature. These results show that the algorithm proposed allows us to obtain the best known results for the benchmarks in a short time. The presented properties and ideas can be applied in any local search procedures.


*Keywords:* Sequencing; Weighted tardiness problem; Heuristics; Tabu search

## 1. Introduction

In the single machine total weighted tardiness problem (TWTP), denoted as $1||\Sigma w_i T_i$, a *set of jobs* $N=\{1,2,\ldots,n\}$ has to be processed without interruption on a single machine that can handle only one job at a time.

* Corresponding author.
*E-mail addresses:* wbo@ict.pwr.wroc.pl (W. Bożejko), grabow@ict.pwr.wroc.pl (J. Grabowski), mwd@ii.uni.wroc.pl (M. Wodecki).

0360-8352/$ - see front matter © 2006 Published by Elsevier Ltd.
doi:10.1016/j.cie.2005.12.001

Each job $i \in N$ has integer *processing time* $p_i$, *due date* $d_i$, and *positive weight* $w_i$. For a given sequence of jobs, the (earliest) *completion time* $C_i$, *tardiness* $T_i = \max\{0, C_i - d_i\}$ and *cost* $f_i(C_i) = w_i T_i$ of job $i \in N$ can be computed. The objective is to find a job sequence which minimizes the sum of the costs given by formula $\sum_{i=1}^{n} f_i(C_i) = \sum_{i=1}^{n} w_i T_i$.

The total weighted tardiness problem is NP-hard in strong sense by a result of (Lawler, 1977; Lenstra, Rinnooy Kan, & Brucker, 1977). A large number of studies have been devoted on the problem. Emmons (1969) derives several dominance rules that restrict search process for an optimal solution. These rules are used in many algorithms. Enumerative algorithms that use dynamic programming and branch and bound approaches for the problem are described by Fisher (1976), Lawler (1979), Potts and Van Wassenhove (1985) and Rinnooy Kan, Lageweg, and Lenstra (1975). These and other algorithms are discussed and tested in review paper by Abdul-Razaq, Potts, and Van Wassenhove (1990). The algorithms are a significant improvement over exhaustive search, but they remain laborious and are applicable only to relatively small problems (with the number of jobs not exceed to 50). The enumerative algorithms require considerable computer resources both in terms of computation times and core storage. Therefore, many algorithms have been proposed to find near optimal schedules in reasonable time. These algorithms can be broadly classified into construction and interchange methods.

The construction methods use dispatching rules to build a solution by fixing a job in a position at each step. Several constructive heuristics are described by Cheng, Ng, Yuan, and Liu (2005), Fisher (1976) and Morton, Rachamadugu, and Vepsalainen (1984), and in review paper by Potts and Van Wassenhove (1991). They are very fast, but the quality of the solutions is not good.

Interchange methods start from an initial solution and repeatedly try to improve the current solution by local changes. The interchanges are continued until a solution that cannot be improved is obtained which is a local minimum. To increase the performance of local search algorithms, there are used metaheuristics like tabu search (Crauwels, Potts, & Van Wassenhove, 1998), Simulated Annealing (Matsuo, Otto, & Sullivan, 1987; Potts & Van Wassenhove, 1991), Genetic Algorithms (Crauwels et al., 1998), Ant Colony Optimization (Den Basten, Stützle, & Dorigo, 2000, 2001). A very effective local search method has been proposed by Congram, Potts, and Van de Velde (2002), and next improved by Grosso, Della Croce, and Tadei (2004). Recently, the Congram local search approach has been applied by Angel and Bampis (2005), to the TWTP where the processing time of each job is dependent on its starting time. The key aspect of the method is its ability to explore an exponential-size neighborhood in polynomial time, using a dynamic programming technique.

In this paper, we present new properties of the problem associated with the so-called blocks of jobs, and a fast algorithm based on a tabu search approach with specific neighborhood and compound moves technique. The properties allow us to skip some non-perspective solutions during the search of the solutions space. Applied to local search algorithms, they result in a significant reduction of the neighborhood size. While, a proposed compound move consists of several single moves that are performed simultaneously in each iteration of the algorithm and have a feature that their total contribution to changing the objective function is simply the sum of their separate contributions, without interacting effects that modify this total when the moves are performed together. It allows the algorithm to visit the more promising areas, and to achieve very good solutions in a much shorter time, accelerating the convergence of the algorithm. Also, in our algorithm, we propose a tabu list with dynamic length which is changed cyclically, as the current iteration number increases. This kind of tabu list can be viewed as a specific disturbance that gives an additional assistance to avoid getting trapped at a local optimum of algorithm.

Besides, the computational effort of our algorithm is decreased by applying the idea presented in the paper by Grabowski and Pempera (1998), which improves the complexity for the search of neighborhood from $O(n^3)$ to $O(n^2)$, Additionally, the effort is decreased by reducing of the neighborhood size, using some elimination criteria.

The paper is organized as follows. Section 2 presents the notations, basic definitions and new properties of the problem. In Section 3, the moves and neighborhood structure, search process, dynamic tabu list, structure of compound moves, and heuristic algorithm are described. Computational results are shown in Section 4 and compared with those taken from the literature. Section 5 gives our conclusions and remarks.

## 2. Problem description and preliminaries

Each schedule of jobs can be represented by permutation $\pi=(\pi(1), \pi(2),\ldots,\pi(n))$ on set $N$. Let $\Pi$ denotes the set of all such permutations. The total cost of $\pi\in\Pi$ is $F(\pi)=\sum_{i=1}^{n} w_{\pi(i)}T_{\pi(i)}$ with $T_{\pi(i)}=\max\{0,C_{\pi(i)}-d_{\pi(i)}\}$, where $C_{\pi(i)}=\sum_{j=1}^{i} p_{\pi(j)}$ is a completion time of the job $\pi(i)$. Job $\pi(i)$ is considered as *early* one, if it is completed before its due date (i.e. $C_{\pi(i)}\leq d_{\pi(i)}$), or *tardy* if the job is completed after its due date (i.e. $C_{\pi(i)}>d_{\pi(i)}$).

The problem is to find a permutation $\pi^*\in\Pi$ which minimizes the function $F$ on the set $\Pi$, i.e.

$$F(\pi^*) = min\{F(\pi):\pi\in\Pi\}.$$

Each permutation $\pi\in\Pi$ is decomposed into $m$ ($m\leq n$) subsequences $B_1,B_2,\ldots,B_m$, called *blocks* in $\pi$, each of them contains the jobs having in common specific properties, where

1. $B_k=(\pi(f_k),\pi(f_k+1),\ldots,\pi(l_k-1)),\pi(l_k)), \quad l_{k-1}+1=f_k\leq l_k$, $k=1,2,\ldots,m$, $l_0=0$, $l_m=n$.
2. All the jobs $j\in B_k$ satisfy the following condition
   either
   $$d_j\geq C_{\pi(l_k)}, \tag{C1}$$
   or
   $$d_j< S_{\pi(f_k)}+p_j, \tag{C2}$$
   where $S_{\pi(f_k)}$ is a *starting time* of the job $\pi(f_k)$, i.e. $S_{\pi(f_k)}=C_{\pi(f_k)}-p_{\pi(f_k)}$. Clearly, each job $j\in B_k$ satisfying Condition C1 (or C2) is a early (or tardy) one in $\pi$.
3. $B_k$ is maximal subsequence of $\pi$ in which all the jobs satisfy either Condition C1 or Condition C2.

