Int J Adv Manuf Technol (2009) 40:797–807
DOI 10.1007/s00170-008-1395-7

ORIGINAL ARTICLE

A block approach to earliness-tardiness scheduling problems

Mieczysław Wodecki

Received: 28 July 2007 / Accepted: 9 January 2008 / Published online: 28 February 2008

Abstract In this paper, we consider a single-machine job scheduling problem where the objective is to minimize the weighted sum of earliness and tardiness (E/T) penalties of jobs. This problem is consistent with the just-in-time (JIT) production. We propose partitioning of permutation into subsequences (blocks) and replacing sets of moves with its representatives, significantly decreasing the size of the searched neighborhood. Some new properties of the problem and compound moves make eliminating "bad" elements and speeding up calculations possible. These properties allow us to propose a new fast local search algorithm based on a tabu search method. Computational experiments are presented and the results show that the algorithm proposed allows us to obtain the best-known results in a short time.

Keywords Scheduling · Single machine · Tabu search

1 Introduction

The problem considered in this paper deals with a set on n jobs (enumerated by $1, 2, ..., n$) on a single machine which is capable of processing only one job at a time without preemption. With each job i is associated *processing time* p_i. By e_i and d_i we mean adequately demanded the *earliest* and the *latest moment* of finishing processing of a job i. The *earliness* of job i is defined as $E_i = \max\{0, e_i - C_i\}$ and its *tardiness* as $T_i = \max\{0, C_i - d_i\}$, where C_i is the completion time of job i. An expression $f_i(C_i) = u_i E_i + w_i T_i$ is the *cost* of job execution, where u_i and w_i are nonnegative coefficients of a goal function. The objective is to find a sequence of jobs that minimize the following non-regular function:

$$\sum_{i=1}^{n} (u_i E_i + w_i T_i).$$

This problem became more popular after the introduction of JIT manufacturing philosophy, where jobs are desired to be completed as close as possible to their due dates. The assumption of no idle time on the machine represents the situation where the machine idleness cost is higher than the earliness cost incurred by finishing a job before its due date and the machine have to be kept running. Examples of production systems where such a situation can be observed have been given by Korman [16], Landis [17] and Schaller [21]. In the classical scheduling notation in literature, the problem is denoted by $1||\sum(u_i E_i + w_i T_i)$ and it belongs to a strong NP-hard class (if we assume that $u_i = 0$, $i = 1, 2, \dots, n$, we obtain a strong NP-hard problem $1||\sum w_i T_i$ - Lawler [13] and Lenstra et al. [18]).

Many authors have studied the earliness and tardiness (E/T) problem which was first introduced by Kanett [15]. A useful review of early/tardy scheduling is provided in Baker and Scudder [1] and the book T'kindt and Billaut [24]. In paper [1] Baker and Scudder proved that there can be an idle time in an optimal solution (jobs don't need to be processed directly one after another), that is $C_{i+1} \geq C_i + p_{i+1}$, $i = 1, 2, \dots, n-1$. Solving the problem amounts to

M. Wodecki (✉)
Institute of Computer Science, University of Wrocław,
Joliot-Curie 15, 50-383 Wrocław, Poland
e-mail: mwd@ii.uni.wroc.pl

establishing a sequence of jobs and their starting times. Hoogeven and van de Velde [12] proposed an algorithm based on a branch-and-bound method. Because of the exponentially growing computation time, this algorithm can only be used to solve instances where the number of jobs is no larger than 20. Therefore, in practice, almost always the approximate algorithms are used. The best of them are based on artificial intelligence methods. Calculations are performed in two stages:

1. determining the sequence of jobs (with no idle times).
2. establishing jobs' optimal starting times.

There is an algorithm in the paper of Wan and Yen [26] based on this scheme. To determine scheduling, a tabu search algorithm is used. Algorithms for optimal job sequencing are relatively less studied. Szwarc [23] proposed a branch-and-bound algorithm based on some adjacent ordering conditions for jobs with distinct penalty weights. Lee and Choi [14] proposed a genetic algorithm and Yano and Kim [28] studied several dominance properties for sequencing jobs with penalty weights proportional to processing times.

The most important results for unrestrictive and restrictive scheduling E/T problems are presented in the papers of Bank and Werner [2], Gordon et al. [8], Valente and Alves [27], Feldmann and Biskup [5], and Tung-I Tsai [25].

In this paper, we consider *TWET* problem additionally assuming that the machine begins the execution of jobs in time zero and it works with no idleness (*TWET-no-idle* problem). We present new elements of the neighborhood search method. Partition of permutation into blocks (subsequences) and replacement sets of moves with their representatives significantly decrease size of the neighborhood, eliminating "bad" moves and speeding up the calculations. Computational experiments follow that such a neighborhood is not bigger (in average) than the size of classical neighborhood generated only by insert moves.

The rest of the paper is organized as follows: in Section 1 notations and basic definitions (block in permutations) are introduced. Section 2 presents new elements of the neighborhood structure: partitioning of the permutation into blocks and elimination of unnecessary moves. In the Section 3 we propose new methods of determining representatives of the set of moves. A conclusion is given in Section 4.

Neighborhood-based metaheuristics are extensions of iterative improvement algorithms which avoid getting stuck in locally optimal solution by allowing to move to worse solutions in one's neighborhood. The local search (LS) method is designed to find a near-optimal solution to combinatorial optimization problems. For any iteration of a local search algorithm, a subset of moves applicable to it is defined. This subset of moves generates a subset of solutions—the neighborhood. Every move transforms a permutation (current solution) into another permutation. The basic version of LS starts from an initial solution x^0. The elementary step of the method performs, for a given solution x^i, a search through the neighborhood $\mathcal{N}(x^i)$ of x^i. The neighborhood $\mathcal{N}(x^i)$ is defined by moves performed from x^i. A move transforms a solution into another solution. The aim of this elementary search is to find $\mathcal{N}(x^i)$ a solution x^{i+1} with the lowest cost functions. Then the search is repeated itself from the best solution as a new starting point and the process is continued. Simulated annealing and tabu search methods belong to such a type of metaheuristics.

Let k and l $(k \neq l)$ be a pair of positions in a permutation:

$$\pi = (\pi(1), \pi(2), \ldots, \pi(k-1), \pi(k), \pi(k+1), \ldots, \pi(l-1), \pi(l), \pi(l+1), \ldots, \pi(n)).$$

Among many types of moves considered in literature, two of them appear prominently:

1. Insert move (*i-move*) i_l^k, consists in removing the job $\pi(k)$ from the position k and inserting it on the position l. Thus the move i_l^k generates a new permutation $i_l^k(\pi) = \pi_l^k$.
2. Swap move (*s-move*) s_l^k, in which the jobs $\pi(k)$ and $\pi(l)$ are swapped among some positions k and l. The move s_l^k generates permutation $s_l^k(\pi) = \pi_l^k$.

Execution of the *i*-move can be realized in time $O(1)$ (if double linked lists are used), execution of the *s*-move – in constant time $O(1)$ in the classic linear representation of permutation.

1.1 Blocks of jobs in permutation

For the *TWET-no-idle* problem, each schedule of jobs can be represented by permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n-1), \pi(n))$ of the set of jobs $\mathcal{J} = \{1, 2, \ldots, n\}$. Let $\Phi(n)$ denote the set of all such permutations.

