

The permutation flow shop problem with blocking. A tabu search approach

Józef Grabowski*, Jarosław Pempera

Wrocław University of Technology, Institute of Engineering Cybernetics, Janiszewskiego 11-17, 50-372 Wrocław, Poland

Processed by A.E. Jeet Gupta

Received 12 September 2004; accepted 7 July 2005

Available online 31 August 2005

Abstract

This paper develops a fast tabu search algorithm to minimize makespan in a flow shop problem with blocking. Some properties of the problem associated with the blocks of jobs have been presented and discussed. These properties allow us to propose a specific neighbourhood of algorithms. Also, the multimoves are used that consist in performing several moves simultaneously in a single iteration and guide the search process to more promising areas of the solutions space, where good solutions can be found. It allow us to accelerate the convergence of the algorithm. Besides, a dynamic tabu list is proposed that assists additionally to avoid being trapped at a local optimum. The proposed algorithms are empirically evaluated and found to be relatively more effective in finding better solutions than attained by the leading approaches in a much shorter time. The presented ideas can be applied in many local search procedures.

© 2005 Elsevier Ltd. All rights reserved.

Keywords: Flow-shop; Blocking; Makespan; Tabu search; Empirical evaluation

1. Introduction

Flow shop scheduling problems exist naturally in many real-life situations, since there are many practical as well as important applications for a job to be processed in series with more than one-stage in industry. This paper considers the blocking flow shop problem to minimize makespan. The classical flow shop problem disregards the behaviour of the jobs between two consecutive operations, i.e. it is assumed that intermediate buffers have infinite capacity and that a job can be stored for unlimited amount of time. In practice, there are many production factories, in which the buffer capacity has to be taken into account, and there may be limits on the

capacity of buffers, and/or on the amount of time that a job can spend in the buffer between two consecutive operations.

In a flow shop problem with blocking, there are no buffers between the machines and hence intermediate queues of jobs waiting in the production system for their next operations are not allowed. With *blocking* constraint a job, having completed processing on a machine, remains on this machine and *blocks* it, until the next machine downstream becomes available for processing. Also, in some processes, technological requirements do not allow storage in manufacturing stages, for example, temperature and other characteristic of the materials require that each operation follows the previous one without the use of intermediate buffers [1]. Such situations arise in the chemical and pharmaceutical industries where partially processed jobs (physical and chemical transformation of materials) are sometimes held in the machines because storage is not allowed. Hall and Sriskandarajah [1] give a good survey of the applications on the flow shop problem with blocking in modern just-in-time and

* Corresponding author. Tel.: +48 71 362 84 82;
fax: +48 71 321 26 77.

E-mail addresses: grabow@ict.pwr.wroc.pl (J. Grabowski),
jpempera@ict.pwr.wroc.pl (J. Pempera).

other manufacturing systems. Grabowski and Pempera [2] present a real-life example in production of concrete blocks that does not allow storage in some stages.

As regards the methods used for solving the problem, Reddi and Ramamoorthy [3] have shown that the flow shop of two machine with blocking can be reduced to a special case of the traveling salesman problem, which then can be solved in $O(n \log n)$ time using an improved implementation by Gilmore et al. [4] of the algorithm of Gilmore and Gomory [5]. For $m > 2$, the problem is unfortunately NP-hard. For this reason many various algorithms have been proposed and tested to find better quality solutions in a short time.

Among heuristic approaches, McCormick et al. [6] have developed a constructive heuristic, known as profile fitting (PF) which creates a partial sequence by adding the unscheduled job that leads to the minimum sum of idle times and blocking times on machines. Another approach is presented by Leistein [7] who has proposed some algorithms adopted from the cases of no-wait requirements (where the jobs do not wait for processing), and two special heuristics that optimize the use of buffers storage. However, it is concluded that the heuristics do not produce better solutions than those provided by the NEH algorithm of Nawaz et al. [8]. Recently, Ronconi [9] has provided three constructive heuristics, including the PF heuristic and the enumeration scheme of NEH, that combined together outperform the NEH. Abadi et al. [10] has proposed a heuristic (for minimization the cycle time) based on the idea of slowing down certain operations (i.e. increasing their processing times) in order to establish a connection between the no-wait flow shop problem and flow shop with blocking. This approach, for calculating the value of makespan for a given sequence of the jobs, has been used by Caraffa et al. [11] to develop a genetic algorithm (GA) to minimize makespan. Computational results confirm that GA outperforms the heuristic of Abadi.

Recently, Ronconi [12] has developed an interesting exactly algorithm based on branch-and-bound method, using the new lower bounds that exploit the blocking nature and are better than those presented in earlier paper [13]. The computational results reported are basic for the comparisons with our algorithms.

In this paper, we propose two new heuristic algorithms to minimize makespan based on tabu search approach. So far, for the flow shop problem with blocking, tabu search algorithms have not been applied. While for the classic flow shop problem they were successfully used by many authors (see, e.g. [14–18]).

The paper is organized as follows. In Section 2, the problem is defined and formulated. Section 3 presents the moves and neighbourhood structure, search process, dynamic tabu list, and heuristic algorithm. 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

The flow-shop problem with blocking can be formulated as follows.

Problem. Each of n jobs from the set $J = \{1, 2, \dots, n\}$ has to be processed on m machines $1, 2, \dots, m$ in that order, having no intermediate buffers. Job $j \in J$, consists of a sequence of m operations $O_{j1}, O_{j2}, \dots, O_{jm}$; operation O_{jk} corresponds to the processing of job j on machine k during an uninterrupted processing time p_{jk} . Since the flow shop has no intermediate buffers, a job j , having completed processing of the operation O_{jk} , cannot leave the machine k , until the next machine $k + 1$ is free, $k = 1, 2, \dots, m - 1$, $j \in J$. If the machine $k + 1$ is not free, then job j is blocked on the machine k . We want to find a schedule such that the processing order of jobs is the same on each machine and the maximum completion time is minimal.