Jobs $\pi(f_k)$ and $\pi(l_k)$ in $B_k$ are the *first* and *last* ones, respectively. Note that a block can contain only one job, i.e. $|B_k|=1$, and then $f_k=l_k$. Note that all the blocks are connected 'in series' in permutation $\pi$. By definition, there exist two type of blocks implied by either C1 or C2. To distinguish them, we will use the E-block and T-block notions (or alternatively $B_k^{\mathrm{E}}$ and $B_k^{\mathrm{T}}$), respectively (see Fig. 1).

**Example 1**. Let us consider $n=10$ jobs' instance that is specified in Table 1.

Let $\pi=(1,2,3,4,5,6,7,8,9,10)$. Permutation $\pi$ contains five blocks (i.e. $m=5$), $f_1=1$, $l_1=4$, $f_2=5$, $l_2=6$, $f_3=l_3=7$, $f_4=l_4=8$, $f_5=9$, $l_5=10$ and $B_1=(1,2,3,4)$, $B_2=(5,6)$, $B_3=(7)$, $B_4=(8)$, $B_5=(9,10)$. There are three E-blocks: $B_1,B_3,B_4$, and two T-blocks: $B_2$ and $B_5$. These blocks are shown in Fig. 1.

Let

$$F_k(\pi) = \sum_{j\in B_k} w_j T_j$$

be a *partial value* of the objective associated with the block $B_k$ in $\pi$. It is clear that by the definition of blocks in $\pi$, we have:

$$F(\pi) = \sum_{k=1}^{m} F_k(\pi).$$

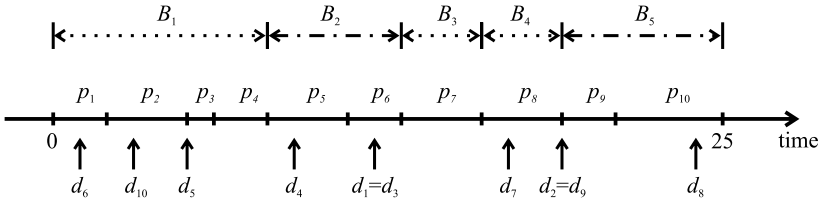


Fig. 1. Blocks in permutation $\pi$.

Table 1
Data for the instance

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $p_i$ | 2 | 3 | 1 | 2 | 3 | 2 | 3 | 3 | 2 | 4 |
| $d_i$ | 12 | 19 | 12 | 9 | 5 | 1 | 17 | 24 | 19 | 3 |
| $w_i$ | 3 | 1 | 2 | 5 | 3 | 3 | 4 | 2 | 4 | 5 |

It is evident that by Condition C1, for any permutation of jobs within an E-block $B_k^{\mathrm{E}}$ of $\pi$ (i.e. in the positions $f_k$, $f_k+1,\ldots,l_k-1$, $l_k$ of $B_k^{\mathrm{E}}$), we have:

$$F_k(\pi) = \sum_{j \in B_k^{\mathrm{E}}} w_j T_j = 0. \tag{1}$$

With respect to T-blocks $B_k^{\mathrm{T}}$ in $\pi$, it should be noticed that by Condition C2, for any permutation of jobs within $B_k^{\mathrm{T}}$ (i.e. in the positions $f_k$, $f_k+1,\ldots,l_k-1$, $l_k$ of $B_k^{\mathrm{T}}$), all the jobs are tardy. Therefore, an optimal sequence of the jobs within $B_k^{\mathrm{T}}$ of $\pi$ can obtained, using well-known weighted shortest processing time (WSPT) rule proposed by Smith (1956). The WSPT rule creates an optimal sequence of the jobs in the non-increasing order of the ratios $w_j/p_j$.

Let $\vec{B}_k^T$ denotes a T-block of the jobs from $B_k^T$ of $\pi$ ordered by the WSPT rule, then we have

$$F_k(\pi) = \sum_{j \in B_k^T} w_j T_j \geq \sum_{j \in \vec{B}_k^T} w_j T_j = \vec{F}_k(\pi), \tag{2}$$

where the jobs from $B_k^T$ are ordered in any permutation.

Fundamental Block Properties of the TWTP are derived from the following Theorem.

**Theorem 1**. *Let $\pi \in \Pi$ be any permutation with blocks $B_1,B_2,\ldots,B_m$, and let the jobs of each T-block of $\pi$ are ordered according to the WSPT rule. If the permutation $\beta$ has been obtained from $\pi$ by an interchange of jobs that $F(\beta) < F(\pi)$, then in $\beta$*

(i) *at least one job from $B_k$ precedes at least one job from blocks $B_1,B_2,\ldots,B_{k-1}$, for some $k \in \{2,3,\ldots,m\}$, or*
(ii) *at least one job from $B_k$ succeeds at least one job from blocks $B_{k+1},B_{k+2},\ldots,B_m$, for some $k \in \{1,2,\ldots,m-1\}$.*

The proof of the theorem is given in Appendix A.

Note that Theorem 1 provides the necessary condition to obtain a permutation $\beta$ from $\pi$ such that $F(\beta) < F(\pi)$.

## 3. Tabu search algorithm with compound moves (TS+M)

Currently, tabu search approach (TS) (see Glover, 1989, 1990; Glover & Laguna, 1998) is one of the most effective methods using local search techniques to find near-optimal solutions of many combinatorial intractable optimization problems, such as the vast majority of scheduling problems. This technique aims to guide the search by exploring the solution space of a problem beyond local optimality. The main idea of this method involves starting from an initial basic job permutation and searching through its neighborhood, a set of permutations generated by the moves, for a permutation with the lowest value of objective function. The search then is repeated starting from the best permutation, as a new basic permutation, and the process is continued. One of the main ideas of TS is the use of a tabu list to avoid cycling, overcoming local optimum, or continuing the search in a too narrow region and to guide the search process to the solutions regions which have not been examined. The tabu list records the performed moves that, for a chosen span of time, have tabu status and cannot be applied currently (they are forbidden); that is they determine forbidden permutations in the currently analyzed neighborhood. The list content is refreshed each time a new basic permutation is found; the oldest element is removed and the new one is added. In our algorithms, a tabu list with dynamic length is applied that assists us additionally to avoid being trapped at a local optimum. The algorithm TS terminates when a given number of iterations (*Maxiter*) has been reached without improvement of the best current

value of objective function, the algorithm has performed a given number of iterations, time has run out, the neighborhood is empty, a permutation with a satisfying value of objective function has been found, etc.

### 3.1. Moves and neighborhoods

One of the main components of a local search algorithm is the definition of the move set that creates a neighborhood. A move changes the location of some jobs in a given permutation. In the literature, we can meet many types of a move based on interchanges of jobs on a machine (see Grabowski & Pempera, 2001). The intuition following from Theorem 1 suggests that the 'insert' or 'swap' type moves should be the most proper ones for the problem considered.