The total cost (the goal function) of $\pi \in \Phi(n)$ is $F(\pi) = \sum_{i=1}^{n} f_{\pi(i)}(C_{\pi(i)})$, where $C_{\pi(i)}$ is the time when the job $\pi(i)$ is completed, $C_{\pi(i)} = \sum_{j=1}^{i} p_{\pi(j)}$. The job $\pi(i)$ is considered as *early* if it is completed before its earliest moment of finishing $(C_{\pi(i)} < e_{\pi(i)})$, *on time* if $e_{\pi(i)} \leq C_{\pi(i)} \leq d_{\pi(i)}$, and *tardy* if the job is completed after its due date (i.e., $C_{\pi(i)} > d_{\pi(i)}$).

For further considerations, we refer to the notions and notations taken from the papers [4].

Table 1 Data of the instance

i	1	2	3	4	5	6	7	8	9	10	11
p_i	2	3	1	2	3	2	3	3	2	2	2
e_i	1	3	1	5	1	3	7	22	23	11	14
d_i	12	19	12	6	6	6	9	24	24	15	15
u_i	2	1	3	1	4	2	6	6	2	3	2
w_i	3	1	2	6	5	4	3	2	4	5	5

Each permutation $\pi \in \Phi(n)$ is decomposed into subsequences of jobs $\mathcal{B} = [B^1, B^2, \dots, B^v]$ called *blocks*. Each of these blocks contain the jobs and:

1. $B^i = (\pi(a^i), \pi(a^i+1), \dots, \pi(b^i-1), \pi(b^i))$, $a^i = b^{i-1}+1, \quad 1 \le i \le v, \quad b^0 = 0$.
2. All the jobs $j \in B^i$ satisfy one of the following conditions:
   $e_j > C_{\pi(b^i)}$, or (C1)
   $e_j \le C_{\pi(b^{i-1})} + p_j$ and $d_j \ge C_{\pi(b^i)}$, or (C2)
   $d_j < C_{\pi(b^{i-1})} + p_j$. (C3)
3. B^i are maximal subsequences of π in which all the jobs satisfy either condition $C1$ or condition $C2$ or condition $C3$.

By definition, there exist three types of blocks implied by either C1 or C2 or C3. To distinguish them, we will use the $\mathcal{E}$*-block*, $\mathcal{O}$*-block* and $\mathcal{T}$*-block* notations, respectively.

It is easy to prove that if B^i is $\mathcal{E}$-block, then $\min\{e_j : j \in B^i\} > C_{last}$, where $last = \pi(b^i)$. So in any permutation of jobs from B^i, every job is executed early in the permutation π. From this property we propose an algorithm to determine the first $\mathcal{E}$-block in the permutation π.

Algorithm A$\mathcal{E}$-block

```
Input: permutation $\pi = (\pi(1), \pi(2), \dots, \pi(n))$;
Output: subsequence ($\mathcal{E}$-block)
        $B = (\pi(l), \pi(l+1), \dots, \pi(k-1), \pi(k))$;
Let $\pi(l)$ be the first job in $\pi$ such, that $C_{\pi(l)} < e_{\pi(l)}$.
$B \leftarrow \pi(l)$;   $k \leftarrow l$;
while $|B| = k - l + 1$ and $k < n$ do
    if $C_{\pi(k+1)} < e_{\pi(k+1)}$ and
        if $(\forall \ \pi(i) \in B, \ C_{\pi(k+1)} < e_{\pi(i)})$ then
            $B \leftarrow B \cup \{\pi(k+1)\}$;
    $k \leftarrow k+1$
end.
```

Computational complexity of the algorithm is $O(n)$. Similarly, one can present an algorithm determining the first $\mathcal{O}$ and $\mathcal{T}$-block. Considering a permutation π, starting from $\pi(1)$ and applying appropriate algorithm determining $\mathcal{E}$, $\mathcal{O}$ or $\mathcal{T}$-block we can partition π into $\mathcal{E}$, $\mathcal{O}$ and $\mathcal{T}$ blocks. Complexity of partition procedure is O(n).

Example 1 Let us consider the problem's instance of n=11 jobs, which is specified in Table 1.

Permutation $\pi = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)$ has four blocks, v=4, a^1=1, b^1=3, a^2 = 4, b^2=7, a^3=8, b^3 = 9, a^4=10, b^4=11 and B^1=(1, 2, 3), B^2=(4, 5, 6, 7), B^3 = (8, 9), B^4 = (10, 11). Two $\mathcal{T}$-blocks: B^2, B^4, $\mathcal{O}$-blocks B^1 and $\mathcal{E}$-blocks B^3. We can see these blocks in Fig. 1.

Property 1 For any permutation $\pi \in \Phi(n)$ there are partitions into to blocks (subsequences) such that every of them is:

i. $\mathcal{E}$-block, or
ii. $\mathcal{O}$-block, or
iii. $\mathcal{T}$-block.

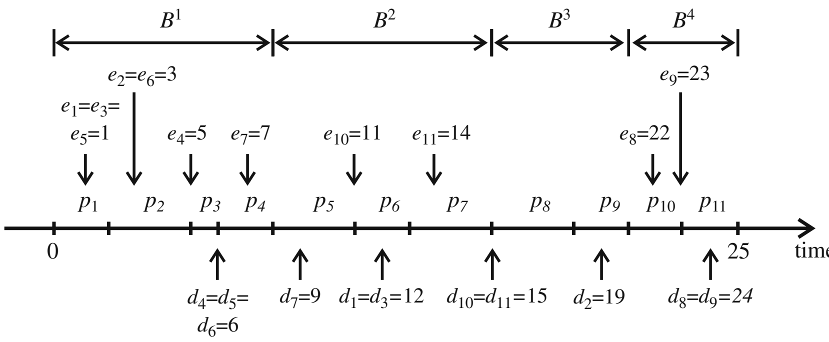

Fig. 1 Blocks of permutation π

Proof Let us assume that a permutation π is partitioned into blocks. From definition of $\mathcal{E}$, $\mathcal{O}$ and $\mathcal{T}$ blocks, if $C_{\pi(1)} < e_{\pi(1)}$, then $\pi(1)$ belongs to the first $\mathcal{E}$-block else if $C_{\pi(1)} > d_{\pi(1)}$, then $\pi(1)$ belongs to the first $\mathcal{T}$-block or (on the contrary) to the first $\mathcal{O}$-block. We sequentially consider jobs $\pi(2), \pi(3), \ldots, \pi(n)$. For the job $\pi(i)$, $2 \le i \le n$, let the previous job $\pi(i-1)$ belong to a block B.

Let us assume that $C_{\pi(i)} < e_{\pi(i)}$. If

1. B is $\mathcal{O}$ or $\mathcal{T}$-block, then $\pi(i)$ is the first job of the next $\mathcal{E}$-block.
2. B is $\mathcal{E}$-block and $B \cup \{\pi(i)\}$ is $\mathcal{E}$-block, then job $\pi(i) \in B$, and on the contrary $\pi(i)$ is the first job of the next $\mathcal{E}$-block.

We can consider the case of $C_{\pi(i)} > d_{\pi(i)}$ and $e_{\pi(i)} \le C_{\pi(i)} \le d_{\pi(i)}$ similarly. □

The definition of blocks and Property 1 follows that after partitioning a permutation:

1) every job belongs to some $\mathcal{E}$ or $\mathcal{O}$ or $\mathcal{T}$ block,
2) blocks are disjoint sets of jobs.

For any block B in a partition $\mathcal{B}$ of permutation $\pi \in \Phi(n)$, let

$$F_B(\pi) = \sum_{i \in B} (u_i E_i + w_i T_i).$$

Therefore, the value of a goal function takes the form of

$$F(\pi) = \sum_{i=1}^{n} (u_i E_i + w_i T_i) = \sum_{B \in \mathcal{B}} F_B(\pi).$$

If B is a $\mathcal{T}$-block then every job that belongs to it is early. Therefore, in the permutation π, an optimal sequence of jobs within B (which is minimizing $F_B(\pi)$) can be obtained using the well-known weighted shortest processing time (*WSPT*) rule proposed by Smith [22]. The *WSPT* rule creates an optimal sequence of jobs in the non-increasing order of the ratios w_j/p_j. Similarly, if B is an $\mathcal{E}$-block, then an optimal sequence of jobs can be obtained using the weighted longest processing time (*WLPT*) rule which creates a sequence of jobs in non-decreasing order of the ratios u_j/p_j.