Each schedule of jobs can be represented by permutation $\pi = (\pi(1), \dots, \pi(n))$ on set J . Let Π denote the set of all such permutations. We wish to find such permutation $\pi^* \in \Pi$, that

$$C_{\max}(\pi^*) = \min_{\pi \in \Pi} C_{\max}(\pi),$$

where $C_{\max}(\pi)$ is the time required to complete all jobs on the machines in the processing order given by π . It is obvious that $D_{ik} \geq C_{ik}$, where D_{ik} and C_{ik} are the departure time and completion time of job i on machine k , respectively. It is known from literature [9] that, for a given permutation π , $C_{\max}(\pi) = D_{\pi(n)m}$, where the departure time $D_{\pi(j)k}$ of job $\pi(j)$ on machine k can be found using the following recursive expressions

$$\begin{aligned} D_{\pi(1)0} &= 0, \\ D_{\pi(1)k} &= \sum_{l=1}^k p_{\pi(1)l}, \quad k = 1, \dots, m - 1, \\ D_{\pi(j)0} &= D_{\pi(j-1),1}, \quad j = 2, \dots, n, \\ D_{\pi(j)k} &= \max\{D_{\pi(j)k-1} + p_{\pi(j)k}, D_{\pi(j-1),k+1}\}, \\ & \quad j = 2, \dots, n, \quad k = 1, \dots, m - 1, \\ D_{\pi(j)m} &= D_{\pi(j),m-1} + p_{\pi(j)m}, \quad j = 1, \dots, n, \end{aligned}$$

where $D_{\pi(j)0}$, $j = 1, \dots, n$, denotes a starting time of job $\pi(j)$ on the first machine. In the recursion, first, it is calculated the departure time of the first job in π , then of the second job, until the last one, and since $C_{\pi(j)m} = D_{\pi(j)m}$ for all j , then $C_{\max}(\pi)$ is given by $D_{\pi(n)m}$. Thus, the value of $C_{\max}(\pi)$ for given π can be found in $O(nm)$ time.

It is useful to present the flow shop problem with blocking, using a grid graph by Grabowski and Pempera [2] and Smutnicki [19] as most proper for the problem considered, see Fig. 1. For the given processing order π , we create the

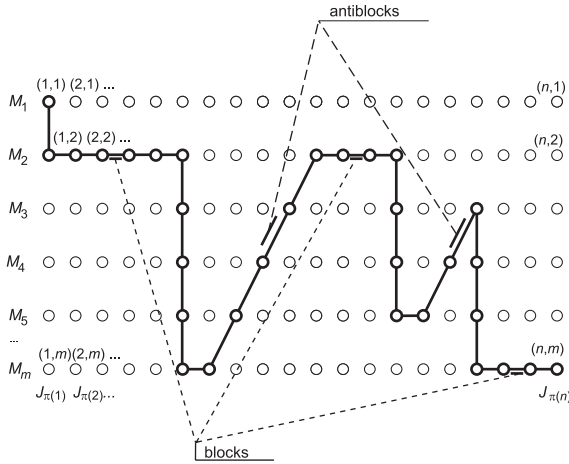


Fig. 1. Blocks and antiblocks on critical path.

graph $G(\pi) = (N, R \cup F^0(\pi) \cup F^-(\pi))$ with a set of nodes N and a set of arcs $R \cup F^0(\pi) \cup F^-(\pi)$, where

- $N = \{1, \dots, n\} \times \{1, \dots, m\}$, where node (j, k) represents the k th operation of job $\pi(j)$. The weight of node $(j, k) \in N$ is given by the processing time $p_{\pi(j)k}$.
- $R = \bigcup_{j=1}^n \bigcup_{k=1}^{m-1} \{((j, k), (j, k+1))\}$. Thus, R contains arcs connecting consecutive operations of the same job.
- $F^0(\pi) = \bigcup_{j=1}^n \bigcup_{k=1}^{m-1} \{((\pi(j), k), (\pi(j+1), k))\}$. Arcs from $F^0(\pi)$ connect jobs to be processed on the machines in the processing order given by π . Each arc of $F^0(\pi)$ has weight zero.
- $F^-(\pi) = \bigcup_{j=1}^n \bigcup_{k=1}^{m-1} \{((\pi(j), k+1), (\pi(j+1), k))\}$. Each arc $((\pi(j), k+1), (\pi(j+1), k)) \in F^-(\pi)$ has weight minus $p_{\pi(j), k+1}$ and ensures the blocking of job $\pi(j+1)$ (i.e. operation $O_{\pi(j+1), k}$) on the machine k , if the machine $k+1$ is not free.

The makespan $C_{\max}(\pi)$ of the flow-shop problem with blocking is equal to the length of the longest (critical) path from node $(1, 1)$ to (n, m) in $G(\pi)$. Now, we can rephrase the problem as that of finding a processing order $\pi \in \Pi$ that minimizes $C_{\max}(\pi)$ in the resulting graph. For given π , $C_{\max}(\pi)$ can be found in time $O(nm)$.

Each path from $(1, 1)$ to (n, m) can be represented by a sequence of nodes, and let denote a critical path in $G(\pi)$ by $u = (u_1, u_2, \dots, u_w)$, where $u_i = (j_i, k_i) \in N$, $1 \leq i \leq w$ and w is the number of nodes in this path, see Fig. 1. Obviously, it has to be $u_1 = (j_1, k_1) = (1, 1)$, and $u_w = (j_w, k_w) = (n, m)$. Clearly, there may exist several critical paths, however further we consider only one of them, arbitrary selected. The critical path u depends on π , but for simplicity in notation we will not express it explicitly. The critical path can naturally be decomposed into several specific subpaths and each

of them contains the nodes linked by the same type of arcs, i.e. all arcs of the subpath belong either to $F^0(\pi)$ or $F^-(\pi)$.