Let $v=(x,y)$ be a pair of position in a permutation $\pi$, $x,y\in\{1,2,\ldots,n\}$, $x\neq y$. The pair $v=(x,y)$ defines a move in $\pi$ as follows:

(i) Insert move (*I-move*), in which the job $\pi(x)$ is removed from its original position $x$, and next insert it in a position $y$ in $\pi$. Thus, move $v=(x,y)$ generates a permutation $\pi_v$ from $\pi$ in the following way:

$$\pi_v = (\pi(1),\ldots,\pi(x-1),\pi(x+1),\ldots,\pi(y),\pi(x),\pi(y+1),\ldots,\pi(n)), \quad \text{if} \quad x<y,$$

$$\pi_v = (\pi(1),\ldots,\pi(y-1),\pi(x),\pi(y),\ldots,\pi(x-1),\pi(x+1),\ldots,\pi(n)), \quad \text{if} \quad x>y.$$

(ii) Swap move (*S-move*), in which the jobs $\pi(x)$ and $\pi(y)$, $x\neq y$, are interchanged in some positions $x$ and $y$ in $\pi$. Now, the move $v=(x,y)$ generates $\pi_v$ from $\pi$ in the following manner:

$$\pi_v = (\pi(1),\ldots,\pi(x-1),\pi(y),\pi(x+1),\ldots,\pi(y-1),\pi(x),\pi(y+1),\ldots,\pi(n)), \quad x\neq y.$$

Since, by definition of S-moves it follows that $v=(x,y)=(y,x)$, then it is sufficient to consider the S-moves $v=(x,y)$ with $x<y$.

In our algorithm, both I- and S-moves will be used.

The neighborhood of $\pi$ consists of permutations $\pi_v$ obtained by moves from a given set $Z$, and denoted as $N(Z,\pi)=\{\pi_v|v\in Z\}$. The proper selection of $Z$ is very important to construct an effective algorithm.

The largest neighborhood generated by both I- and S-moves set requires a great computational effort for the search, namely $O(n^3)$, assuming that all the neighbors are evaluated explicitly by the objective function. In our algorithm, this complexity has been reduced to $O(n^2)$, using the idea proposed by Grabowski and Pempera (1998), whereby the value $F(\pi_{v'})$ for a move $v=(x,y)$ is obtained by employing the component values of $F(\pi_v)$ for the move $v'=(x,y-1)$ (or $v'=(x,y+1)$), calculated immediately prior to the $F(\pi_v)$.

In sequel, we consider some non-perspective moves that result in a significant reduction of the neighborhood size and computational effort.

For any block $B_k$ in $\pi$, let us consider the set of moves $W_k(\pi)$, which can be performed within this block, i.e. in the positions $f_k, f_k+1,\ldots,l_k-1, l_k$, $k=1,2,\ldots,m$. Precisely, set $W_k(\pi)$ is defined by formula:

$$W_k(\pi) = \{(x,y)|x,y\in\{f_k,f_k+1,\ldots,l_k-1,l_k\},x\neq y\}, \quad k=1,2,\ldots,m.$$

All these moves create the set $W(\pi)=\cup_{k=1}^{m} W_k(\pi)$.

Immediately from Theorem 1 we obtain Corollary 1 which provides the basis for elimination.

**Corollary 1**. *Let $\pi\in\Pi$ be any permutation in which the jobs of each T-block are ordered according to the WSPT rule. If the permutation has been generated from $\pi$ by I-move (or S-move) $v\in W(\pi)$, then $F(\pi_v)\geq F(\pi)$.*

Corollary 1 states that for the permutation $\pi$, the moves from set $W(\pi)$ are not interesting, taking into account the possibility of an immediate improvement of $F(\pi)$ after making a move.

As a consequence of the above, in our algorithm, we will employ permutations (as basic ones) in which the jobs of each T-block are ordered according to the WSPT rule, and let $\pi\in\Pi$ denote any such permutation.

Next, we will give a detailed description of the moves and neighborhood structure considered in this paper. Moves are associated with the blocks in $\pi$. Let us take a block $B_k=(\pi(f_k),\pi(f_k+1),\ldots,\pi(l_k))$, $k=1,2,\ldots,m$. For I-moves, each

block $B_k$ contains the jobs that are candidates for being moved to the right and left. More precisely, we move job $\pi(x)$, $\pi(x)\in B_k$, $k=1,2,\ldots,m-1$, to the right in to the positions $f_{k+1}, f_{k+1}, +1,\ldots,n$, and this sets of I-moves takes the form:

$$IR_{kx} = \{(x,y)|f_{k+1} \le y \le n\}, \quad x = f_k, f_k + 1,\ldots,l_k.$$

By symmetry, job $\pi(x)$, $\pi(x)\in B_k$, $k=2,3,\ldots,m$, is moved to the left in to the positions $1,2,\ldots,l_{k-1}$, and this sets of I-moves takes the form:

$$IL_{kx} = \{(x,y)|1 \le y \le l_{k-1}\}, \quad x = f_k, f_k + 1,\ldots,l_k.$$

Note that for any move $v\in IR_{kx}$, or $v\in IL_{kx}$, Corollary 1 cannot be applied, i.e. $v\notin W_k$. For illustration, these I-moves performed to the right and left are shown in Fig. 2.

According to the description given, in our tabu search algorithm, we will use the set of the I-moves:

$$\mathrm{IM} = \bigcup_{k=1}^{m-1} \bigcup_{x=f_k}^{l_k} [IR_{kx} \cup IL_{k+1,x}].$$

The similar considerations can be provided for the S-moves, we then obtain the set of S-moves

$$S_{kx} = \{(x,y)|f_{k+1} \le y \le n\}, \quad x = f_k, f_k + 1, \ldots, l_k,$$

i.e. job $\pi(x)$, $\pi(x)\in B_k$, $k=1,2,\ldots,m-1$, is moved to the right in to the position $y$, $x<f_{k+1}\le y\le n$, and job $\pi(y)$ is moved to the left in to the position $x$ (jobs $\pi(x)$ and $\pi(y)$ are interchanged).

As a consequence, in our algorithm, we will use the set of all S-moves:

$$\mathrm{SM} = \bigcup_{k=1}^{m-1} \bigcup_{x=f_k}^{l_k} S_{kx}.$$

For illustration, these S-moves are shown in Fig. 2.

Finally, in our TS+M, we propose the following set of moves

$$\mathrm{M} = \mathrm{IM} \cup \mathrm{SM},$$

which creates neighborhood $N(M,\pi)$.

Further decrease of the computational effort can be gained by reducing the neighborhood size, using some elimination criteria.

Suppose that we have found that there exists an optimal permutation whereby, for job $j$, the set of jobs $b(j)$ precedes $j$, and the set of jobs $a(j)$ follows $j$. Any established relation '*i precedes j*' implies that $i\in b(j)$ and $j\in a(j)$. Let $\bar{a}(j)$, $j=1,2,\ldots,n$, be the complement set of $a(j)$, i.e. $\bar{a}(j) = N - a(j)$. The sets $a(j)$ and $b(j)$ can be created by application of Theorem 2.