Partition $\mathcal{B}$ of the permutation π is *ordered* if there are jobs scheduled by the *WSPT* rule in any $\mathcal{T}$-block and jobs scheduled by the *WLPT* rule in any $\mathcal{E}$-block.

Example 2 In permutation $\pi = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)$ from Example 1 the $\mathcal{T}$-block $B^2 = (4, 5, 6, 7)$. Because $\frac{w_5}{p_5} = \frac{5}{3} < \frac{w_6}{p_6} = \frac{4}{2}$, we swap jobs 5 and 6. Similarly, in the $\mathcal{E}$-block $B^3 = (8, 9)$, because $\frac{u_8}{p_8} = \frac{6}{3} > \frac{u_9}{p_9} = \frac{2}{2}$, we swap jobs 8 and 9. We obtain ***ordered*** permutation (1,2,3,4,**6**,**5**,7,**9**,**8**,10,11).

Lemma 1 *If a permutation $\pi \in \Phi(n)$ is ordered, then changing the order of jobs in any block does not generate permutation with less cost of the goal function.*

Proof Let $B = (\pi(a), \pi(a+1), \ldots, \pi(b))$, $1 \le a < b \le n$ be a block in a partition of ordered permutation $\pi \in \Phi(n)$. Let us assume that permutation β was generated from π by changing the order of jobs in block B. Therefore,

$$\beta(i) = \pi(i),\ i = 1, 2, \ldots, a-2, a-1, b+1, b+2, \ldots, n$$

and sets of jobs fulfill the equality

$$\{\beta(j):\ j = a, a+1, \ldots, b\} = \{\pi(j):\ j = a, a+1, \ldots, b\}.$$

We should consider two cases:

1. B is $\mathcal{E}$-block. If jobs from the set $\{\beta(a), \beta(a+1), \ldots, \beta(b-1), \beta(b)\}$ do not fulfill the *WLPT* rule in permutation β, then $F(\beta) \ge F(\pi)$.
2. B is $\mathcal{O}$-block. From the definition of $\mathcal{O}$-block, every job $\beta(j)$, $j = a, a+1, \ldots, b$ is on time in permutation β, so $F(\beta) = F(\pi)$.
3. B is $\mathcal{T}$-block. If jobs from the set $\{\beta(a), \beta(a+1), \ldots, \beta(b-1), \beta(b)\}$ do not fulfill the *WSPT* rule in permutation β, then $F(\beta) \ge F(\pi)$. □

The next theorem is a base of the neighborhood's construction in the local search algorithms.

Theorem 1 *For each ordered permutation $\pi \in \Phi(n)$ if a permutation $\beta \in \Phi(n)$ was obtained from π by any interchange of its elements and*

$$F(\beta) < F(\pi)$$

then in the permutation β at least one job of some block of π was moved before the first or after the last job of this block.

Proof Let $[B^1, B^2, \ldots, B^v]$ be a partition of ordered permutation $\pi \in \Phi(n)$ into blocks. Each block is a subsequence of jobs

$$B^i = (\pi(a^i), \pi(a^i+1), \ldots, \pi(b^i)), \quad i = 1, 2, \ldots, v,$$
$$1 \le a^1 \le b^1 < a^2 \le b^2 <, \ldots, < a^v \le b^v.$$

By

$$Y^i(\pi) = \{\pi(a^i), \pi(a^i+1), \ldots, \pi(b^i)\}$$

we represent the set of jobs from the block B^i.

Let permutation $\beta \in \Phi(n)$ and $F(\beta) < F(\pi)$. Let us assume on the contrary, that in permutation β any job from any block $B^1, B^2, \ldots, B^v$ has not been moved before the first or after the last job of this block. Therefore

$$Y^i(\pi) = Y^i(\beta), \quad i = 1, 2, \ldots, v.$$

Then for $i = 1, 2, \ldots, v$ subsequences $(\pi(a^i), \pi(a^i + 1), \ldots, \pi(b^i - 1), \pi(b^i))$ in permutation π and $(\beta(a^i), \beta(a^i + 1), \ldots, \beta(b^i - 1), \beta(b^i))$ in β are permutations of the same subset of jobs $\{\pi(a^i), \pi(a^i + 1), \ldots, \pi(b^i)\}$. Lemma 1 follows that $F(\beta) \geq F(\pi)$, which contradicts assumption on the contrary. □

Let us notice that Theorem 1 provides the necessary condition to obtain a permutation β from π such that $F(\beta) < F(\pi)$.

Let $\mathcal{B} = [B_1, B_2, \ldots, B_v]$ be an ordered partition of the permutation $\pi \in \Phi(n)$ into blocks. If a job $\pi(j) \in B_i$ $(B_i \in \mathcal{B})$, therefore existing moves, which can improve goal function value, consist in reordering a job $\pi(j)$ before the first or after the last job of this block. Let $\mathcal{M}_j^{bf}$ and $\mathcal{M}_j^{af}$ be sets of such moves (obviously $\mathcal{M}_1^{bf} = \mathcal{M}_v^{af} = \oslash$). Therefore, the neighborhood of the permutation $\pi \in \Phi(n)$ has the form of

$$\mathcal{M}(\pi) = \bigcup_{j=1}^{n} \mathcal{M}_j^{bf} \cup \bigcup_{j=1}^{n} \mathcal{M}_j^{af}. \tag{1}$$

A move $\hat{m}$ is a *representative* of moves from the set $M \subseteq \mathcal{M}(\pi)$, if

$$\forall\, m \in M, \quad F(m(\pi)) \geq F(\hat{m}(\pi)).$$

Computational experiments show that the neighborhood defined in (1) has a half smaller size than the neighborhood of all the insert and swap moves.

Example 3 In the ordered permutation (1,2,3,4,**6**,**5**,7,**9**,**8**,10,11) from the Example 2 there are blocks: $B^1 = (1, 2, 3)$, $B^2 = (4, 6, 5, 7)$, $B^3 = (8, 9)$ and $B^4 = (10, 11)$. For element 6 from the block B^2 (taking into consideration, that $i_3^6 = s_3^6$ and $i_8^6 = s_8^6$) there are sets of moves $\mathcal{M}_6^{bf} = \{i_1^6, i_2^6, i_3^6\} \cup \{s_1^6, s_2^6\}$ and $M_6^{af} = \{i_8^6, i_9^6, i_{10}^6, i_{11}^6\} \cup \{s_9^6, s_{10}^6, s_{11}^6\}$.

Similarly, we can determine sets of moves for jobs from other blocks. Set $\mathcal{M}(\pi)$ has $\sum_{j=1}^{11} \left(\left|\mathcal{M}_j^{bf}\right| + \left|\mathcal{M}_j^{af}\right| \right) = 118$ moves. For this example, the number of all the possible (swap and insert) moves is 10*(10-1)+10(10-1)/2=165.

1.2 Properties of insert moves

In this section we assume that permutation $\pi \in \Phi(n)$ is ordered and a move of a job consists in making an i-move.