The first type of subpaths is determined by a maximal sequence $(u_g, \dots, u_h) = ((j_g, k_g), \dots, (j_h, k_h))$ of u such that $k_g = k_{g+1} = \dots = k_h$ and $((j_i, k_i), (j_{i+1}, k_{i+1})) \in F^0(\pi)$ for all $i = g, \dots, h-1$, $g < h$. Each such subpath determines the sequence of jobs

$$B_{gh} = (j_g, j_{g+1}, \dots, j_{h-1}, j_h),$$

which is called the *block* of jobs in π , see Fig. 1. Jobs j_g and j_h in B_{gh} are the *first* and *last* ones, respectively. A block corresponds to a sequence of jobs (operations) processed on machine k_g without inserted idle time.

Next, we define the *internal block* of B_{gh} as the subsequence

$$B_{gh}^* = B_{gh} - \{j_g, j_h\}.$$

The second type of subpaths is defined by a maximal sequence $(u_s, \dots, u_t) = ((j_s, k_s), \dots, (j_t, k_t))$ of u such that $((j_i, k_i), (j_{i+1}, k_{i+1})) \in F^-(\pi)$ for all $i = s, \dots, t-1$, $s < t$. Each such subpath determines the sequence of jobs

$$A_{st} = (j_s, j_{s+1}, \dots, j_{t-1}, j_t),$$

which is called the *antiblock* of jobs in π , see Fig. 1. Similarly to in B_{gh} , jobs j_s and j_t in A_{st} are the *first* and *last* ones, respectively. An antiblock corresponds to blocked jobs. Note that by the definition, operation $O_{\pi(j_i)k_i}$ of job $j_{\pi(j_i)}$ is processed on machine k_i with $k_i > k_{i+1} = k_i - 1$, for all $i = s, \dots, t-1$.

Similarly to in B_{gh} , now we define the *internal antiblock* of A_{st} as the subsequence

$$A_{st}^* = A_{st} - \{j_s, j_t\}.$$

No other type of subsequences have been detected. Note that all the blocks and antiblocks are connected in series in critical path u , see Fig. 1.

The critical path u can contain many different blocks and antiblocks, and let l_B and l_A denote, respectively, the numbers of blocks and antiblocks in u . Each of blocks (or antiblocks) can be characterized by the pair (g_i, h_i) , $i = 1, \dots, l_B$ (or (s_i, t_i) , $i = 1, \dots, l_A$), and let $GH = \{(g_i, h_i) \mid i = 1, \dots, l_B\}$ (or $ST = \{(s_i, t_i) \mid i = 1, \dots, l_A\}$) be the set of the pairs denoting all blocks (or antiblocks) in u .

Property 1 (Grabowski and Pempera [2]). *For any block B_{gh} in π , let α be a processing order obtained from π by an interchange of jobs from the internal block B_{gh}^* . Then we have $C_{\max}(\alpha) \geq C_{\max}(\pi)$.*

Property 2 (Grabowski and Pempera [2]). *For any antiblock A_{st} in π , let α be a processing order obtained from π by an interchange of jobs from the internal antiblock A_{st}^* . Then we have $C_{\max}(\alpha) \geq C_{\max}(\pi)$.*

Immediately from Properties 1 and 2, it results the following Theorem

Theorem 1. *Let $\pi \in \Pi$ be any permutation with blocks B_{gh} and antiblocks A_{st} . If the permutation β has been obtained from π by an interchange of jobs that $C_{\max}(\beta) < C_{\max}(\pi)$, then in β*

- (i) *at least one job $j \in B_{gh}$ precedes job j_g , for some $(g, h) \in GH$, or*
- (ii) *at least one job $j \in B_{gh}$ succeeds job j_h , for some $(g, h) \in GH$, or*
- (iii) *at least one job $j \in A_{st}$ precedes job j_s , for some $(s, t) \in ST$, or*
- (iv) *at least one job $j \in A_{st}$ succeeds job j_t , for some $(s, t) \in ST$.*

Note that Theorem 1 provides the necessary condition to obtain a permutation β from π such that $C_{\max}(\beta) < C_{\max}(\pi)$.

3. Tabu search algorithms (TS and TS + M)

Currently, tabu search approach, (see [20,21]), 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 tabu search algorithm 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 getting trapped at a local optimum. The algorithm tabu search 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 algorithms, there are used some of the components that have been proposed by Grabowski and Wodecki [18,22],

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 neighbourhood

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 [16]). The intuition following from Theorem 1 suggests that the “insert” type is most proper for the problem considered. In general, the insert move operates on a sequence of jobs on a machine and removes a job placed at a position in this sequence and inserts it in another position of the sequence. More precisely, let $v = (\pi(x), \pi(y))$ be a pair of jobs in a permutation π , $x, y \in \{1, 2, \dots, n\}$, $x \neq y$. The pair $v = (\pi(x), \pi(y))$ defines a move in π . This move consists in removing job $\pi(x)$ from its original position x , and next inserting it in the position immediately after job $\pi(y)$ (or before $\pi(y)$) in π if $x < y$ (or $x > y$). Thus the move v generates a permutation π_v from π in the following way

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

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

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

Next we will give a detailed description of the moves and neighbourhood structure used in our algorithms. For each job j we consider at most one move to the right and at most one to the left. Moves are associated with blocks and antiblocks. Let us take the block $B_{gh} = (j_g, j_{g+1}, \dots, j_{h-1}, j_h)$ in π . Then, we define the sets of *candidates*

$$E_{gh}^a = \{j_g, j_{g+1}, \dots, j_{h-1}\} = B_{gh} - \{j_h\},$$

$$E_{gh}^b = \{j_{g+1}, \dots, j_{h-1}, j_h\} = B_{gh} - \{j_g\}.$$

Each set E_{gh}^a (or E_{gh}^b) contains the jobs from block B_{gh} of π that are candidates for being moved to a position *after* (or *before*) all other jobs in this block. More precisely, we move job j , $j \in E_{gh}^a$, to the right in the position immediately after job j_h , and this move takes the form $v = (j, j_h)$. By symmetry, job j , $j \in E_{gh}^b$, is moved to the left in the position immediately before job j_g , and this move takes the form $v = (j, j_g)$. Note that after performing a move $v = (j, j_h)$, $j \in E_{gh}^a$ (or $v = (j, j_g)$, $j \in E_{gh}^b$), job j in π_v is to be processed as the last (or first) job of block B_{gh} of π .