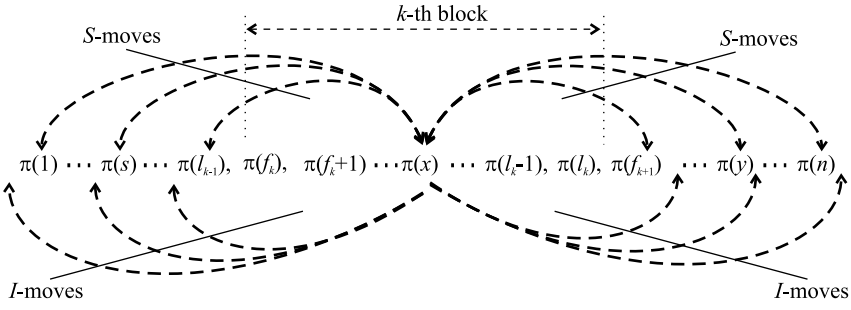


Fig. 2. The moves of job $\pi(x)$.

**Theorem 2**. (Emmons, 1969; Fisher, 1976; Rinnooy Kan et al., 1975; Shwimer, 1972). *There exists an optimal permutation whereby job i precedes* job *j if at least one of the conditions* (1)–(3) *is satisfied*

1. $w_i \geq w_j$, $p_i \leq p_j$, $d_i \leq \max\left(d_j, \sum_{l\in b(j)} p_l + p_j\right)$,
2. $w_i \geq w_j$, $d_i \leq d_j$, $d_j \geq \sum_{l\in \bar{a}(j)} p_l - p_j$,
3. $d_j \geq \sum_{l\in \bar{a}(i)} p_l$.

In our algorithm TS + M, we employed the algorithm proposed by Fisher (1976), in which the sets $a(j)$ and $b(j)$, initially empty, are augmented by successively application of Theorem 2. This algorithm is performed at beginning of TS + M. There is a problem concerning the implementation of a number of elimination criteria because it is possible to create the *precedence cycles*. In our algorithm, we avoid the creation of precedence cycles by immediately constructing the *transitive closure* of each precedence relation '*i precedes j*' that we find:

$a(l) := a(l) \cup \{j\} \cup a(j)$ for each $l \in \{i\} \cup b(i)$, and

$b(l) := b(l) \cup \{i\} \cup b(i)$ for each $l \in \{j\} \cup a(j)$.

According to the definitions of $a(j)$ and $b(j)$, it is easy to verify that we can eliminate from the set of I-moves (i.e. from IM), the following set of moves

$$\mathrm{IE}(\pi) = \mathrm{IER}(\pi) \cup \mathrm{IEL}(\pi)$$

where

$$\mathrm{IER}(\pi) = \{(x,y) \in \mathrm{IM} | \pi(x) \in b(j), j \in \{\pi(x+1), \ldots, \pi(y)\}, x < y\},$$

$$\mathrm{IEL}(\pi) = \{(x,y) \in \mathrm{IM} | \pi(x) \in a(j), j \in \{\pi(y), \ldots, \pi(x-1)\}, x > y\}.$$

While, from the set of S-moves (i.e. from SM) we can eliminate

$$\mathrm{SE}(\pi) = \mathrm{SE}b(\pi) \cup \mathrm{SE}a(\pi)$$

where

$$\mathrm{SE}b(\pi) = \{(x,y) \in \mathrm{SM} | \pi(x) \in b(j), j \in \{\pi(x+1), \ldots, \pi(y)\}, x < y\},$$

$$\mathrm{SE}a(\pi) = \{(x,y) \in \mathrm{SM} | \pi(y) \in a(j), j \in \{\pi(x), \ldots, \pi(y-1)\}, x < y\}.$$

As a consequence of above considerations, in our algorithm TS, we will employ the neighborhood $N(\mathrm{ME},\pi)$, where $\mathrm{ME} = M - E(\pi)$ and $E(\pi) = \mathrm{IE}(\pi) \cup \mathrm{SE}(\pi)$.

In order to accelerate the convergence of the algorithms to good solutions, we propose to use the *compound moves*, introduced originally by Glover and Laguna (1998). A compound moves consist of *several* moves that are performed *simultaneously* in a single iteration of algorithm. The performances of the compound moves allow us to generate permutations that differ in various significant ways from those obtained by performing a single move and to carry the search process to hitherto non-visited regions of the solution space. Furthermore, in our algorithm, the compound moves have the purpose of guiding the search to visit the more promising areas, where 'good solutions' can be found. To this order, we stipulate a condition that assure each move from a compound move can be performed simultaneously and that their total contribution to changing the objective function is simply the sum of their separate contributions, without interacting effects that modify this total when the moves are performed together. This kind of compound move was employed on that very fast tabu search algorithm proposed by Grabowski and Pempera (2005), where it was successfully applied to the no-wait flow shop problem.

In local search algorithms, the use of compound moves can be viewed as a way to apply a mixture of intensification and diversification strategies in the search process.

In the following, we present a method that will be used in our heuristic algorithms to provide the compound moves.

Let

$$P = \{v \in \mathrm{ME} | F(\pi_v) < F(\pi)\}$$

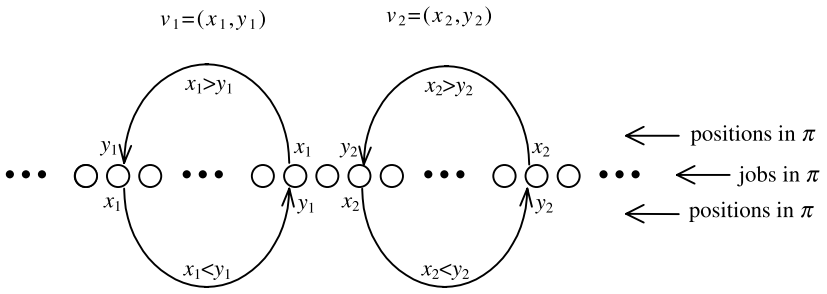


Fig. 3. Moves $v_1$ and $v_2$ satisfying Condition (C3).

be the set of the *profitable moves* performing of which generates permutation $\pi_v$ 'better' than $\pi$. Note that the move $v \in P$ can be either I-move or S-move.

Two moves $v_1=(x_1,y_1)\in P$ and $v_2=(x_2,y_2)\in P$ are called the *independent* ones with respect to $\pi$, if each of the positions $x_1$, $y_1$ is separated from each of the positions $x_2$, $y_2$. More precisely, $v_1$ and $v_2$ are independent if the following Condition is satisfied:

$$\begin{cases} \max(x_1, y_1) < \min(x_2, y_2) \\ \text{or} \\ \max(x_2, y_2) < \min(x_1, y_1). \end{cases} \tag{C3}$$

Condition (C3) implies that the moves $v_1$ and $v_2$ operate in series in the job sequence $\pi$, i.e. $v_1$ precedes $v_2$ (or $v_1$ succeeds $v_2$) in the sequence, and they are separated, see Fig. 3.