Theorem 2 *If for two jobs $\pi(l), \pi(k)$ $(1 \leq l < k \leq n)$*

$$C_{\pi(l)} + p_{\pi(k)} < e_{\pi(k)} \text{ and } C_{\pi(l)} + p_{\pi(k)} < e_{\pi(l)}, \tag{2}$$

then for the pair of moves i_l^k, i_{l+1}^k it occurs that: if $u_{\pi(l)}/p_{\pi(l)} \geq u_{\pi(k)}/p_{\pi(k)}$ then

$$F(\pi_{l+1}^k) \geq F(\pi_l^k) \text{ else } F(\pi_{l+1}^k) \leq F(\pi_l^k).$$

Proof We consider two moves i_l^k and i_{l+1}^k. It occurs:

$$\pi_l^k(j) = \pi_{l+1}^k(j) \text{ for } j = 1, 2, \ldots, l-1, l+2, \ldots, n \text{ and}$$

$$\pi_l^k(l) = \pi_{l+1}^k(l+1) = \pi(k),\ \pi_l^k(l+1) = \pi_{l+1}^k(l) = \pi(l),$$

where permutations π_l^k and π_{l+1}^k are generated (adequately) by moves i_l^k and i_{l+1}^k.

From the definition of permutations π_l^k, π_{l+1}^k and from the formula of the cost function:

$$\begin{aligned}
F(\pi_{l+1}^k) - F(\pi_l^k) &= \sum_{j=1}^{n} f_{\pi_{l+1}^k(j)}\big(C_{\pi_{l+1}^k(j)}\big) - \sum_{j=1}^{n} f_{\pi_l^k(j)}\big(C_{\pi_l^k(j)}\big) \\
&= \sum_{j=1}^{l-1} f_{\pi_{l+1}^k(j)}\big(C_{\pi_{l+1}^k(j)}\big) + f_{\pi_{l+1}^k(l)}\big(C_{\pi_{l+1}^k(l)}\big) \\
&\quad + f_{\pi_{l+1}^k(l+1)}\big(C_{\pi_{l+1}^k(l+1)}\big) \\
&\quad + \sum_{j=l+2}^{n} f_{\pi_{l+1}^k(j)}\big(C_{\pi_{l+1}^k(j)}\big) \\
&\quad - \Bigg(\sum_{j=1}^{l-l} f_{\pi_l^k(j)}\big(C_{\pi_l^k(j)}\big) + f_{\pi_l^k(l)}\big(C_{\pi_l^k(l)}\big) \\
&\quad + f_{\pi_l^k(l+1)}\big(C_{\pi_l^k(l+1)}\big) \\
&\quad + \sum_{j=l+2}^{n} f_{\pi_l^k(j)}\big(C_{\pi_l^k(j)}\big) \Bigg).
\end{aligned}$$

Because

$$C_{\pi_l^k(j)} = C_{\pi_{l+1}^k(j)} = C_{\pi(j)}, \quad j = 1, 2, \ldots, l-1,$$

$$l+2, \ldots, n, \quad \text{and } C_{\pi_l^k(l+1)} = C_{\pi_{l+1}^k(l+1)},$$

$$C_{\pi_l^k(l)} = A - p_{\pi(l)}, C_{\pi_{l+1}^k(l)} = A - p_{\pi(k)},$$

where $A = C_{\pi(l)} + p_{\pi(k)}$, therefore

$$\begin{aligned} F(\pi^k_{l+1}) - F(\pi^k_l) &= f_{\pi^k_{l+1}(l)}(C_{\pi^k_{l+1}(l)}) + f_{\pi^k_{l+1}(l+1)}(C_{\pi^k_{l+1}(l+1)}) \\ &\quad - f_{\pi^k_l(l)}(C_{\pi^k_l(l)}) - f_{\pi^k_l(l+1)}(C_{\pi^k_l(l+1)}) \\ &= f_{\pi(l)}(A - p_{\pi(k)}) + f_{\pi(k)}(A) \\ &\quad - f_{\pi(k)}(A - p_{\pi(l)}) - f_{\pi(l)}(A). \end{aligned}$$

From the assumption (2) it follows that jobs $\pi(k)$ and $\pi(l)$ are early, so

$$\begin{aligned} F(\pi^k_{l+1}) - F(\pi^k_l) &= u_{\pi(l)} \cdot (e_{\pi(l)} - A + p_{\pi(k)}) \\ &\quad + u_{\pi(k)} \cdot (e_{\pi(k)} - A) \\ &\quad - u_{\pi(k)} \cdot (e_{\pi(k)} - A + p_{\pi(l)}) \\ &\quad - u_{\pi(l)} \cdot (e_{\pi(l)} - A) \\ &= u_{\pi(l)} p_{\pi(k)} - u_{\pi(k)} p_{\pi(l)}. \end{aligned}$$

If $u_{\pi(l)} p_{\pi(k)} - u_{\pi(k)} p_{\pi(l)} \geq 0$ then

$$F(\pi^k_{l+1}) - F(\pi^k_l) \geq 0 \quad \text{else} \quad F(\pi^k_{l+1}) - F(\pi^k_l) \leq 0,$$

finishing the proof of the theorem. □

Now we will prove a similar theorem for the moves executed after the last job of the block.

Theorem 3 *If for two jobs $\pi(l), \pi(k) \quad (1 \leq k < l \leq n)$*

$$C_{\pi(l)} - p_{\pi(k)} > d_{\pi(l)} \text{ and } C_{\pi(l)} - p_{\pi(l)} > d_{\pi(k)}, \tag{3}$$

then for the pair of moves i^k_{l-1}, i^k_l it occurs that: if $w_{\pi(l)}/p_{\pi(l)} \geq w_{\pi(k)}/p_{\pi(k)}$ then

$$F(\pi^k_l) \geq F(\pi^k_{l-1}) \quad \textit{else} \quad F(\pi^k_l) \leq F(\pi^k_{l-1})$$

Proof Similarly, as in the proof of Theorem 2 from the definition of permutations π^k_l, π^k_{l-1} and from the formula of the cost function:

$$\begin{aligned} F(\pi^k_l) - F(\pi^k_{l-1}) &= f_{\pi(l)}(H - p_{\pi(k)}) + f_{\pi(k)}(H) \\ &\quad - f_{\pi(k)}(H - p_{\pi(l)}) - f_{\pi(l)}(H), \end{aligned}$$

where $H = C_{\pi(l)}$.

From the assumption (3) if follows that jobs $\pi(k)$ and $\pi(l)$ are tardy, so

$$\begin{aligned} F(\pi^k_l) - F(\pi^k_{l-1}) &= w_{\pi(l)} \cdot (d_{\pi(l)} - H + p_{\pi(k)}) \\ &\quad + w_{\pi(k)} \cdot (d_{\pi(k)} - H) - w_{\pi(k)} \\ &\quad \cdot (d_{\pi(k)} - H + p_{\pi(l)}) \\ &\quad - w_{\pi(l)} \cdot (d_{\pi(l)} - H) \\ &= w_{\pi(l)} p_{\pi(k)} - w_{\pi(k)} p_{\pi(l)}. \end{aligned}$$

If $w_{\pi(l)} p_{\pi(k)} - w_{\pi(k)} p_{\pi(l)} \geq 0$ then

$$F(\pi^k_l) - F(\pi^k_{l-1}) \geq 0 \quad \text{else} \quad F(\pi^k_l) - F(\pi^k_{l-1}) \leq 0,$$

finishing the proof of the theorem. □

Property 2 Let $B^i = (\pi(a^i), \pi(a^i+1), \ \ldots \ , \pi(b^i - 1), \pi(b^i))$, be an $\mathcal{E}$-block of partition of the ordered permutation π and let the job $\pi(k) \notin B^i$ $(b^i < k \leq n)$. If

$C_{\pi(b^i)} - p_{\pi(b^i)} + p_{\pi(k)} < e_{\pi(k)}$ and

$\forall\, \pi(j) \in B^i, \ C_{\pi(j)} + p_{\pi(k)} < e_{\pi(j)}$, then

$F(\pi^k_l) \geq F(\pi^k_h)$, $l = a^i, a^i+1, \ldots, h-1, h+1, \ldots, b^i$,

where

$$h = min \left\{ x : \frac{u_{\pi(x)}}{p_{\pi(x)}} \geq \frac{u_{\pi(k)}}{p_{\pi(k)}}, \ a^i \leq x \leq b^i \right\}.$$

This property follows from Theorem 2 and from the fact that jobs in the $\mathcal{E}$-block are scheduled by the *WLPT* rule.