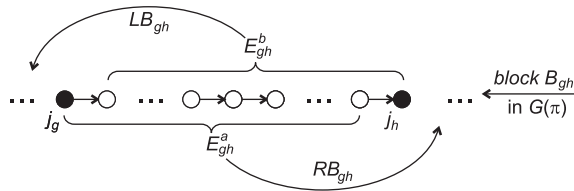


Fig. 2. Job movements.

According to the description given, for any block B_{gh} in π , we define the following set of moves to the right

$$RB_{gh} = \{(j, j_h) \mid j \in E_{gh}^a\},$$

and the set to the left

$$LB_{gh} = \{(j, j_g) \mid j \in E_{gh}^b\}.$$

Set RB_{gh} contains all moves of jobs of E_{gh}^a to the right after the last job j_h of block B_{gh} . By analogy, set LB_{gh} contains all moves of jobs of E_{gh}^b to the left before the first job j_g of block B_{gh} . It should be found that for each move $v \in RB_{gh}$ (or $v \in LB_{gh}$), the necessary condition of Theorem 1 is satisfied to obtain a permutation π_v from π such that $C_{\max}(\pi_v) < C_{\max}(\pi)$. For illustration, the moves performed to the right and left are shown in Fig. 2.

As a consequence, in our algorithms, we will employ the set of moves

$$MB = \bigcup_{(g,h) \in GH} [RB_{gh} \cup LB_{gh}].$$

The similar considerations can be provided for the antiblocks A_{st} , and we then obtain the set of moves

$$MA = \bigcup_{(s,t) \in ST} [RA_{st} \cup LA_{st}].$$

Finally, in our algorithms, we propose the following set of moves

$$M = MB \cup MA,$$

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

3.2. Tabu search algorithm (TS)

Similarly to in other local search algorithms, our TS starts from an initial basic permutation π that implies neighbourhood $N(M, \pi)$. This neighbourhood is searched in the following manner.

First, the “best” move $v^* \in M$ that generates the permutation $\pi_{v^*} \in N(M, \pi)$ with the lowest makespan is chosen, i.e. $C_{\max}(\pi_{v^*}) = \min_{v \in M} C_{\max}(\pi_v)$. If $C_{\max}(\pi_{v^*}) < C^*$ (where C^* is the best makespan found so far), then the move v^* is selected for the search process. Otherwise, i.e.

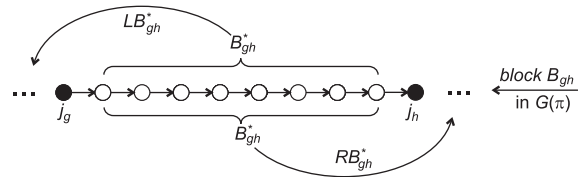


Fig. 3. Illustration sets.

if $C_{\max}(\pi_{v^*}) \geq C^*$, then the set of *unforbidden* moves (UF) that do not have the tabu status, is defined

$$UM = \{v \in M \mid \text{move } v \text{ is UF}\}.$$

Next, the “best” move $v^* \in UM$ that generates the permutation $\pi_{v^*} \in N(UM, \pi)$ with the lowest makespan is chosen for the search, i.e. $C_{\max}(\pi_{v^*}) = \min_{v \in UM} C_{\max}(\pi_v)$.

If the move v^* is selected, then a pair of jobs corresponding to the move v^* is added to the tabu list (see Section 3.4 for details) and the resulting permutation π_{v^*} is created. Next, this permutation becomes the new basic one, i.e. $\pi := \pi_{v^*}$ and algorithm restarts to next iteration. If all moves from M are forbidden (a very rare case), i.e. if $UM = \emptyset$, then the oldest element of tabu list is deleted and the search is repeated until a *UF* move is found.

3.3. Tabu search algorithm with multimoves (TS + M)

In order to accelerate the convergence of algorithm to good solutions, we develop an improved algorithm to TS, called as TS + M, by using the *multimoves*. A multimove consists of *several* moves that are performed *simultaneously* in a single iteration of algorithm. The performances of the multimoves 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 multimoves 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 multimoves 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 TS + M to provide the multimoves.

For the blocks B_{gh} , we consider the following sets of moves

$$RB_{gh}^* = \{(j, j_h) \mid j \in B_{gh}^*\},$$

$$LB_{gh}^* = \{(j, j_g) \mid j \in B_{gh}^*\}.$$

Set RB_{gh}^* (or LB_{gh}^*) contains all moves that insert the jobs from the internal block B_{gh}^* after the last job j_h (or before the first job j_g) of this block, see Fig. 3. Note that $RB_{gh}^* \subset RB_{gh}$ and $LB_{gh}^* \subset LB_{gh}$.