Let IP be a subset of $P$ containing the independent moves of $P$. This means that for each pair of moves $v_1\in$IP and $v_2\in$IP, $v_1\neq v_2$, is satisfied Condition (C3). From the definition of IP it results that each move $v\in$IP produces permutation $\pi_v$ 'better' than $\pi$. Therefore, as a *compound move*, we took the set IP. The use of this compound move consists in performing all the moves from IP simultaneously, generating a permutation, denoted as $\pi_{\hat{v}}$, where $\hat{v}=$IP. To simplify, in the further considerations, compound move IP will be denoted alternatively by $\hat{v}$. Note that the permutation $\pi_{\hat{v}}$ does not belong to $N(ME,\pi)$, unless $|\hat{v}|=1$. The intuition following from the definition of $\hat{v}$ suggests that $\pi_{\hat{v}}$ should be significantly better than $\pi_v$ generated by the best (single) move $v\in\hat{v}$, since the total improvement of $F(\pi_{\hat{v}})$ is obtained by adding all the improvements produced by the individual moves from $\hat{v}$. It is a specific property of the considered problem that holds for the independent and profitable moves and that has not been applied to our problem. It allows the algorithm to achieve very good solutions in a much shorter time. Therefore, the performance of compound move $\hat{v}$ guides the search to visit new more promising regions of the solution space where good solutions can found. Note that if $\hat{v}=\varnothing$, then the compound move cannot be used. Next, we present a procedure that creates a compound move $\hat{v}$, used in our heuristic algorithm.

*Step 1*: For given $\pi$, create the set of profitable moves $P$ and set $\hat{v}:=\varnothing$.
*Step 2*: Find the best move $v^*$, i.e. $F(\pi_{v^*})=\min_{v\in P}F(\pi_v)$, and set $P:=P-\{v^*\}$ and $\hat{v}:=\hat{v}\cup\{v^*\}$.
*Step 3*: Find the best move $v^*$, i.e. $F(\pi_{v^*})=min_{v\in P}F(\pi_v)$, and for each move $v\in\hat{v}$ check Condition (C3) for the moves $v^*$ and $v$. If there is a move $v\in\hat{v}$ such that for $v^*$ and $v$ the condition is not satisfied, then set $P:=P-\{v^*\}$. Otherwise set $P:=P-\{v^*\}$ and $\hat{v}:=\hat{v}\cup\{v^*\}$.
*Step 4*: Repeat Step 3 until $P\neq\varnothing$.

It should be noticed that this procedure is based on a greedy algorithm, and therefore it does not necessarily create the best compound move.

Generally, in our TS+M, for the given initial permutation, we identify the blocks (if there is more than one partition of the permutation into blocks, any one them can be used), and order the jobs of each T-block according to the WSPT rule. Then, for the resulting (basic) permutation $\pi$, we calculate $F(\pi)$, create the set of moves $ME$, compound move $\hat{v}$, and the permutation $\pi_{\hat{v}}$ (see Section 3.3). Next, the search process of TS+M is repeated for the new initial permutation $\pi_{\hat{v}}$ until a given number of iterations is reached. According to the philosophy of TS, the

compound move cannot contain the single moves with a status tabu; these moves are not allowed (see following sections for details).

### 3.2. Tabu list and tabu status of move

In our algorithm, we use the cyclic tabu list defined as a finite list (set) $T$ with length *LengthT* containing ordered triplets. The list $T$ is a realization of the short-term search memory. If a move $v=(x,y)$ is performed on permutation $\pi$, then, a triplet $(\pi(x),y,F(\pi_v))$ is added to $T$. If the compound move $\hat{v}$ is performed, then the triplet corresponding to each move from $\hat{v}$ is added to the tabu list. Each time before adding a new element to $T$, we must remove the oldest one. With respect to a permutation $\pi$, a move $v=(x,y)\in ME$ is forbidden, i.e. it has *tabu* status, if there is a triplet $(r,s,\phi)$ in $T$ such that $\pi(x)=r$, $y=s$, and $F(\pi_v)\geq\phi$.

As mentioned above, our algorithm uses a tabu list with dynamic length. This length is changed, as the current iteration number *iter* of TS+M increases, using a 'pick' that can be treated as a specific disturbance (diversification). This kind of tabu list was employed on those very fast tabu search algorithms proposed by Grabowski and Wodecki (2004, 2005), where it was successfully applied to the classical flow shop and job shop problems. Here, we extend this component of TS+M in the original form (Grabowski & Wodecki, 2004), to the problem considered. In this tabu list, length *LengthT* is a cyclic function shown in Fig. 4, and defined by the expression

$$\text{Length } T = \begin{cases} \text{LTS}, & \text{if } W(l) < iter \leq W(l) + h(l), \\ \text{LTS} + \psi, & \text{if } W(l) + h(l) < iter \leq W(l) + h(l) + H, \end{cases}$$

where $l=1,2,\ldots$ is the number of the cycle, $W(l)=\sum_{s=1}^{l} h(s-1)+(l-1)H$ (here $h(0)=0$). Further, $H$ is the width of the pick equal to $\psi$, and $h(l)$ is the interval between the neighbor picks equal to $3\times$LTS. If *LengthT* decreases then a suitable number of the oldest elements of tabu list $T$ is deleted and the search process is continued. The LTS and $\psi$ are tuning parameters which are to be chosen experimentally.

### 3.3. Search process

TS+M starts from an initial basic permutation $\pi$ in which the jobs of each T-block are ordered according to the WSPT rule. For this permutation, the value of $F(\pi)$ is calculated, and the neighborhood $N(ME,\pi)$ is created and searched in the following manner. First, the set of *unforbidden* moves (UF) that do not have the tabu status, is defined

$$\text{UME} = \{v \in ME | \text{move } v \text{ is } UF\},$$

and, the compound move $\hat{v}$ is created according to the method described earlier. If $\hat{v}=\varnothing$, we then take $\hat{v}:=v^*$, where $v^*$ is the 'best' move from UME, i.e. $F(\pi_{v^*})=\min_{v\in\text{UME}}F(\pi_v)$. If the compound move $\hat{v}$ is selected, then the triplets corresponding to the moves from $\hat{v}$ are added to the tabu list (see Section 3.2 for details) and the permutation $\pi_{\hat{v}}$ is

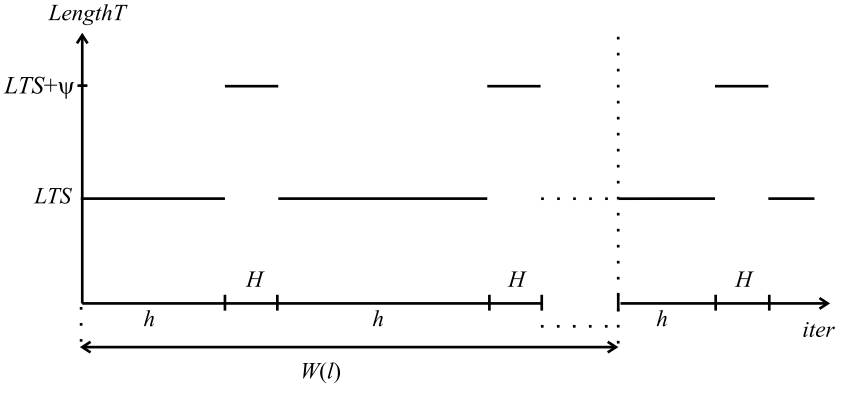


Fig. 4. Dynamic tabu list.

created. Next, within $\pi_{\hat{v}}$, the blocks are identified and the jobs of each T-block are ordered according to the WSPT rule. Then the resulting permutation becomes the new basic one, and algorithm restarts to next iteration.