From the Property 2 follows that i^k_h is a representative of moves from the set $\{i^k_{a^i}, i^k_{a^i+1}, \ldots, i^k_{b^i}\}$. Moves $i^k_{a^i}, i^k_{a^i+1}, \ldots, i^k_{h-1}, i^k_{h+1}, \ldots, i^k_{b^i}$ can be omitted while determining an optimal move (permutation with the minimal value of the goal function) from the neighborhood (1), so

$$\mathcal{M}(\pi) \leftarrow \mathcal{M}(\pi) \backslash \{i^k_{a^i}, i^k_{a^i+1}, \ldots, i^k_{h-1}, i^k_{h+1}, \ldots, i^k_{b^i}\}. \tag{4}$$

Property 3 Let $B^i = (\pi(a^i), \pi(a^i+1), \ \ldots \ , \pi(b^i - 1), \pi(b^i))$ be a $\mathcal{T}$-block of partition of the ordered permutation π and let the job $\pi(k) \neq B^i$ $(1 \leq k < a^i)$. If

$C_{\pi(a^i)} > d_{\pi(k)}$ and $\forall\, \pi(j) \in B^i, \ C_{\pi(j)} - p_{\pi(k)} > d_{\pi(j)}$,

then

$F(\pi^k_l) \geq F(\pi^k_t)$, $l = a^i, a^i+1, \ldots, t-1, t+1, \ldots, b^i$,

where

$$t = max \left\{ x : \frac{w_{\pi(x)}}{p_{\pi(x)}} \geq \frac{w_{\pi(k)}}{p_{\pi(k)}}, \ a^i \leq x \leq b^i \right\}.$$

The following property follows from Theorem 3 and from the fact that jobs in the $\mathcal{T}$-block are scheduled by the *WSPT* rule.
The move i^k_t is a representative of moves from the set $\{i^k_{a^i}, i^k_{a^i+1}, \dots, i^k_{b^i}\}$. Moves $i^k_{a^i}, i^k_{a^i+1}, \dots, i^k_{t-1}, i^k_{t+1}, \dots, i^k_{b^i}$ can be omitted during determining an optimal move (permutation with the minimal value of the goal function) from the neighborhood (1), therefore

$$\mathcal{M}(\pi) \leftarrow \mathcal{M}(\pi) \setminus \{i^k_{a^i}, i^k_{a^i+1}, \dots, i^k_{t-1}, i^k_{t+1}, \dots, i^k_{b^i}\}. \quad (5)$$

Property 4 Let $B^i = (\pi(a^i), \pi(a^i+1), \ \dots \ , \pi(b^i-1), \pi(b^i))$ be an $\mathcal{O}$-block of a partition of the permutation π and let the job $\pi(k) \neq B^i$. If

A. $b^i < k \le n$ and $C_{\pi(b^i)} - p_{\pi(b^i)} + p_{\pi(k)} \le d_{\pi(k)}$ then

$$F(\pi^k_j) \ge F(\pi^k_{b^i}), \ \ j = a^i, a^i+1, \dots, , b^i - 1,$$

and

$$\mathcal{M}(\pi) \leftarrow \mathcal{M}(\pi) \setminus \{i^k_{a^i}, i^k_{a^i+1}, \dots, i^k_{b^i-2}, i^k_{b^i-1}\}. \quad (6)$$

B. $1 \le k < a^i$ and $C_{\pi(a^i)} - p_{\pi(a^i)} + p_{\pi(k)} \ge e_{\pi(k)}$ then

$$F(\pi^k_j) \ge F(\pi^k_{a^i}), \ \ j = a^i+1, a^i+2, \dots, , b^i,$$

and

$$\mathcal{M}(\pi) \leftarrow \mathcal{M}(\pi) \setminus \{i^k_{a^i+1}, i^k_{a^i+2}, \dots, i^k_{b^i-1}, i^k_{b^i}\}. \quad (7)$$

This property can be easily proved using definition of blocks and goal function.

We have also proved other properties of moves; therefore they allow us to eliminate only a few elements from the neighborhood.

Application of block elimination properties (Theorem 1) and Properties 2, 3, and 4 allow us to remove from each neighborhood generated by insert moves over 50 % of its elements.

Similar theorems and properties, as presented in this section, can be proved for the swap moves, too.

2 Tabu search algorithm

The tabu search method (TS - proposed by Glover [6] and [7]) is one of the most effective methods using local search techniques to find near-optimal solutions to many combinatorial intractable optimization problems. The elementary step of the method performs, for a given solution x^i, a search through the *neighborhood* $\mathcal{N}(x^i)$ of x^i. The neighborhood is defined by *move*. A move transforms a solution into another solution. The aim of the elementary search is to find in $\mathcal{N}(x^i)$ a solution x^{i+1} with the lowest cost functions. Then the search is repeated from the best found, as a new starting solution, and the process is continued. In order to avoid cycling, becoming trapped to a local optimum, and more general to conduct the search in "good regions" of the solution space, a memory of the search history is introduced. Among many classes of the memory introduced for tabu search, the most frequently used is the short-term memory called the tabu list. This list recorded, for a chosen span of time, solutions or selected attributes of these solutions (or moves). The search is stopped when a given number of iterations or current neighborhood is empty.

In this section we present a tabu search algorithm to solve *TWET-no-idle* problem. In the algorithm's construction there are properties of moves and multi-moves.

2.1 Neighborhood

For the permutation $\pi \in \Phi(n)$ using the definition of the set of insert moves (1) and Property 1-3, we define *sets of representatives of moves* $\mathcal{M}(\pi)$.

The *neighborhood* of the π is a set of permutations

$$\mathcal{N}(\pi) = \{m(\pi): \ \ m \in \mathcal{M}(\pi)\}.$$

2.2 Tabu list

To prevent from arising cycle too quickly (returning to the same permutation after some small number of iterations of the algorithm), some attributes of each move are saved on a so-called tabu list (list of the prohibited moves). This list is served as a FIFO queue, see [9; 10]. Making a move $i^r_j \in \mathcal{M}(\pi)$, (that means generating permutation π^r_j from $\pi \in \Phi(n)$) we save attributes of this move, triple $(\pi(r), j, F(\pi^r_j))$, on the tabu list. Let us assume that we consider a move $i^k_l \in \mathcal{M}(\beta)$ that generates permutation β^k_l. If there is a triple (r, j, Ψ) such that $\beta(k) = r, \ l = j$ on the tabu list, and $F(\beta^k_l) \ge \Psi$, then such a move is eliminated (removed) from the set $\mathcal{M}(\beta)$.

The dynamic length *lTS* of tabu list *TSL* is a cyclic function defined by the expression:

$$lTS(iter) = \begin{cases} low & if \ \ S(k) < iter \le S(k) + h, \\ low + \alpha & if \ \ S(k) + h < iter \le S(k+1), \end{cases}$$

where: *iter* is the number of iteration of the algorithm, $k = 1, 2, \ \dots$ is the number of the cycle. Integer number $\alpha > 0$, $S(k) = (k-1)(h+H)$, here $S(0) = 0$. *Low* is the standard length of the *TSL* list (by h iterations of the algorithm) and H is the width of the pick equal $low + \alpha$.