Next, for the block B_{gh} , the “best” move $v_{R(gh)} \in RB_{gh}^*$ and $v_{L(gh)} \in LB_{gh}^*$ are chosen (respectively)

$$C_{\max}(\pi_{v_{R(gh)}}) = \min_{v \in RB_{gh}^*} C_{\max}(\pi_v),$$

$$C_{\max}(\pi_{v_{L(gh)}}) = \min_{v \in LB_{gh}^*, v \neq v_{R(gh)}} C_{\max}(\pi_v),$$

and the following sets of moves are created

$$RB = \{v_{R(gh)} \mid (g, h) \in GH\},$$

$$LB = \{v_{L(gh)} \mid (g, h) \in GH\},$$

and

$$BB = RB \cup LB = \{v_1, v_2, \dots, v_{2l_B}\}.$$

Let

$$BB^{(-)} = \{v \in BB \mid C_{\max}(\pi_v) < C_{\max}(\pi)\}$$

$$= \{v_1, v_2, \dots, v_p\}, \quad p \leq 2l_B$$

be the set of the *profitable moves* of blocks B_{gh} , the performance of which generates permutation π_v “better” than π .

The similar consideration can be provided for the antiblocks A_{st} , and we then obtain their the set of *profitable moves*

$$BA^{(-)} = \{v \in BA \mid C_{\max}(\pi_v) < C_{\max}(\pi)\}$$

$$= \{v_1, v_2, \dots, v_q\}, \quad q \leq 2l_A.$$

The main idea of a multimove is to consider a search which allows us several moves to be made in a single iteration and carry the search to the more promising areas of solution space, where “good solutions” can be found. In our algorithm, the set of promising moves can be defined as follows

$$BM^{(-)} = BB^{(-)} \cup BA^{(-)}$$

$$= \{v_1, v_2, \dots, v_z\}, \quad z \leq 2(l_B + l_A).$$

From the definition of $BM^{(-)}$ it results that each move $v \in BM^{(-)}$ produces permutation π_v “better” than π . Therefore, as a *multimove*, we took the set $BM^{(-)}$. The use of this multimove consists in performing *all* the moves from $BM^{(-)}$ *simultaneously*, generating a permutation, denoted as $\pi_{\bar{v}}$, where $\bar{v} = BM^{(-)}$. To simplify, in the further considerations, multimove $BM^{(-)}$ will be denoted alternatively by \bar{v} . Note that the permutation $\pi_{\bar{v}}$ does not belong to $N(M, \pi)$, unless $|\bar{v}| = 1$. The intuition following from the definition of \bar{v} suggests that $\pi_{\bar{v}}$ should be significantly better than π_v generated by the best (single) move $v \in \bar{v}$, since the total improvement of $C_{\max}(\pi_{\bar{v}})$ can be obtained by combining all the improvements produced by the individual moves from \bar{v} . It allows the algorithm to achieve very good solutions in a much shorter time. Therefore, the performance of multimove \bar{v} guides the search to visit new more promising regions of the solution space where good solutions can be found. Note that if $\bar{v} = \emptyset$, then the multimove cannot be used.

For algorithm TS + M, similarly to in Grabowski and Wodecki [18,22], we propose to use a multimove in some specific situations, namely, when at least *Piter consecutive non-improving iterations* pass in the algorithm. More precisely, if permutation $\pi_{\bar{v}}$ is obtained by performing a multimove \bar{v} , then the next one is made when *Piter* of the iterations will pass in TS + M. In other words, the multimove is used periodically, where *Piter* is the number of the iterations between the neighbouring ones. The *Piter* is a tuning parameter which is to be chosen experimentally.

If a multimove is not performed (or if $\bar{v} = \emptyset$), then the search process is continued according to the description given in Section 3.2. If a multimove \bar{v} is performed, then a pair of jobs corresponding to the move $v^* \in \bar{v}$ with the smallest value of $C_{\max}(\pi_{v^*})$ is added to tabu list T (see Section 3.4 for details).

3.4. Tabu list and tabu status of move

In our algorithms we use the cyclic tabu list defined as a finite list (set) T with length $LengthT$ containing ordered pairs of jobs. The list T is a realization of the short-term search memory. If a move $v = (\pi(x), \pi(y))$ is performed on permutation π , then the pair of jobs $(\pi(x), \pi(x + 1))$ if $x < y$, or the pair $(\pi(x - 1), \pi(x))$ if $x > y$, representing a precedence constraint, is added to T . Each time before adding a new element to T , we must remove the oldest one. With respect to a permutation π , a move $(\pi(x), \pi(y)) \in M$ is forbidden, i.e. it has *tabu* status, if $A(\pi(x)) \cap \{\pi(x + 1), \pi(x + 2), \dots, \pi(y)\} \neq \emptyset$ if $x < y$, and $B(\pi(x)) \cap \{\pi(y), \pi(y + 1), \dots, \pi(x - 1)\} \neq \emptyset$ otherwise, where

$$A(j) = \{i \in J \mid (j, i) \in T\},$$

$$B(j) = \{i \in J \mid (i, j) \in T\}.$$

Set $A(j)$ (or set $B(j)$) indicates which jobs are to be processed *after* (or *before*) job j with respect to the current content of the tabu list T .

As mentioned above, our algorithms use a tabu list with dynamic length. This length is changed, as the current iteration number *iter* increases. The length change is used as a “pick” intended to carry the search to another area of the solutions space. It can be viewed as a specific disturbance that gives an additional assistance to avoid getting trapped at a local optimum of algorithm.

