AUTOMATYKA • 2005 • Tom 9 • Zeszyt 1–2

Wojciech Bożejko*, Józef Grabowski*, Mieczyslaw Wodecki**

# Block Approach-Local Search Algorithm for Single Machine Total Weighted Tardiness Problem***

## 1. Introduction

In the single machine total weighted tardiness problem (TWTP), denoted as $1\|\sum w_iT_i$ a *set of jobs* $N = \{1, 2, ..., n\}$ have to be processed without interruption on a single machine that can handle only one job at a time. 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* $f_i(C_i) = w_iT_i$ 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 $\sum_{i=1}^{n} f_i(C_i) = \sum_{i=1}^{n} w_iT_i$.

The total weighted tardiness problem is *NP*-hard in strong sense (Lawler [11] and Lenstra *et al*. [12]). 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.

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 *et al.* [3]), Simulated Annealing (Matsuo *et al*. [13]), Genetic Algorithms (Crauwels *et al*. [3]), Ant Colony Optimization (Den Basten *et al*. [4]). A very effective local-search method has been proposed by Congram *et al*. [2], and next improved by Grosso *et al*. [10]. 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 a specific neighborhood which employs the properties and a compound moves technique.

* Institute of Engineering Cybernetics, Wroclaw University of Technology
** Institute of Computer Science, University of Wroclaw
*** This research was supported by KBN Grant 4 T11A 016 24

## 2. Problem description and preliminaries

Each schedule of jobs can be represent by permutation $\pi = (\pi(1), \pi(2), ..., \pi(n))$ 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. $C_{\pi(i)} \le d_{\pi(i)}$), 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 $m$ ($m \le n$) subsequences $B_1, B_2, ..., B_m$, called *blocks* in $\pi$, each of them contains the jobs having in common specific properties, where:

1. $B_k = (\pi(f_k), \pi(f_k + 1), ..., \pi(l_k - 1)), \pi(l_k))$, $l_{k-1} + 1 = f_k \le l_k$, $k = 1, 2, ..., m$, $l_0 = 0$, $l_m = n$.
2. All the jobs $j \in B_k$ satisfy the following condition:
   either
   $$d_j \ge C_{\pi(l_k)} \qquad (C1)$$
   or
   $$d_j \le S_{\pi(f_k)} + p_j \qquad (C2)$$
   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 $C2$.

By the 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 $B_k^E$ and $B_k^T$), respectively. Jobs $\pi(f_k)$ and $\pi(l_k)$ in $B_k$ are the *first* and *last* ones, respectively.

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

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

**Theorem 1** [1]. *Let $\pi \in \Pi$ be any permutation with blocks $B_1, B_2, ..., B_m$, 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 $B_1, B_2, ..., B_{k-1}$, for some $k \in \{2, 3, ..., m\}$, or*

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

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

## 3. Algorithm Tabu Search with compound moves (TS+C)

Currently, Tabu Search approach, (see Glover [6]) 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 neighbourhood, a set of permutations generated by the moves, for a permutation with the lowest makespan. 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 neighbourhood. 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 trapped at a local optimum. The algorithm TS terminates when a given number of iterations has been reached without improvement of the best current makespan, the algorithm has performed a given number of iterations (*Maxiter*), time has run out, the neighbourhood is empty, a permutation with a satisfying makespan has been found, etc.

In our algorithm, denoted here as TS+M, are used some of the components that have been proposed by Grabowski and Wodecki [8, 9], where they were successfully applied on those very fast tabu search algorithms for the classical flow shop and job shop problems. In this paper, we extend these elements in the original or modified form, to the problem considered.

### 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 neighbourhood. 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 [7]). 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 $\pi$, $x, y \in \{1, 2, ..., n\}$, $x \neq y$. 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:

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

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

(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

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

Since, by the 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 $x < y$.

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

In our tabu search algorithm TS+C, we will employ the set of I-moves

$$IM = \bigcup_{k=1}^{m-1} \bigcup_{x=f_k}^{l_k} \left[\{(x, y) \mid f_{k+1} \leq y \leq n\} \cup \{(x, y) \mid 1 \leq y \leq l_{k-1}\}\right],$$

and the set of S-moves

$$SM = \bigcup_{k=1}^{m-1} \bigcup_{x=f_k}^{l_k} \{(x, y) \mid f_{k+1} \leq y \leq n\}.$$

Finally, in our algorithm, we use the following set of moves

$$M = IM \cup SM,$$

which creates neighbourhood $\mathcal{N}(M, \pi)$.