If *lTS* decreases, then a suitable number of the oldest elements of tabu list *TSL* is deleted and the search process is continued. All parameters of length of the tabu list are empirical, based on preliminary experiments. Changing the tabu list's length causes diversification of the search process.

2.3 Multimoves

Any move $i^r_j \in \mathcal{M}(\pi) \quad (1 \le r,\ j \le n)$ generates permutation $i^r_j(\pi) \in \Phi(n)$ from $\pi \in \Phi(n)$. We consider two moves $i^k_l, i^r_j \in \mathcal{M}(\pi)$. Let δ be a permutation created by executing a move i^k_l on the permutation $i^r_j(\pi)$, that is $\delta = i^k_l(i^r_j(\pi))$. Permutation δ is generated by a composition of two moves i^r_j and i^k_l. Of course, $\delta \in \Phi(n)$. We call *multimove* superposition of the sequence s of moves $i^{r_1}_{j_1},\ i^{r_2}_{j_2},\ \dots,\ i^{r_s}_{j_s} \in \mathcal{M}(\pi)$ and we represent it by $R(i^{r_1}_{j_1},\ i^{r_2}_{j_2},\ \dots,\ i^{r_s}_{j_s})$. It generates permutation $R(i^{r_1}_{j_1},\ i^{r_2}_{j_2},\ \dots,\ i^{r_s}_{j_s})(\pi) = i^{r_1}_{j_1}(i^{r_2}_{j_2}(\ \dots\ (i^{r_s}_{j_s}(\pi))\ \dots\ ))$.

For each move $i^r_j \in \mathcal{M}(\pi)$ $(\pi \in \Phi(n))$, let

$$\Delta\left(i^r_j\right) = F(\pi) - F\left(i^r_j(\pi)\right).$$

If $\Delta(i^r_j) > 0$, then move i^r_j is improves.

Moves $i^r_j,\ i^k_l \in \mathcal{M}(\pi)$ we call *independent*, if

$$\max\{r,\ j\} < \min\{l,\ k\} \text{ or } \min\{r,\ j\} > \max\{l,\ k\}.$$

Theorem 4 *Let $\delta = R(m^{r_1}_{j_1},\ m^{r_2}_{j_2},\ \dots,\ m^{r_h}_{j_h})(\pi)$ be a permutation generated by multimove $R(m^{r_1}_{j_1},\ m^{r_2}_{j_2},\ \dots,\ m^{r_h}_{j_h})$, $h \ge 1$. If moves from the set $\{m^{r_1}_{j_1},\ m^{r_2}_{j_2},\ \dots,\ m^{r_h}_{j_h}\}$ are pair-independent (independent for each pair of them), then*

$$F(\delta) - F(\pi) = \sum_{v=1}^{h} \Delta\left(m^{r_v}_{j_v}\right).$$

Proof Let m^r_j be one of moves of the multimove $R(m^{r_1}_{j_1},\ m^{r_2}_{j_2},\ \dots,\ m^{r_h}_{j_h})$. Let us assume that $j \ge r$. By the $\sigma(m^r_j) = (\pi(r), \pi(r+1),\ \dots, \pi(j))$ we represent some subpermutation of permutation π. Move m^r_j changes sequence of elements only in subpermutation $\sigma(m^r_j)$. If moves $m^r_j,\ m^k_l$ are independent then subpermutations $\sigma(m^r_j)$ and $\sigma(m^k_l)$ are disjoint and $\pi^r_j(t) = \pi^k_l(t)$ $(t = 1, 2,\ \dots, n, \quad t \notin \sigma(m^r_j) \cup \sigma(m^l_k))$. So for the sequence of moves $(m^{r_1}_{j_1},\ m^{r_2}_{j_2},\ \dots,\ m^{r_h}_{j_h})$, which are pair-independent, permutation π is in form $\pi = (\dots, \sigma(m^{r_1}_{j_1}),\ \dots, \sigma(m^{r_2}_{j_2}),\ \dots,\ \sigma(m^{r_h}_{j_h}),\ \dots)$. Because $\Delta(m^{r_s}_{j_s}) = F(\pi) - F(m^{r_s}_{j_s}(\pi)) \quad (1 \le s \le h)$, therefore

$$F\left(R\left(m^{r_1}_{j_1}, m^{r_2}_{j_2},\ \dots\ m^{r_h}_{j_h}\right)(\pi)\right) - F(\pi) = \sum_{v=1}^{h} \Delta\left(m^{r_v}_{j_v}\right).$$

If we assume that $j < r$, for move m^r_j then the proof is similar. □

Intensification of calculations.

To achieve local minimum quickly, we make *improving multimove*, which is a composition of pair-independent improving moves from the set $\mathcal{M}(\pi)$.

Algorithm of determining the improving multimove R.

```
Step 1: Sort elements of the set $\mathcal{M}(\pi)$ so that
        $\Delta(m^{r_1}_{j_1}) \ge \Delta(m^{r_2}_{j_2}) \ge,\ \dots, \ge \Delta(m^{r_t}_{j_t})$,
        where $|\mathcal{M}(\pi)| = t$;
        if $\Delta(m^{r_1}_{j_1}) \le 0$ then $h \leftarrow 0$ and EXIT;
Step 2 : $R \leftarrow m^{r_1}_{j_1}$;   $h \leftarrow 1$;
        for each $k = 2, 3,\ \dots, t$ do
        if move $m^{r_k}_{j_k}$ improves ($\Delta(m^{r_k}_{j_k}) > 0$) and
          if independent with each move of the
          sequence $R$ then
          $h \leftarrow h + 1$ and $R \leftarrow R \cup m^{r_k}_{j_k}$;
```

The foregoing algorithm is based on greedy method and has computational complexity O($t$ln$t$), where t is $O(n^2)$. Executing the improving multimove $R(m^{i_1}_{j_1}, m^{i_2}_{j_2}, ..., m^{i_h}_{j_h})$ which intensifies the calculations we save attributes of all moves $m^{i_1}_{j_1},\ m^{i_2}_{j_2},\ \dots,\ m^{i_h}_{j_h}$ on the tabu list.

Strong diversification of calculations.

The aim of strong diversification is to move the area of search process from the neighborhood of local minimum to other area of the set of problem's solutions. It consists of executing a *dispersing multimove*, which is a composition of worsening moves from the set $\mathcal{M}(\pi)$. We call a move worsening if it makes the cost function worse.

Let *Length*Φ be a parameter, the number of moves in the dispersing multimove.

Algorithm of determining the dispersing multimove Φ.

```
Step 0: Let $t = \left|\{m^k_l \in \mathcal{M}(\pi) :\ \ \Delta(m^k_l) < 0\}\right|$;
        if $t = 0$ then $h \leftarrow 0$ and EXIT;
Step 1: Sort elements from the set $\mathcal{M}(\pi)$ such that
        $\Delta(m^{r_1}_{j_1}) \le \Delta(m^{r_2}_{j_2}) \le,\ \dots, \le \Delta(m^{r_t}_{j_t})$;
Step 2 : $\Phi \leftarrow m^{r_1}_{j_1}$;   $h \leftarrow 1$;
        for each $l = 2, 3,\ \dots, t$ do
          $h \leftarrow h + 1$ and $\Phi \leftarrow \Phi \cup m^{i_l}_{j_l}$;
          if $h = \min\{t,\ Length\Phi\}$ then EXIT.
```

Computational complexity of this algorithm is $O(t\ln t)$. Executing the dispersing multimove $\Phi(m^{r_1}_{j_1}, m^{r_2}_{j_2},\ \dots, m^{r_h}_{j_h})$ we remove all elements from the tabu list and next we save attributes of moves $m^{r_1}_{j_1},\ m^{r_2}_{j_2},\ \dots,\ m^{r_h}_{j_h}$ on the tabu list.