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 [18,22]. Here, we extend this component in the original form [22], to the problem considered. In this tabu list, length $LengthT$ is a cyclic function shown in Fig. 4, and defined by the expression

$$LengthT = \begin{cases} LTS & \text{if } W(l) < iter \leq W(l) + H(l), \\ 1, 5 \times LTS & \text{if } W(l) + H(l) < iter \\ & \leq W(l) + H(l) + h, \end{cases}$$

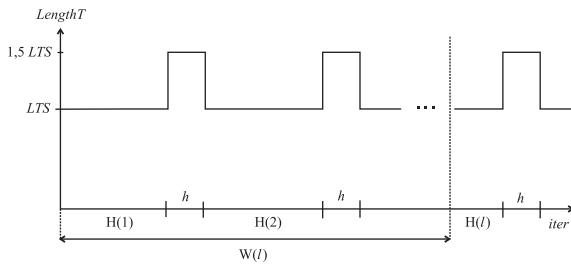


Fig. 4. Dynamic tabu list.

where $l = 1, 2, \dots$ is the number of the cycle, $W(l) = \sum_{s=1}^l H(s-1) + (l-1) \times h$ (here $H(0) = 0$), and LTS is the value taken from [16] equal to $6 + \lceil n/(10m) \rceil$, where $\lceil x \rceil$ represents the integer of x . Further, h is the width of the pick equal to $2 \times \text{LTS}$, and $H(l)$ is the interval between the neighbour picks equal to $6 \times \text{LTS}$. If $\text{Length}T$ decreases then a suitable number of the oldest elements of tabu list T is deleted and the search process is continued.

4. Computational results

In this section, we report the results of empirical tests to evaluate the relative effectiveness of the proposed tabu search algorithms. In this paper, we compare our algorithms TS and TS + M, and that algorithm GA by Caraffa et al. [11] with an exactly algorithm proposed recently by Ronconi [12], here denoted as RON, because, it is natural to inquire whether these heuristics find comparable solutions with the ones provided by RON, which is based on the branch-and-bound method. Besides, it allows us to compare TS and TS + M with GA against a background of RON.

Our algorithms TS and TS + M, and GA were coded in C++, run on a PC with Pentium IV 1000 MHz processor and tested, similarly to in RON, on the benchmark instances provided by Taillard [23] (see also OR Library [24] for more details) for the classic permutation flow shop, by considering all machines as the blocking constraints are required. The benchmark set contains 120 particularly hard instances of 12 different sizes, selected from a large number of randomly generated problems. For each size (group) $n \times m$: 20×5 , 20×10 , 20×20 , 50×5 , 50×10 , 50×20 , 100×5 , 100×10 , 100×20 , 200×10 , 200×20 , 500×20 , a sample of 10 instances was provided [24].

Our algorithms TS and TS + M need an initial permutation, which can be found by any method. In our tests, we use, similarly to in RON, algorithm NEH [8] in its original version, which is considered to be the best one among simple constructive heuristics for flow-shop scheduling.

In our tests, TS and TS + M are terminated after performing a number Maxiter of iterations on each instance. For TS + M, in order to detect its convergence in relation to RON, GA and TS, we set $\text{Maxiter} = 100, 200, 500, 1000,$

3000, 10,000, and 30,000 iterations. While for TS, we set $\text{Maxiter} = 30,000$ iterations. For GA, we set the same values of parameters as those in the paper of Caraffa et al. [11], in particular, we set a number of generations (Maxiter) equal to 100. The value of tuning parameter Piter for TS + M is drawn from [18] equal to 3. The effectiveness of the algorithms was analyzed in both terms of CPU time and solution quality.

For each test instance, we collected the following values

- C^A —the makespan found by the algorithm $A \in \{\text{TS}, \text{TS} + \text{M}, \text{GA}\}$.
- Time —CPU in seconds.

In the case of non-deterministic algorithm GA, C^{GA} and Time are averaged values over 10 runs for each instance.

Then, for each group $n \times m$, the following measures of the algorithm quality were calculated

- $\text{PRD}(A) = 100(C^{\text{RON}} - C^A)/C^{\text{RON}}$ —the value (average for 10 instances) of the percentage relative difference between the reference makespan C^{RON} produced by algorithm RON [12], and makespan C^A of algorithm A.
- $\text{CPU}(A)$ —the computer time (average for 10 instances) of algorithm A (in seconds).

The computational results are summarized in Tables 1 and 2. Comparing the results, it should be highlighted that algorithm RON was performed on a PC Pentium IV 1400 MHz with limited CPU time to 3600 s, for each instance. Note that, by the definition, if $\text{PRD}(A)$ is positive, then algorithm A improves some makespans of RON.

The results reported in Table 1 indicate that our algorithm TS + M, tested under a number of iterations larger than 500, produces makespans with the positive overall PRD values. For 500 iterations, TS + M found makespans with $\text{PRD} = 0.23$ (in CPU time = 2.6 s for the largest instances with $n \times m = 500 \times 20$), and for 30,000 iterations it found the ones with $\text{PRD} = 1.68$. The superiority of TS + M over RON increases as, for a given n , the number of machines m increases. Unfortunately, not for all instances the results are better in terms of the PRD values. For the groups with $(n \times m) = (20 \times 5)$ and (100×5) , some makespans of RON are better, for the average, than those of TS + M, inducing the negative values of PRD. For these instances, TS + M needs more than 30,000 iterations to obtain better PRD values. With respect to CPU times, it should be seen that for the group with the extreme case of 500 jobs and 20 machines, the CPU time of TS + M (on a PC Pentium IV 1000 MHz) is not greater than 210 s, whereas, algorithm RON was run (on a PC Pentium IV 1400 MHz) with limited CPU time to 3600 s on each instance. Generally speaking, TS + M produces better results (in terms of PRD values) than RON in significantly shorter time.