From the definition of $M$, it follows that each move $v \in M$ satisfies the necessary condition of Theorem 1.

In order to decrease the computational effort for the search, we propose a reduction of the neighbourhood size by using 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 $j \in a(i)$. Let $\overline{a}(j)$, $j = 1, 2, ..., n$, be the complement set of $a(j)$, i.e. $\overline{a}(j) = N - a(j)$. The sets $a(j)$ and $b(j)$ can be created by application of the following theorem.

**Theorem 2** [15]. *There exists an optimal permutation whereby job i precedes job j if at least one of the conditions (1)–(3) is satisfied:*

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

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

$$IE(\pi) = \{(x, y) \in IM \mid \pi(x) \in b(j),\ j \in \{\pi(x+1), ..., \pi(y)\},\ x < y\} \cup$$

$$\cup\, (x, y) \in IM \mid \pi(x) \in a(j),\ j \in \{\pi(y), ..., \pi(x-1)\},\ x < y\}.$$

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

$$SE(\pi) = \{(x, y) \in SM \mid \pi(x) \in b(j),\ j \in \{\pi(x+1), ..., \pi(y)\},\ x < y\} \cup$$

$$\cup (x, y) \in SM \mid \pi(y) \in a(j),\ j \in \{\pi(x), ..., \pi(y-1)\},\ x < y\}.$$

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

In order to accelerate the convergence of the algorithms to good solutions, we propose to use the *compound moves*, introduced by Glover and Laguna (see [6]). A compound moves consists 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. 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

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

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-moves or S-moves.

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:

$$max(x_1, y_1) < min(x_2, y_2) \text{ or } max(x_2, y_2) < min(x_1, y_1) \qquad (1)$$

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 Conditions (1). 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 $\mathcal{N}(ME, \pi)$, 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 $\hat{v} = \emptyset$, then the compound move can not be used. The procedure that creates a compound move $\hat{v}$ is based on a greedy algorithm.

### 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+C increases, using a "pick" that can be viewed as a specific disturbance (diversification).

This kind of tabu list was employed on those very fast tabu search algorithms proposed by Grabowski and Wodecki, where it was successfully applied to the classical flow shop and job shop problems [8, 9]. Here, we extend this component of TS+C in the original form [8], to the problem considered. In this tabu list, length *LengthT* is a cyclic function shown in Figure 1, and defined by the expression:

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

where:

$l = 1, 2, \ldots$ – the number of the cycle,

$W(l) = \sum_{s=1}^{l} h(s-1) + (l-1) \times H$ (here $h(0) = 0$),

$H$ – the width of the pick equal to $\psi$,

$h(l)$ – the interval between the neighbour 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.

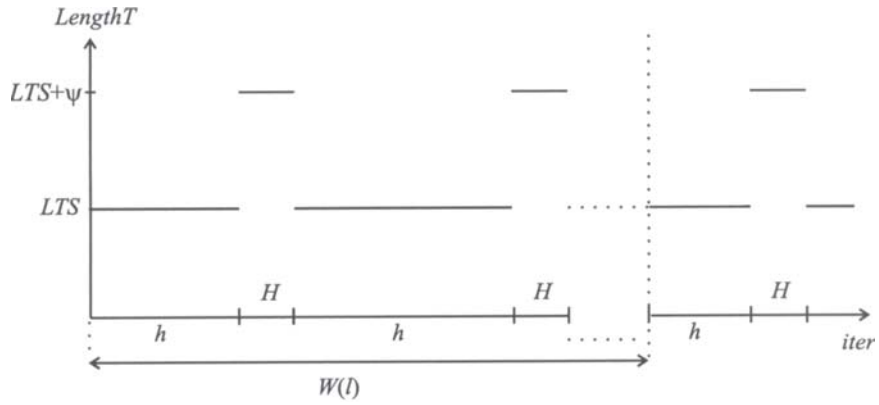


**Fig. 1.** Dynamic tabu list

### 3.3. Search process

TS+C starts from an initial basic permutation $\pi$ in which the jobs of each T-block are ordered according to the WSPT rule. This permutation implies the neighbourhood $\mathcal{N}(ME, \pi)$ that is searched in the following manner. First, the set of *unforbidden* moves (UF) that do not have the *tabu* status, is defined

