A FAST TABU SEARCH ALGORITHM FOR FLOW SHOP PROBLEM WITH BLOCKING

Jozef Grabowski, Jaroslaw Pempera
Wroclaw University of Technology, Institute of Engineering Cybernetics
Janiszewskiego 11-17, 50-372 Wroclaw, Poland

Keywords: Flow-shop scheduling; Blocking; Makespan; Local search; Tabu Search.

Abstract: This paper develops a fast tabu search algorithm to minimize makespan in a flow shop problem with blocking. We present a fast heuristic algorithm based on tabu search approach. In the algorithm the multimoves are used that consist in performing several moves simultaneously in a single iteration of algorithm 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, in the algorithm a dynamic tabu list is used that assists additionally to avoid being trapped at a local optimum. The proposed algorithm is empirically evaluated and found to be relatively more effective in finding better solutions in a much shorter time.

1 INTRODUCTION

This paper considers the blocking flow shop problem to minimize makespan. The classical flow shop problem disregards the behavior 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 a flow shop problem with blocking, there are no buffers between the machines and a job, having completed processing on a machine, remains on this machine and *blocks* it, until the next machine downstream becomes available for processing.

The flow shop problem with blocking has been studied by many researchers including (Abadi et al., 2000), (Caraffa et al., 2001), (Grabowski and Pempera, 2000), (Hall and Sriskandarajah, 1996), (Leistein, 1990), (McCormick et al., 1989), (Nowicki, 1999), (Peng et al., 2004), (Reddi and Ramamoorthy, 1972), (Ronconi, 2004), (Ronconi and Armentano, 2001), (Smutnicki, 1983).

In this paper, we propose a new heuristic algorithm based on tabu search technique. The paper is organized as follows. In Section 2, the problem is defined and formulated. Section 3 presents the description of algorithm, moves, neighbourhood structure, search process and dynamic tabu list. Computational results are shown in Section 4 and compared with those taken from the literature.

2 PROBLEM DESCRIPTION AND PRELIMINARIES

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

PROBLEM: Each of n jobs from the set $J = \{1, 2, \ldots, n\}$ has to be processed on m machines $1, 2, \ldots, m$ in that order, having no intermediate buffers. Job $j \in J$, consists of a sequence of m operations $O_{j1}, O_{j2}, \ldots, O_{jm}$; 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}, can not leave the machine k, until the next machine $k+1$ is free, $k = 1, 2, \ldots, m-1$, $j \in J$. 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 $\pi = (\pi(1), \ldots, \pi(n))$ 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

Grabowski J. and Pempera J. (2005).
A FAST TABU SEARCH ALGORITHM FOR FLOW SHOP PROBLEM WITH BLOCKING.
In *Proceedings of the Second International Conference on Informatics in Control, Automation and Robotics*, pages 71-76
DOI: 10.5220/0001160900710076

by π. It is known from literature (Ronconi, 2004) that $C_{\max}(\pi) = D_{\pi(n)m}$, where the departure time $D_{\pi(j)k}$ of job $\pi(j)$ on machine k can be found using the following recursive expressions:

$$D_{\pi(1)0} = 0, \qquad k = 1, \ldots, m-1,$$

$$D_{\pi(1)k} = \textstyle\sum_{l=1}^{k} p_{\pi(1)l}, \qquad k = 1, \ldots, m-1,$$

$$D_{\pi(j)0} = D_{\pi(j-1),1}, \qquad j = 2, \ldots, n,$$

$$D_{\pi(j)k} = \max\{D_{\pi(j)k-1} + p_{\pi(j)k}, D_{\pi(j-1),k+1}\},$$
$$j = 2, \ldots, n, \quad k = 1, \ldots, m-1,$$

$$D_{\pi(j)m} = D_{\pi(j),m-1} + p_{\pi(j)m}, \qquad j = 1, \ldots, n.$$

The two-machine flow shop problem with blocking can be solved in $O(n\log n)$ time (Reddi and Ramamoorthy, 1972) using an improved implementation by (Gilmore et al., 1985) of the algorithm of (Gilmore and Gomory, 1964). For $m > 2$, the problem is unfortunately NP-hard.

It is useful to present the considered problem by using a graph (Grabowski and Pempera, 2000). For the given processing order π, we create the 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:

- $N = \{1, ..., n\} \times \{1, ..., m\}$, 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.
- $F^0(\pi) = \bigcup_{j=1}^{n-1} \bigcup_{k=1}^{m} \{((\pi(j), k), (\pi(j+1), k))\}$. 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.
- $F^-(\pi) = \bigcup_{j=1}^{n-1} \bigcup_{k=1}^{m-1} \{((\pi(j), k+1), (\pi(j+1), k)\}$. 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 $G(\pi)$.

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 $u = (u_1, u_2, \ldots, u_w)$, where $u_i = (j_i, k_i) \in N$, $1 \le i \le w$ and w is the number of nodes in this path. Obviously, it has to be $u_1 = (j_1, k_1) = (1, 1)$, and $u_w = (j_w, k_w) = (n, m)$. 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 to $F^-(\pi)$.