Regarding a comparison TS and RON, it should be noted that in the terms of PRD values, TS produces better results

Table 1
Computational results^a of TS, TS + M, and GA

Algorithm	GA ^b		TS		TS + M		100		200		500		1000		3000		10,000		30,000		
	PRD	CPU	PRD	CPU	PRD	CPU	PRD	CPU	PRD	CPU	PRD	CPU	PRD	CPU	PRD	CPU	PRD	CPU	PRD	CPU	
Maxiter=	100		30,000																		
$n \times m$																					
20 × 5	-6.36	0.1	-1.64	2.4	-3.04	0.0	-2.52	0.0	-2.02	0.0	-1.86	0.1	-1.51	0.2	-0.86	0.9	-0.34	2.7			
20 × 10	-4.35	0.2	1.45	4.1	-0.95	0.0	-0.70	0.0	-0.16	0.1	0.33	0.2	1.26	0.4	1.52	1.6	1.76	4.6			
20 × 20	-1.26	0.4	2.88	7.1	1.32	0.0	1.49	0.1	1.96	0.2	2.26	0.3	2.39	0.7	2.66	2.6	2.94	7.6			
50 × 5	-8.53	0.3	-0.55	6.0	-1.51	0.0	-1.21	0.0	-0.74	0.1	-0.50	0.2	-0.16	0.6	0.38	2.0	0.55	6.2			
50 × 10	-5.97	0.5	1.98	10.6	0.73	0.0	0.95	0.1	1.46	0.2	1.85	0.4	2.27	1.1	3.23	3.6	3.52	10.8			
50 × 20	-4.33	1.1	3.68	19.0	1.08	0.0	1.25	0.1	1.91	0.3	2.11	0.6	2.58	1.9	3.58	6.4	4.26	19.3			
100 × 5	-14.40	0.5	-3.03	12.2	-3.54	0.1	-3.40	0.1	-3.28	0.2	-3.12	0.4	-2.95	1.2	-2.64	4.1	-2.62	12.4			
100 × 10	-7.89	1.1	1.71	21.9	0.91	0.1	1.07	0.2	1.31	0.4	1.52	0.7	1.79	2.2	2.44	7.3	2.66	22.1			
100 × 20	-5.64	2.1	2.01	39.2	0.98	0.2	1.11	0.3	1.40	0.7	1.73	1.2	2.11	4.0	2.65	13.0	3.03	39.4			
200 × 10	-11.04	2.2	-0.60	44.1	-0.89	0.2	-0.81	0.3	-0.63	0.8	-0.51	1.6	-0.16	4.5	0.30	15.0	0.58	44.3			
200 × 20	-7.00	4.3	1.24	79.2	0.52	0.3	0.68	0.5	0.82	1.4	1.06	2.7	1.49	8.0	1.99	26.5	2.31	79.4			
500 × 20	-8.08	10.8	0.63	207	0.56	0.5	0.62	1.0	0.67	2.6	0.76	7.1	0.86	20.5	1.13	69.8	1.47	209			
All	-7.07		0.81		-0.32		-0.12		0.23		0.47		0.83		1.36		1.68				

^aCPU times on Pentium IV 1000MHz.

^bAveraged values over 10 runs for each instance.

than RON for most of the groups and for all instances as well. Besides, CPU times of TS are much shorter than those of RON.

From Table 1, it follows that tabu search heuristic TS + M performs significantly better than the existing heuristic of GA. For 100 iterations, TS + M found makespans with the overall PRD value equal to -0.32, whereas GA found the ones with PRD = -7.07. Especially, TS + M is far superior to the GA for larger sizes of instances. With respect to CPU times, it appears that, for all groups (sizes), CPU times of TS + M for 100 iterations are substantially shorter than for GA. Generally, it should be observed that the PRD values of TS + M are much greater than those of GA, while the CPU times of TS + M are much shorter than those of GA.

Also, we have tested GA for the number of generations equal to 10,000. Then the overall PRD value was -0.53, whereas CPU times were much larger. For the group with the extreme case of 500 jobs and 20 machines, PRD was equal to -2.57 with CPU = 475.0 s.

Note that GA is non-deterministic algorithm, thus, it can produce different results for the same data. Therefore, in order to evaluate the efficiency of GA, it is necessary to run the algorithm multiple, whereas TS and TS + M, as deterministic ones, need a single run. In our tests, GA was run 10 times for each instance, and the results presented in Table 1 are averaged over the runs.

Comparing algorithm TS + M with TS, it appears that TS + M produces significantly better PRD values. For 30,000 iterations, the PRD values for all instances obtained by TS + M and TS are equal to 1.68 and 0.81, respectively. Analysing the performance of TS + M and TS for all instances, we observe that TS + M converges to the good solutions significantly faster than TS. The overall PRD value produced by TS + M for 3000 iterations is comparable with the one provided by TS in 30,000 iterations.

Table 2 reports the makespans provided in Ref. [12] (RON), and those makespans found by TS + M for Maxiter = 30,000. A general purpose is to provide, for future research, the new reference makespans, better than those presented recently in the literature. It should be revealed that for 94 out of 120 instances, TS + M has produced better makespans (highlighted in bold) than RON.

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

5. Conclusions

In this paper, we have presented tabu search algorithms to minimize makespan in a flow shop problem with blocking.

In order to decrease the computational effort for the search, we propose to use the multimoves that consist in performing several moves simultaneously in a single iteration

Table 2
Upper bounds produced by TS + M for Maxiter=30,000, comparison with RON^a