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

and, the compound move $\hat{v}$ is created according to the method described earlier. If $\hat{v} = 0$, we then take $\hat{v} := v^*$, where $v^*$ is the "best" move from *UME*, i.e. $F(\pi_{v^*}) = \min_{v \in UME} F(\pi_v)$. 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 *Tabu list and tabu status of move* for details) and the permutation $\pi_{\hat{v}}$ is created. Next, the jobs of each T-block within $\pi_{\hat{v}}$ are ordered according to the WSPT rule, and 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 $UME = \emptyset$, then the oldest element of tabu list is deleted and the search is repeated until a *UME*-move is found.

It should be noted that the algorithm TS+C 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+C was 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 Crauwels *et al*. [3], Den Basten

*et al*. [4], Congram *et al*. [2] and Grosso *et al*. [10]. The best speed has been obtained by Iterated Dynasearch Algorithm (here denoted as IDA) by Congram et al. [2], that uses dynamic programming to search a neighbourhood. However, the best results have been produced by Generalized Pairwise Interchanges Dynasearch, denoted as GPI-DS, by Grosso *et al*. [10]. 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 neighbourhood 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 [10], 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+C 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.* [3], 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 5 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 neighbourhood) which is an initial solution in the next iteration.

In order to gain more insight into the performance of the proposed algorithm TS+C, its behavior was analyzed, similar to in IDA, GPI-DS, TS(P,1) and TS(P,5), on benchmark problems drawn from the OR-library [14]. The benchmark set contains 375 particularly hard instances of 3 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 [14].

Proposed algorithm, at the initial stage 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+C, IDA, GPI-DS, TS(P,1), TS(P,5)},

$CPU(A)$ – the computer time of algorithm $A$ (in seconds),

$PRD(A) = 100(F^A - OPT) / OPT$ – the percentage relative deviation of the cost function $F^A$ from the optimal (or best known) solution value $OPT$.

Then, for each size (group) $n$, 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;

$CPU_z^{opt}(A)$ – the computer time to find the optimal value by algorithm $A$ out of 125 instances, $z \in \{minimal, average, maximal\}$;

$NI(A)$ – the number of iterations performed for $A \in$ {TS+C, TS(P,1), TS(P,5)}, or number of descents (to a local optimum) performed for $A \in$ $\in$ {GPI-DS, IDA}.

Table 1 shows some detailed computational results of the proposed algorithm for different numbers (NI) of iterations. Note that for NI=$2n^2$, TS+C finds optimal (or best known) solution values for all the benchmark instances. For NI = $n$, this value is 333 with APRD and MPRD equal to 0.005 and 0.174, respectively. So we can conclude that the convergence of TS+C is fairly good.

**Table 1**
Results of TS+C 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** |

The comparison of TS+C with the best currently existing tabu search algorithms TS(P,1) and TS(P,5) is presented in Table 2. Note that both algorithms TS(P,1) and TS(P,5) are nondeterministic ones, so it possible that we do not obtain a satisfied solution in single run of this algorithm, even after performing many iterations, therefore, it is necessary to run algorithm multiple, starting from randomly choice initial permutations, whereas TS+C is deterministic one, thus the result is obtained in single run. The results from Table 2 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+C 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+C, over TS(P,1) and TS(P,5) increases as the number of jobs $n$ increases. Note that for $n$ iterations, TS+C found solutions with the overall average APRD and MPRD values equal to 0.005 and 0.233, respectively, whereas the best of existing tabu search algorithms TS(P,5) found the respective values equal to 0.025 and 1.660 for $n^2$ iterations. We also observe that, in terms of the NO values, for $n$ iterations, the TS+C provided comparable value with TS(P,5) for $n^2$ iterations.

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

| $n$ | Algorithm TS+C | | | Algorithm TS(P,1) | | | Algorithm TS(P,5) | | |
|---|---|---|---|---|---|---|---|---|---|
| | 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** | **1.66** |

All these results, presented in Table 3, confirm the favorable performance of TS+C in the terms of CPU times and PRD values as well. This promising conclusion encourage us to run an additional test in order to evaluate the quality of TS+C 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+C assuming NI = $20n^2$ but no better values were found.