If all moves from ME are forbidden (a very rare case), i.e. if UME $= \varnothing$, then the oldest element of tabu list is deleted and the search is repeated until a UF move is found.

It should be noted that the algorithm TS + M with compound moves is similar to the classic TS except that at each iteration a compound move $\hat{v}$ (containing several single moves) is performed, as opposed to a single move $v^*$.

## 4. Computational results

Our algorithm TS + M was coded in C + +, run on a PC with Celeron 450 MHz processor and tested on the benchmark instances. The results obtained by our algorithm were then compared with results from the literature. Several heuristic algorithms exist in the literature to solve the problem stated. So far the best approximation algorithms for the problem were presented in papers by Congram et al. (2002), Crauwels et al. (1998), Grosso et al. (2004) and Potts & Van Wassenhove (1991). The best speed has been obtained by Iterated Dynasearch Algorithm (here denoted as IDA) by Congram et al. (2002) that uses dynamic programming to search a neighborhood. However, the best results have been produced by Generalized Pairwise Interchanges Dynasearch, denoted as GPI-DS, by Grosso et al. (2004). The solution quality achieved by GPI-DS is better than the one of IDA, in about the same CPU time. In GPI-DS, a dynasearch neighborhood of IDA has been enhanced by applying additionally GPI operators and some elimination criteria. Both IDA and GPI-DS are non-deterministic ones because for each run of algorithms, the starting permutations are chosen randomly (in the tests of Grosso et al., 2004, for GPI-DS, 25 independent runs were performed). It is reported that both GPI-DS and IDA provide better results than the ones proposed by other authors. Therefore, most comparison of our algorithm TS + M are made with GPI-DS and IDA, but there are a few made with those TS(P,1) and TS(P,5) of Crauwels et al. (1998), that are currently the best algorithms based on tabu search approach for the problem considered. The difference between TS(P,1) and TS(P,5) is that if TS(P,1) performs *Maxiter* iterations, then TS(P,5), as distinct from TS(P,1), performs *Maxiter*/5 iterations, and it runs five times. Both TS(P,1) and TS(P,5) are non-deterministic ones because the diversification is made by randomly choice a permutation (from the neighborhood) which is an initial solution in the next iteration.

In order to gain more insight into the performance of the proposed algorithm TS + M, its behavior was analyzed, similarly to in IDA, GPI-DS, TS(P,1) and TS(P,5), on benchmark problems drawn from the OR-library (see http:// peo- ple.brunel.ac.uk/mastjjb/jeb/info.html). The benchmark set contains 375 particularly hard instances of three different sizes, selected from a large number of randomly generated problems. For each size $n = 40$, 50, and 100, a sample of 125 instances was provided.

Proposed algorithm need an initial permutation, which can found by any method. In our tests, we use algorithm AU Potts and Van Wassenhove (1991), which is considered to be the best one among simple constructive heuristics (based on the apparent urgency priority rule), to the problem considered. The complexity of this heuristic is $O(n^2)$. At the initial stage, our algorithm was run several times, for small-size instances generated randomly, in order to find the proper value of tuning parameters LTS and $\psi$. These were chosen experimentally as a result of the compromise between the running time and solution quality and we set LTS $= \psi = 20$.

For each test instance, we collected the following values:

- $F^A$—the cost function found by the algorithm $A \in \{$TS + M, IDA, GPI-DS, TS(P,1), TS(P,5)$\}$.
- PRD$(A) = 100(F^A - OPT)/OPT$—the percentage relative deviation of the cost function $F^A$ from the optimal (or best known) solution value $OPT$.
- CPU$(A)$—the computer time of algorithm $A$ (in seconds).

Then, for each size $n$, the following measures of the algorithm quality were calculated:

- $APRD(A)$—the average (for 125 instances) percentage relative deviation of the cost function found by algorithm $A$ from the optimal (or best known) solution value.
- $MPRD(A)$—the maximum (out of 125 instances) percentage relative deviation of the cost function found by algorithm $A$ from the optimal (or best known) solution value.
- $NO(A)$—the number of optimal (or best known) solution values found by algorithm $A$ out of 125 instances.

Table 2
Computational results of TS + M algorithm for $n$, $2n$, $n^2$ and $2n^2$ iterations

| $n$ | NI = $n$ | | | NI = $2n$ | | | NI = $n^2$ | | | NI = $2n^2$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | NO | APRD | MRPD | NO | APRD | MRPD | NO | APRD | MRPD | NO | APRD | MRPD |
| 40 | 124 | 0.002 | 0.151 | 125 | 0.000 | 0.000 | 125 | 0.000 | 0.000 | 125 | 0.000 | 0.000 |
| 50 | 115 | 0.005 | 0.171 | 117 | 0.004 | 0.101 | 124 | 0.000 | 0.029 | 125 | 0.000 | 0.000 |
| 100 | 94 | 0.007 | 0.210 | 105 | 0.005 | 0.175 | 122 | 0.001 | 0.162 | 125 | 0.000 | 0.000 |
| All | 333 | 0.005 | 0.174 | 347 | 0.003 | 0.092 | 371 | 0.000 | 0.064 | 375 | 0.000 | 0.000 |

- $CPU^{opt}_{min}(A)$—the minimal computer time to find the optimal value by algorithm $A$ out of 125 instances.
- $CPU^{opt}_{max}(A)$—the maximal computer time to find the optimal value by algorithm $A$ out of 125 instances.
- $CPU^{opt}_{ave}(A)$—the average (for 125 instances) computer time to find the optimal value by algorithm $A$.
- $NI(A)$—the number of iterations performed for $A \in \{TS + M, TS(P,1), TS(P,5)\}$, or number of descents (to a local optimum) performed for $A \in \{GPI\text{-}DS, IDA\}$.

Table 2 shows some detailed computational results of the proposed algorithm for different numbers (NI) of iterations. Note that for NI = $2n^2$, TS + M finds optimal (or best known) solution values for all the benchmark instances. Furthermore, while testing our algorithm, we detect the real NI at which TS + M has reached these values; it appears that the correspondent NI is equal to $1.2n^2$. For NI = $n$, the NO value is equal to 333 with APRD and MPRD equal to 0.005 and 0.174, respectively. So we can conclude that the convergence of TS + M is fairly good.

The comparison of TS + M with the best currently existing tabu search algorithms TS(P,1) and TS(P,5) is presented in Table 3. The results show that, in terms of PRD and NO values (number of optimal, or best known, solution values found by algorithms out of 125 instances), TS + M performs better than the existing tabu search algorithms of TS(P,1) and TS(P,5). Especially, the proposed algorithm is far superior to the TS(P,1). Inspection of the results reveals that the superiority of TS + M, over TS(P,1) and TS(P,5) increases as the number of jobs $n$ increases. Note that for $n$ iterations, TS + M found solutions with the overall average APRD and MPRD values equal to 0.005 and 0.174, respectively, whereas the best of existing tabu search algorithms TS(P,5) found the respective values equal to 0.025 and 1.660 for $2n^2$ iterations. We also observe that, in terms of the NO values, for $n$ iterations, the TS + M provided comparable value with TS(P,5) for $2n^2$ iterations.