TS + M	RON	TS + M	RON	TS + M	RON	TS + M	RON
20 × 5		50 × 5		100 × 5		200 × 10	
1387	1384	3163	3151	6639	6455	14,220	14,113
1424	1411	3348	3395	6481	6214	14,089	14,127
1293	1294	3173	3184	6299	6124	14,149	14,416
1451	1448	3277	3303	6120	5976	14,156	14,435
1348	1366	3338	3272	6340	6173	14,130	14,119
1366	1363	3330	3400	6244	6094	13,963	13,909
1387	1381	3168	3228	6346	6262	14,386	14,563
1388	1384	3228	3260	6289	6061	14,256	14,329
1392	1378	3068	3104	6559	6474	13,954	13,923
1302	1283	3285	3264	6509	6366	14,224	14,435
20 × 10		50 × 10		100 × 10		200 × 20	
1698	1736	3776	3913	7320	7496	15,334	15,579
1836	1897	3641	3798	7108	7281	15,522	15,728
1674	1677	3588	3723	7233	7400	15,713	15,915
1555	1622	3786	3885	7413	7670	15,687	16,039
1631	1658	3745	3934	7168	7317	15,443	15,938
1603	1640	3747	3831	6993	7301	15,472	15,911
1629	1634	3778	3957	7092	7247	15,522	15,898
1754	1741	3708	3774	7143	7315	15,540	16,022
1759	1777	3668	3784	7327	7631	15,394	15,817
1782	1847	3729	3928	7299	7411	15,523	15,969
20 × 20		50 × 20		100 × 20		500 × 20	
2449	2530	4627	4886	8101	8347	37,860	38,334
2242	2297	4411	4668	8105	8372	38,044	38,642
2483	2560	4388	4666	8071	8265	37,732	38,163
2348	2399	4479	4650	8081	8365	38,062	38,625
2450	2538	4359	4475	8074	8304	37,991	38,492
2398	2467	4372	4521	8151	8450	38,132	38,551
2397	2502	4402	4576	8273	8507	37,561	38,179
2345	2411	4444	4688	8248	8584	37,750	38,664
2363	2421	4423	4532	8116	8341	37,730	38,339
2334	2407	4609	4846	8261	8489	38,014	38,540

^aThe results of RON drawn from [12].

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 being 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 the proposed algorithm provides better results than attained by the leading approaches. Nevertheless, some improvements in our algorithm are possible. For instance, attempts to calculate the lower bounds on the makespans instead of computing makespans explicitly for selecting the best solution and to refine the multi-

moves may induce 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 to the problems with different objective functions or to other sequencing problems.

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.

References

- [1] Hall NG, Sriskandarajah C. A survey of machine scheduling problems with blocking and no-wait in process. *Operations Research* 1996;44:510–25.
- [2] Grabowski J, Pempera J. Sequencing of jobs in some production system. *European Journal of Operational Research* 2000;125:535–50.
- [3] Reddi SS, Ramamoorthy CV. On flowshop sequencing problems with no-wait in process. *Operational Research Quarterly* 1972;23:323–31.
- [4] Gilmore PC, Lawler EL, Shmoys DB. Well-solved special cases. In: Lawler LE, Lenstra JK, Rinnooy Kan AHG, Shmoys DB, editors. *The traveling salesman problem: a guided tour of combinatorial optimization*. Chichester: Wiley; 1985. p. 87–143.
- [5] Gilmore PC, Gomory RE. Sequencing a state-variable machine: a solvable case of the traveling salesman problem. *Operations Research* 1964;12:655–79.
- [6] McCormick ST, Pinedo ML, Shenker S, Wolf B. Sequencing in an assembly line with blocking to minimize cycle time. *Operations Research* 1989;37:925–35.
- [7] Leistein R. Flowshop sequencing with limited buffer storage. *International Journal of Production Research* 1990;28:2085–100.
- [8] Nawaz M, Enscore EE, Ham I. A heuristic algorithm for the m -machine, n -job flowshop sequencing problem. *OMEGA The International Journal of Management Science* 1983;11:91–5.
- [9] Ronconi DP. A note on constructive heuristics for the flowshop problem with blocking. *International Journal of Production Economics* 2004;87:39–48.
- [10] Abadi INK, Hall NG, Sriskandarajah C. Minimizing cycle time in a blocking flowshop. *Operations Research* 2000;48:177–80.
- [11] Caraffa V, Ianes S, Bagchi TP, Sriskandarajah C. Minimizing makespan in a blocking flowshop using genetic algorithms. *International Journal of Production Economics* 2001;70:101–15.
- [12] Ronconi DP. A branch-and-bound algorithm to minimize the makespan in a flowshop problem with blocking. *Annals of Operations Research*, in press.
- [13] Ronconi DP, Armentano VA. Lower bounding schemes for flowshops with blocking in-process. *Journal of the Operational Research Society* 2001;52:1289–97.
- [14] Widmer M, Hertz A. A new heuristic method for the flowshop sequencing problem. *European Journal of Operational Research* 1989;42:186–93.
- [15] Nowicki E, Smutnicki C. A fast tabu search algorithm for the permutation flow-shop problem. *European Journal of Operational Research* 1996;91:160–75.
- [16] Grabowski J, Pempera J. New block properties for the permutation flow-shop problem with application in TS. *Journal of the Operational Research Society* 2001;52:210–20.
- [17] Ben-Daya M, Al-Fawzan M. A tabu search approach for the flow shop scheduling problem. *European Journal of Operational Research* 1998;109:88–95.
- [18] Grabowski J, Wodecki M. A very fast tabu search algorithm for the job shop problem. In: Rego C, Alidaee B, editors. *Metaheuristic optimization via memory and evolution; tabu search and scatter search*. Dordrecht: Kluwer Academic Publishers; 2005. p. 115–44.
- [19] Smutnicki C. Some properties of scheduling problem with storing constraints. *Zeszyty Naukowe AGH: Automatyka* 1983;34:223–32 [in Polish].
- [20] Glover F. Tabu search. Part I. *ORSA Journal of Computing* 1989;1:190–206.
- [21] Glover F. Tabu search. Part II. *ORSA Journal of Computing* 1990;2:4–32.
- [22] Grabowski J, Wodecki M. A very fast tabu search algorithm for the flow shop problem with makespan criterion. *Computers and Operations Research* 2004;11:1891–909.
- [23] Taillard E. Benchmarks for basic scheduling problems. *European Journal of Operational Research* 1993;64:278–85.
- [24] OR LIBRARY. <http://mscmga.ms.ic.ac.uk/info.html>