**Table 3**
Comparison between GPI-DS and TS+C

| $n$ | Algorithm GPI-DS* | | | Algorithm TS+C | | |
|---|---|---|---|---|---|---|
| | $CPU^{opt}_{min}$ | $CPU^{opt}_{ave}$ | $CPU^{opt}_{max}$ | $CPU^{opt}_{min}$ | $CPU^{opt}_{ave}$ | $CPU^{opt}_{max}$ |
| 40 | 0.001 | 0.003 | 0.125 | 0.001 | 0.003 | 0.102 |
| 50 | 0.001 | 0.010 | 0.562 | 0.001 | 0.009 | 0.529 |
| 100 | 0.001 | 0.107 | 3.907 | 0.001 | 0.105 | 3.881 |

* Averaged values over 25 runs [10].
GPI-DS on Kayak 800 MHz [10].
TS+C on Celeron 450 MHz.

## 5. Conclusions

In this paper, the problem of scheduling a given jobs on a single machine to minimize total weighted tardiness of jobs is presented. 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. In order to decrease the computational effort for the search, in our algorithms, we propose to use the compound moves that consist in performing several moves simultaneously in a single iteration of algorithms and guide the search process to more promising areas of the solutions space, where "good solutions" can be found. It allows the algorithm to achieve very good solutions in a much shorter time. Also, we propose a tabu list with dynamic length which is changed cyclically, as the current iteration number of algorithms increases, using the "pick" in order to avoid trapped at a local optimum.

Computational experiments are given and compared with the results yielded by the best algorithms discussed in the literature. These results show that algorithm proposed provides better results than in the recent modern heuristics. Nevertheless, some improvements in our algorithm are possible. For instance, attempts to refine the compound moves should reflect a further improvement of the computational results. It might be interesting to develop new more sophisticated neighbourhoods, and to combine them in the series and/or parallel structures, creating new algorithms.

The results obtained encourage us to extend the ideas proposed with different objective functions or to other sequencing problems.

## References

[1] Bożejko W., Grabowski J., Wodecki M.: *A block approach tabu search algorithm for single machine total weihhted tardiness problem*. Technical Report 2/2005, Institute of Engineering Cybernetics Wrocław University of Technology

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

[3] Crauwels H.A.J., Potts C.N., Van Wassenhove L.N.: *Local Search Heuristics for the Single Machine Total Weighted Tardiness Scheduling Problem*. INFORMS Journal on Computing, 10(3), 1998, 341–350

[4] Den Basten M., Stützle T., Dorigo M.: *Design of Iterated Local Search Algorithms an Example Application to the Single Machine Total Weighted Tardiness Problem*. In: EvoWorskshop LNCS 2037 (J.W. Boers *et al*.) 2001, 441–451

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

[6] Glover F., Laguna M.: *Tabu Search*. Kluwer Academic Publishers, 1998

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

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

[9] Grabowski J., Wodecki M.: *A very fast tabu search algorithm for the job shop problem*. In: Metaheuristic Optimization via Memory and Evolution, Tabu Search and Scatter Search, (C. Rego, B. Alidaee (Eds)), Kluwer Academic Publishers 2005

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

[11] Lawler E.L.: *A Pseudopolynomial Algorithm for Sequencing Jobs to Minimize Total Tardiness*. Annals of Discrete Mathematics, 1, 1977, 331–342

[12] Lenstra J.K., Rinnoy Kan A.G.H., Brucker P.: *Complexity of Machine Scheduling Problems*. Annals of Discrete Mathematics, 1, 1977, 343–362

[13] Matsuo H., Otto S.W., Sullivan R.S.: *A controlled search simulated annealingmethod for the single machine weighted tardiness problem*. Working paper 87-12-2, Department of Management, University of Texas at Austin, TX USA 1987

[14] OR Liblary http.//www.ms.ic.ac.uk/info.html

[15] Rinnoy Kan A.H.G., Lageweg B.J., Lenstra J.K.: *Minimizing total costs in one-machine scheduling*. Operations Research, 25, 1975, 908–927