In the tabu search algorithm specification *TSL* represents tabu list, the variable *Maxiter* – number of iterations of the algorithm (stop criterion) and *Maxbp* – number of iterations without improving solutions, after which diversification is executed ($Length\Phi$ - number of moves in dispersing multimove), $\pi*$ – the best known solution.

```
Algorithm TSE/Tni (Tabu Search Algorithm)
  Input: initial solution (permutation) $\pi^0$;
  Output: job's permutation $\pi^*$;
    Step 0:{Initialization}
        Set $\pi^* := \pi^0$;   $\pi := \pi^0$; $F^* := F(\pi^*)$;
        $LTS := \emptyset$; iter:=0; itbp:=0;
    Step 1:{Searching}
        Partition permutation $\pi$ into blocks and
        ordered elements in $\mathcal{E}$ and $\mathcal{T}$ blocks.
        Create a set moves $\mathcal{M}(\pi)$ using definition (1)
        and Property 1-3.
        if $\mathcal{M}(\pi) = \emptyset$ then EXIT;
    Step 2:{Selection}
        if $\mathcal{M}(\pi) \neq \emptyset$ then {intensification of
        calculations}
          execute improving multimove
          $R(m_{j_1}^{r_1}, m_{j_2}^{r_2}, \dots, m_{j_h}^{r_h})$ and
          [$\pi \leftarrow R(m_{j_1}^{r_1}, m_{j_2}^{r_2}, \dots, m_{j_h}^{r_h})(\pi)$; itbp:=0]
        else
          if $itbp = Maxbp$ then { diversification
          of calculations }
            execute dispersing multimove
            $\Phi(m_{j_1}^{r_1}, m_{j_2}^{r_2}, \dots, m_{j_h}^{r_h})$ and
            [$\pi \leftarrow \Phi(m_{j_1}^{r_1}, m_{j_2}^{r_2}, \dots, m_{j_h}^{r_h})(\pi)$; itbp:=0]
          else {non-improving move}
            select a move $m_k^l$ with
            $\Delta(m_l^k) = \max\{\Delta(m_j^r) : \ m_j^r \in \mathcal{M}(\pi)\}$
            and
            [$\pi \leftarrow m_l^k(\pi)$; itbp:=itbp+1];
        Save attributes of executed moves
        on the tabu list TSL;
        if $F^* > F(\pi)$ then $\pi^* := \pi$ and $F^* := F(\pi)$;
        iter:=iter+1;
    Step 3:{Stop criteria}
        if iter > Maxiter then EXIT
        else goto Step 1;
```

Computational complexity of the algorithm is $O(n^2 \ln n \cdot Maxiter)$.

We use multimoves for intensification and diversification of calculations in the *TSE/Tni* algorithm. Each of these multimoves is a superposition of a single swap and insert moves (are no more than $1.5n(n-1)$). Algorithms of determining a single multimove have computational complexity $O(n^2 \ln n)$.

3 Computational experiments

The algorithms have been tested on several commonly used instances of various size and level of difficulty:

a) 375 benchmark instances of three different sizes with 40, 50, and 100 jobs from the OR-Library [20].
b) test problems were generated as follows: for each job i, an integer processing time p_i was generated from the uniform distribution [1, 100] and integer weights u_i and w_i were generated from the uniform distribution [1, 10]. Let $P = \sum_{i=1}^{n} p_i$. Distributions of earliness e_i and deadline d_i depend on P and two additional parameters L and R which take on values from 0.2 to 1.0 in increments of 0.2. An integer deadline d_i was generated from the uniform distribution $[P(L-R/2), P(L+R/2)]$. Earliness e_i was generated as an integer from the uniform distribution $[0, d_i]$. Five problems were generated for each of the 25 pairs of values of R and L, yielding 125 problems for each value of n=200, 300, 400, 500.

These problems are solved by a PC with a Pentium IV 1.2GHz processor.

Initial permutation The quality of solutions calculated by TS algorithm depends also on the starting point. Below we present the constructive heuristic algorithm which calculates these solutions. It is based on the idea of NEH algorithm [19] and creates n elements' permutation $\pi \in \Phi(n)$.

For the job $i \in \mathcal{J}$ and the number $x \geq 0$, let $f_i(x) = \max\{0, e_i - x\} + \max\{0, x - d_i\}$. By $P = \sum_{i=1}^{n} p_i$ we define the time of all performed jobs.

```
Algorithm CAE/T {Constructive Algorithm}
  Let $\alpha_i = max\{p_i/u_i, p_i/w_i\}$;
  Enumerate jobs such as $\alpha_1 \geq \alpha_2 \geq, \dots, \geq \alpha_n$;
  $l := 1$; $k := n$
  if $\alpha_1 = p_1/u_1$ then $\pi(1) := 1$ and $l$:=2
  else $\pi(n) := 1$ and $k := k - 1$;
  for $i := 2$ to $n$ do
    if $\alpha_i = p_i/u_i$ then
    begin
      Insert a job $i$ on one of positions $1, 2, \dots, l$
      so that the sum
      $\sum_{j=1}^{l} f_{\pi(j)}(C_{\pi(j)})$ was minimal,
      where $C_{\pi(j)} = \sum_{s=1}^{j} p_{\pi(s)}$;
      $l := l + 1$
    end {if}
    else
    begin
```

```
        Insert a job $i$ on one of positions $k, k+1, ..., n$
        so that the sum
        $\sum_{j=k}^{n} f_{\pi(j)}(C_{\pi(j)})$ was minimal,
        where $C_{\pi(j)} = P - \sum_{s=j+1}^{n} p_{\pi(s)}$;
        $k := k - 1$
    end {else}
```

The *CAE/T* algorithm requires $O(n^2)$ time.

3.1 Parameter selection

To fix values of parameters of the algorithm we have conducted experiments on randomly chosen instances (five per each number of jobs n=40, 50, 100). After analysis of the received results we have assumed:

1. dynamic length of tabu list ($lTS(iter)$): $h = \lceil n/4 \rceil$, $H = \lceil n/10 \rceil$, low=$\lceil \sqrt{n} \rceil$, α=$\lceil \sqrt{n/4} \rceil$
2. number of moves in the dispersing multimove $Length\Phi = \lceil \sqrt{n/2} \rceil$.
3. number of iterations without improvement of the best founded solution after which there is executed a dispersing multimove, *Maxbp* = 3.

3.2 Comparative results

In Table 2 we present the results obtained for the test problems of class (a). We compare the results of the algorithms *CAE/T* and *TSE/Tni* with the results of the parallel genetic algorithm described in the study [3] (placed on the OR-Library page [20]). For each test instance we compute the values F^A—the makespan found by the algorithm $A \in \{CAE/T, TSE/Tni\}$, and $100 \cdot (F^A - UB)/UB$—the relative percentage difference between makespan F^A and the best upper bound UB (from OR Library [20]).

For each n (group of test instances), we collected the following values:

δ_{aprd} –	the value (the average for 125 instances) of the relative percentage difference between the cost function F^A (found by algorithm A) and the best-known upper bound,
δ_{mrpd} –	the maximum relative percentage deviation from the upper bound value.