Comparing the algorithms TS + M with IDA and GPI-DS, it should be highlighted that IDA and GPI-DS are non-deterministic ones. Thus, for the same data and the same parameter values, they can give different results, so, it possible that we do not obtain a satisfied solution in single run, even after performing 500 (or 1300) descents. Therefore, in order to evaluate their efficiency, it is necessary to run the algorithms multiple for each test instance, whereas TS + M, as deterministic one, needs a single run. In the tests of Grosso et al. (2004), IDA and GPI-DS were run 25 times for each instance, and the results were averaged over the runs (and are presented in Tables 4 and 5). These results show that TS + M and GPI-DS produce the best known solutions for all 375 benchmark instances, while IDA provide the ones for (on the average) 373.2 instances.

There are some complications involving the speed of computers used in the tests. Algorithm TS + M was run on Celeron 450 MHz, whereas GPI-DS was run on Kayak 800 MHz. In order to compare the CPU times of the different algorithms using the different computers, we need to normalize their speeds. One of the methods is that proposed by Dongarra, (2001), introducing the Computer-Independent CPU time, and then using proper conversion factors we can compare CPU times for different machines. However, the benchmark results in Dongarra tests can be used to give a

Table 3
Comparison between TS + M, TS(P,1) and TS(P,5) for NI = $2n^2$

| $n$ | Algorithm TS + M | | | Algorithm TS(P,1)[a] | | | Algorithm TS(P,5)[a] | | |
|---|---|---|---|---|---|---|---|---|---|
| | NO | APRD | MRPD | NO | APRD | MRPD | NO | APRD | MRPD |
| 40 | 125 | 0.00 | 0.00 | 115 | 0.06 | 6.70 | 118 | 0.00 | 0.33 |
| 50 | 125 | 0.00 | 0.00 | 111 | 0.01 | 0.42 | 113 | 0.01 | 0.28 |
| 100 | 125 | 0.00 | 0.00 | 96 | 0.06 | 4.78 | 103 | 0.04 | 4.39 |
| All | 375 | 0.00 | 0.00 | 332 | 0.043 | 3.966 | 334 | 0.025 | 1.660 |

[a] The results taken from Crauwels et al. (1998).

Table 4
Comparison between TS(P,1), TS(P,5), IDA, GPI-DS and TS + M

| Algorithm | NI | NO | | |
|---|---|---|---|---|
| | | $n=40$ | $n=50$ | $n=100$[a] |
| TS(P,1)[b] | $2n^2$ | 115 | 111 | 96 |
| TS(P,5)[b] | $5\times 2n^2/5$ | 118 | 113 | 103 |
| TS(P,1)[b] | $4n^2$ | 121 | 115 | – |
| TS(P,5)[b] | $5\times 4n^2/5$ | 123 | 118 | – |
| IDA[c] | 1300 descents | 125 | 125 | 123.2 |
| GPI-DS[c] | 500 descents | 125 | 125 | 125 |
| TS + M | $2n^2$ | 125 | 125 | 125 |

Number of optima's found.
[a] Only the best upper bounds are available for $n=100$.
[b] The results taken from Crauwels et al. (1998).
[c] The results, averaged over 25 runs, taken from Grosso et al. (2004).

Table 5
Comparison between CPI-DS and TS + M

| | Algorithm GPI-DS[a] | | | Algorithm TS + M | | |
|---|---|---|---|---|---|---|
| | $CPU_{\min}^{opt}$ | $CPU_{ave}^{opt}$ | $CPU_{\max}^{opt}$ | $CPU_{\min}^{opt}$ | $CPU_{ave}^{opt}$ | $CPU_{\max}^{opt}$ |
| 40 | 0.001 | 0.003 | 0.125 | 0.001 | 0.002 | 0.073 |
| 50 | 0.001 | 0.010 | 0.562 | 0.001 | 0.007 | 0.381 |
| 100 | 0.001 | 0.107 | 3.907 | 0.001 | 0.073 | 2.571 |

[a] The results, averaged over 25 runs, taken from Grosso et al. (2004). GPI-DS on Kayak 800 MHz Grosso et al. (2004). TS + M on Celeron 450 MHz.

rough guideline on the performance of different computers, since it refers to floating-point operations, thus cannot be representative for computer performing our algorithms that uses integer operations. Besides, the architecture, configurations, cache, main memory and compilers also affect on the CPU times. Another method is that presented in SPECint95 test (see: http://open.specbench.org/osg/cpu95/results/cint95.html) and Wintune97 database (see: http://www.freewareweb.com), which can allow us to find the proper (practical) coefficients. Then, we assumed that the compared algorithms had been implemented using integer variables, and we found that Celeron 450 MHz was 2.5–3.0 times slower than Kayak 800 MHz used in Grosso et al. (2004) for GPI-DS. Therefore, comparing the results given in Table 5, we can conclude that TS + M is about 4.5 times faster than GPI-DS.

All these results confirm the favorable performance of TS + M in the terms of CPU times and PRD values as well.

This promising conclusion encourages us to run an additional test in order to evaluate the quality of TS + M under a number of iterations larger than $2n^2$. A general purpose was to improve the best known solutions for the $n=100$ instances published recently in OR Library. We ran TS + M assuming NI $=20n^2$ but no better values were found.

## 5. Conclusions

This paper develops a new tabu search algorithm for the single machine problem to minimize total weighted tardiness. Some new properties of the problem associated with the blocks of jobs have been presented and discussed. The properties allow us to skip some non-perspective solutions during the search of the solutions space and to reduction the neighborhood size.

In order to decrease the computational effort for the search, we propose to use the compound move that consists in performing several moves simultaneously in each iteration of the algorithm and have a feature that their total contribution to changing the objective function is simply the sum of their separate contributions, without interacting effects that modify this total when the moves are performed together. It allows the algorithm to visit the more promising areas of the solutions space and to achieve very good solutions in a much shorter time, accelerating the convergence of the algorithm. Also, we propose a tabu list with dynamic length. This length is changed cyclically using a 'pick' that can be viewed as a specific disturbance, giving an additional assistance to avoid getting trapped at a

local optimum. Besides, the computational effort of our algorithm is decreased by applying the idea which improves the complexity for the search of neighborhood from $O(n^3)$ to $O(n^2)$, and by reducing of neighborhood size, using some elimination criteria.

Computational experiments are given and compared with the results yielded by the best algorithms discussed in the literature. These results show that the proposed algorithm provides better results than attained by the leading approaches.

**Acknowledgements**

This research has been supported by KBN Grant 4 T11A 016 24. The authors are due to anonymous referees for their valuable comments and suggestions. The present version of the paper has benefited greatly from these comments.

## Appendix A

### *A.1. Proof of Theorem 1 (by contradiction)*

Without loss of generality and for the simplicity of denotation one can assume that $\pi(i)=i$. Thus, $\pi$ takes the form

$$\pi = (1,2,\ldots,l_1,f_2,f_2+1,\ldots,l_2,\ldots f_k,f_k+1,\ldots,l_k,\ldots f_m,f_m+1,\ldots,n),$$

where

$$B_k = (f_k,f_k+1,\ldots,l_k), \quad k = 1,2,\ldots,m,$$

and

$$F(\pi) = \sum_{k=1}^{m} F_k(\pi) = \sum_{k=1}^{m}\sum_{j=f_k}^{l_k} w_j T_j.$$

Suppose that the theorem is false, and let $\beta$ be an arbitrary permutation of the following form

$$\beta = (x_1^1,x_2^1,\ldots,x_{t_1}^1,x_1^2,x_2^2,\ldots,x_{t_2}^2,\ldots,x_1^k,x_2^k,\ldots,x_{t_k}^k,\ldots,x_1^m,x_2^m,\ldots,x_{t_m}^m), \tag{A1}$$

where $X_k = (x_1^k,x_2^k,\ldots,x_{t_k}^k)$, $k=1,2,\ldots,m$, is any permutation of $(f_k,f_k+1,\ldots,l_k)$, i.e. any permutation of the jobs from block $B_k$ of $\pi$. It is easy to verify that the thesis of the theorem does not hold for $\beta$.