Table 2 The comparison of the results of the constructive algorithm (*CAE/T*) and tabu search (*TSE/Tni*, *Maxiter* =$2n^2$) with the results placed on the OR- Library page [20]

n	Algorithm *CAE/T*		Algorithm *TSE/Tni*	
	δ_{aprd}	δ_{mrpd}	δ_{aprd}	δ_{mrpd}
40	3.21	9.23	−4.37	1.36
50	3.99	26.57	−7.16	2.13
100	8.64	35.14	−12.82	−2.81
all	**5.28**	**24.16**	**−8.12**	**0.23**

Table 3 The comparison of the results of the tabu search algorithm *TSE/Tni* (*Maxiter* =$2n^2$) with the results of the algorithm (*CAE/T*)

	Number of jobs n			
	200	300	400	500
δ_{mrpd}	−16.27	−21.35	−23.64	−39.51

On the basis of the results placed in Table 2 we can state that the average relative error for the constructive algorithm *CAE/T* amounts to 5.28%, whereas the maximum error amounts to 24.16%. In spite of such large errors, the constructive algorithm can be used successfully to determine the starting solutions for the local search algorithms. The algorithm presented in Chap. 3 *TSE/Tni* is much better than the parallel genetic algorithm described in study [3]. The average relative error (the reference results' improvement) amounts to −8.12%. Especially the improvement is considerable for data with the highest size ($n = 100$) and amounts to −12.82%.

In Table 3 we compare the results of the constructive algorithm *CAE/T* with the results of the algorithm *TSE/Tni*, based on the TS method for data from class (b).

The relative average (δ_{mrpd}) improvement of the results of the constructive algorithm is very considerable and amounts to 25.2%.

In Table 4, the results of the algorithm *TSE/Tni* are placed, for several numbers of iterations. The results presented in Table 4 show very fast convergence of the *TSE/Tni* algorithm.

The average (t_{aprd}) and maximum (t_{mrpd}) running time of the proposed algorithm is listed in the Table 5. The average running time is relatively short, particulary when the number of jobs is $n \leq 500$.

As we can see, the method proposed here increases the quality of the solutions, particulary after the application of the block properties proposed in Section 1.1 and it significantly shortens the time of computations.

Table 4 The comparison of the computational results of *TSE/Tni* algorithm with n, $2n$ and n^2 iterations

n	*Maxiter* =n		*Maxiter* =$2n$		*Maxiter* =n^2	
	δ_{aprd}	δ_{mrpd}	δ_{aprd}	δ_{mrpd}	δ_{aprd}	δ_{mrpd}
40	−1.18	9.84	−1.26	9.84	−4.12	1.36
50	−2.03	13.98	−2.39	10.42	−6.85	2.04
100	−4.28	18.72	−6.12	16.87	−12.32	0.21
all	**−2.50**	**14.18**	**−3.26**	**12.38**	**−7.76**	**1.20**

Table 5 Running time (seconds) of algorithm *TSE/Tni* for $2n^2$ iterations

	Number of jobs n						
	40	50	100	200	300	400	500
t_{aprd}	0.007	0.01	0.04	0.10	0.57	2.05	5.36
t_{mrpd}	0.009	0.02	0.06	0.13	0.63	2.86	6.24

4 Conclusions

We have discussed a new approach to neighborhood search for the single-machine earliness-tardiness scheduling problem based on new elimination criteria—block properties. These properties allow us to propose a new, very fast algorithm based on the tabu search approach. Moreover, we propose a tabu list with dynamic length that is changed cyclically, as the current iteration number of TS increases, using the "pick" in order to carry the search to another area of the solution space. Finally, some perturbations associated with block properties are periodically applied. Computational experiments are presented. The results show that the algorithm proposed provides much better results than in recent modern approaches. The results obtained encourage us to extend the ideas proposed to other hard problems of sequencing, for example, to the E/T problem.

References

1. Baker KR, Scudder CD (1990) Sequencing with earliness and tardiness penalties: a review. Oper Res 38:22–36
2. Bank J, Werner F (2001) Heuristic algorithm for unrelated parallel machine scheduling with a common due date, release dates, and linear earliness and tardiness penalties. Math Comput Model 33:363–383
3. Bożejko W, Wodecki M (2005) Task realization's optimization with earliness and tardiness penalties in distributed computation systems. AWIC 2005, LNAI 3528:69–75
4. Bożejko W, Grabowski J, Wodecki M (2006) Block approach-tabu search algorithm for single-machine total weighted tardiness problem. Comput Ind Eng 50:1–14
5. Feldmann M, Biskup D (2003) Single-machine scheduling for minimizing earliness and tardiness penalties by meta-heuristic approaches. Comput Ind Eng 44:307–323
6. Glover F (1989) Tabu search. Part I. ORSA J Comput 1: 190–206
7. Glover F (1990) Tabu search. Part II. ORSA J Comput 2: 4–32
8. Gordon V, Proth JP, Chu C (2002) A survey of the state-of-art of common due date assignment and scheduling research. Eur J Oper Res 139:1–25
9. Grabowski J, Wodecki M (2004) A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion. Comput Oper Res 31:1891–1909
10. Grabowski J, Wodecki M (2005) A very fast tabu search algorithm for the job shop problem. In: Rego C, Alidaee B (ed) Adaptive memory and evolution; tabu search and scatter search. Dordrecht, Kluwer Academic Publishers
11. Hendel Y, Soud F (2006) Efficient neighborhood search for the one-machine earliness-tardiness scheduling problem. Eur J Oper Res 173:108–119
12. Hoogeveen JA, Van de Velde LS (1996) A branch-and-bound algorithm for single-machine earliness-tardiness scheduling with idle time. INFORMS J Comput 8:402–412
13. Lawler EL (1977) A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness. Ann Discrete Math 1:331–342
14. Lee CY, Choi JY (1995) A generic algorithm for job sequencing problem with distinct due dates and general early-tardy penalty weights. Comput Oper Res 22:857–869
15. Kanett JJ (1981) Minimizing the average deviation of job completion times about a common due date. Nav Res Logist 28:643–651
16. Korman K (1994) A pressing matter. Video Mag 46–50, February
17. Landis K (1993) Group technology and cellular manufacturing in the Westvaco Los Angeles VH department. Project report in IOM 581, School of Business, University of Souther California
18. Lenstra JJ, Rinnoy Kan AHG, Brucker P (1977) Complexity of machine scheduling problems. Ann Discrete Math 1: 343–362
19. Navaz M, Enscore Jr EE, Ham I (1983) A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. OMEGA 11/1:91–95
20. OR Library: http://sprocket.ict.pwr.wroc.pl/~wbo/benchmarks.htm
21. Schaller J (2004) Single-machine scheduling with early and quadratic tardy penalties. Comput Ind Eng 46:511–532
22. Smith WE (1956) Various optimizers for single-stage production. Nav Res Logist Q 3:59–66
23. Szwarc W (1993) Adjacent ordering in single-machine scheduling with earliness and tardiness penalties. Nav Res Logist 40:229–243
24. T'kindt V, Billaut J-C (2002) Multicriteria scheduling: theory, models and algorithms. Springer, Berlin Heidelberg New York
25. Tung-I Tsai (2007) A genetic algorithm for solving the single machine earliness/tardiness problem with distinct due dates and ready times. Int J Adv Manuf Technol 32:994–1000
26. Wan G, Yen BPC (2002) Tabu search for single-machine scheduling with distinct due windows and weighted earliness/tardiness penalties. Eur J Oper Res 142:271–281
27. Valente JMS, Alves RAFS (2005) Filtered and recovering beam search algorithms for the early/tardy scheduling problem with no idle time. Comput Ind Eng 48(2):363–375
28. Yano CA, Kim YD (1991) Algorithms for a class of single-machine weighted tardiness and earliness problems. Eur J Oper Res 52:167–178