Now, for permutation $\beta$, we have

$$F(\beta) = \sum_{k=1}^{m} F_k(\beta) = \sum_{k=1}^{m}\sum_{j\in X_k} w_j T_j.$$

Since $\{x_1^k,x_2^k,\ldots,x_{t_k}^k\} = \{f_k,f_k+1,\ldots,l_k\}$ for $k=1,2,\ldots,m$, then

$$F_k(\pi) = F_k(\beta),$$

whenever the permutations of jobs within both $X_k$ and $B_k$ are the same. Hence, since the jobs from $X_k$ of $\beta$ are ordered in any permutation, then, for each T-block $\vec{B}_k^{\mathrm{T}}$ of $\pi$, using (2), we get

$$F_k(\beta) = \sum_{j\in X_k} w_j T_j \geq \sum_{j\in \vec{B}_k^{T}} w_j T_j = \vec{F}_k(\pi),$$

where the jobs from $\vec{B}_k^{\mathrm{T}}$ are ordered by the WSPT rule, according to the assumption of the theorem. Further, for each E-blocks $B_k^{\mathrm{E}}$ of $\pi$, using (1), we have

$$F_k(\beta) = F_k(\pi) = 0.$$

Therefore, for each block $B_k$ of $\pi$, we get

$$F_k(\beta) \geq F_k(\pi).$$

Hence, for any permutation $\beta$ given by (A1), we get

$$F(\beta) = \sum_{k=1}^{m} F_k(\beta) \geq \sum_{k=1}^{m} F_k(\pi) = F(\pi),$$

which contradicts the assumption of the theorem. □

## References

Abdul-Razaq, T. S., Potts, C. N., & Van Wassenhove, L. N. (1990). A survey of algorithms for the single machine total weighted tardiness scheduling problem. *Discrete Applied Mathematics*, *26*, 235–253.

Angel, E., & Bampis, E. (2005). A multi-start dynasearch algorithm for the time dependent single-machine total weighted tardiness scheduling problem. *European Journal of Operational Research*, *162*, 281–289.

Cheng, T. C. E., Ng, C. T., Yuan, J. J., & Liu, Z. H. (2005). Single machine scheduling to minimize total weighted tardiness. *European Journal of Operational Research*, *165*, 423–443.

Congram, R. K., Potts, C. N., & Van de Velde, S. L. (2002). An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, *14*(1), 52–67.

Crauwels, H. A. J., Potts, C. N., & Van Wassenhove, L. N. (1998). Local search heuristics for the single machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, *10*(3), 341–350.

Den Basten, M., Stützle, T., & Dorigo, M. (2000). Ant colony optimization for the total weighted tardiness problem. *Proceedings of PPSN-VI, LNCS* ( Vol. 1917), pp. 611–620.

Den Basten, M., Stützle, T., & Dorigo, M. (2001). *Design of iterated local search algorithms. An example application to the single machine total weighted tardiness problem Evo worskshop, LNCS,* ( Vol. 2037), pp. 441–451.

Dongarra, J. J. (2001). Performance of various computers using standard linear equations software. Working paper, Tennessee: Computer Science Department, University of Tennessee. http://www.netlib.org/benchmark/performance.ps

Emmons, H. (1969). One machine sequencing to minimize certain functions of job tardiness. *Operations Research*, *17*, 701–715.

Fisher, M. L. (1976). A dual algorithm for the one machine scheduling problem. *Mathematical Programming*, *11*, 229–252.

Glover, F. (1989). Tabu search. Part I. *ORSA Journal on Computing*, *1*, 190–206.

Glover, F. (1990). Tabu search. Part II. *ORSA Journal on Computing*, *2*, 4–32.

Glover, F., & Laguna, M. (1998). *Tabu search.* Boston, MA: Kluwer.

Grabowski, J., & Pempera, J. (1998). Some algorithms of sequencing for total weighted tardiness problem. *Zeszyty Naukowe Politechniki Slaskiej*, *125*, 38–45.

Grabowski, J., & Pempera, J. (2001). New block properties for the permutation flow-shop problem with application in TS. *Journal of the Operational Research Society*, *11*, 210–220.

Grabowski, J., & Pempera, J. (2005). Some local search algorithms for no-wait flow shop problem with makespan criterion. *Computers and Operations Research*, *32*, 2197–2212.

Grabowski, J., & Wodecki, M. (2004). A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion. *Computers and Operations Research*, *31*, 1891–1909.

Grabowski, J., & Wodecki, M. (2005). A very fast tabu search algorithm for the job shop problem. In C. Rego, & B. Alidaee (Eds.), *Metaheuristic optimization via memory and evolution; tabu search and scatter search* (pp. 117–144). Boston, MA: Kluwer.

Grosso, A., Della Croce, F., & Tadei, R. (2004). An enhanced dynasearch neighborhood for single-machine total weighted tardiness scheduling problem. *Operations Research Letters*, *32*, 68–72.

Lawler, E. L. (1977). A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness. *Annals of Discrete Mathematics*, *1*, 331–342.

Lawler, E. L. (1979). Efficient implementation of dynamic programming algorithms for sequencing problems. Technical report, BW-106. Amsterdam: Centre for Mathematics and Computer Science.

Lenstra, J. K., Rinnooy Kan, A. G. H., & Brucker, P. (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, *1*, 343–362.

Matsuo, H., Otto, S. W., & Sullivan, R. S. (1987). A controlled search simulated annealing method for the single machine weighted tardiness problem. Working paper no. 03-44-87. Austin: Department of Management, University of Texas.

Morton, T. E., Rachamadugu, R. M., & Vepsalainen, A. (1984). Accurate myopic heuristic for tardiness scheduling. Working paper no. 36-83-84. Pitsburgh: Carnegie-Mellon University.

Potts, C. N., & Van Wassenhove, L. N. (1985). A branch and bound algorithm for the total weighted tardiness problem. *Operations Research*, *33*, 177–181.

Potts, C. N., & Van Wassenhove, L. N. (1991). Single machine tardiness sequencing heuristics. *IIE Transactions*, *23*, 346–354.

Rinnooy Kan, A. H. G., Lageweg, B. J., & Lenstra, J. K. (1975). Minimizing total costs in one-machine scheduling. *Operations Research*, *25*, 908–927.

Shwimer, J. (1972). On the *n*-job, one-machine, sequence-independent scheduling problem with tardiness penalties: A branch-and-bound solution. *Management Science*, *18*, 301–313.

Smith, W. E. (1956). Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, *3*, 59–66.