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

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

which is called the *block* of jobs in π. 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 $(u_s, ..., u_t) = ((j_s, k_s), \ldots, (j_t, k_t)))$ of u such that $((j_i, k_i), (j_{i+1}, k_{i+1})) \in F^-(\pi)$ for all $i = s, \ldots, t-1$, $s < t$. Each such subpath determines the sequence of jobs

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

which is called the *antiblock* of jobs in π. 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 operation $O_{\pi(j_i)k_i}$ of job$j_{\pi(j_i)}$ is processed on machine k_i with $k_i > k_{i+1} = k_i - 1$, $i = s, \ldots, t-1$. Similarly to in B_{gh}, now we define the *internal antiblock* of A_{st} as the subsequence

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

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 $(g_i, h_i), i = 1, \ldots, l_B$ (or $(s_i, t_i), i = 1, \ldots, l_A$), and let $GH = \{(g_i, h_i) \mid i = 1, \ldots, l_B\}$ (or $ST = \{(s_i, t_i) \mid i = 1, \ldots, l_A\}$) be the set of the pairs denoting all blocks (or antiblocks) in u.

Property 1 (Grabowski and Pempera, 2000).
For any block B_{gh} (or antiblock A_{st}) in π, let α be a processing order obtained from π by an interchange of jobs from the internal block B^*_{gh} (or A^*_{st}). Then we have $C_{max}(\alpha) \ge C_{max}(\pi)$.

Immediately from Property 1, 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 $j \in B_{gh}$ (or $j \in A_{st}$) precedes job j_g (or j_s), for some $(g,h) \in GH$ (or $(s,t) \in ST$), or

(ii) at least one job $j \in B_{gh}$ (or $j \in A_{st}$) succeeds job j_h (or j_t) , for some $(g,h) \in GH$ (or $(s,t) \in ST$).

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

3 ALGORITHM TABU SEARCH WITH MULTIMOVE (TS+M)

Currently, tabu search (TS) approach, see (Glover, 1989) and (Glover, 1990), 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. Since TS has not been applied to our problem, we propose a TS approach. The main idea of this method involves starting from an initial basic job permutation and searching through its neighbourhood, a set of permutations generated by the moves, for a permutation with the lowest makespan. The search then is repeated starting from the best permutation, as a new basic permutation, and the process is continued. One of the main ideas of TS is the use of a tabu list to avoid cycling, overcoming local optimum, 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. Usually, the algorithm TS terminates when it 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.

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. The intuition following from Theorem 1 suggests that the "insert" type move should be the most proper one for the problem considered. Let $v = (\pi(x), \pi(y))$ be a pair of jobs in a permutation π, $x, y \in \{1, 2, ..., n\}, x \neq y$. 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 $x > y$).

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

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 antibloks. Let us take the block $B_{gh} = (j_g, j_{g+1}, \dots, j_{h-1}, j_h))$ in π. Now we define the sets of $candidates$

$$E^a_{gh} = \{j_g, j_{g+1}, \dots, j_{h-1}\} = B_{gh} - \{j_h\},$$
$$E^b_{gh} = \{j_{g+1}, \dots, j_{h-1}, J_h)\} = B_{gh} - \{j_g\}.$$

Each set E^a_{gh} (or E^b_{gh}) contains the jobs in block B_{gh} of π that are candidates for being moved to a position *after* (or *before*) all other jobs in this block.

For any block B_{gh} in π, we define the following set of moves to the right

$$RB_{gh} = \{(j, j_h) \mid j \in E^a_{gh}\},$$

and the set to the left

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

Set RB_{gh} contains all moves of jobs of E^a_{gh} 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^b_{gh} 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 $C_{max}(\pi_v) < C_{max}(\pi)$. As a consequence, in algorithm, 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 TS+M, we propose the following set of moves

$$M = MB \cup MA,$$

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

In order to accelerate the convergence of the algorithms to good solutions, we propose to use 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 our TS algorithm 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 Section *Moves and neighbourhood*).

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

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

Let

$$\begin{aligned} BB^{(-)} = &\quad \{v \in BB \mid C_{max}(\pi_v) < C_{max}(\pi)\} = \\ = &\quad \{v_1, v_2, ..., v_p\},\ p \leq 2l_B. \end{aligned}$$

be the set of the *profitable moves* of blocks B_{gh} performing 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*

$$\begin{aligned} BA^{(-)} = &\quad \{v \in BA \mid C_{max}(\pi_v) < C_{max}(\pi)\} = \\ = &\quad \{v_1, v_2, ..., v_q\}, q \leq 2l_A. \end{aligned}$$

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

$$\begin{aligned} BM^{(-)} = &\quad BB^{(-)} \cup BA^{(-)} = \{v_1,\ v_2, ..., v_z\}, \\ &\quad z \leq 2(l_B + l_A). \end{aligned}$$

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 $\pi_{\overline{v}}$, where $\overline{v} = BM^{(-)}$. To simplify, in the further considerations, multimove $BM^{(-)}$ will be denoted alternatively by $\overline{v}$.

3.2 Search process

TS+M starts from an initial basic permutation π that implies the 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. 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, i.e. $C_{max}(\pi_{v^*}) = \min_{v \in UM} C_{max}(\pi_v)$, is chosen for the search.

If the move v^* is selected, then a pair of jobs corresponding to the move v^* is added to the tabu list (see next section *Tabu list and tabu status of move* 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.

We develop our algorithm tabu search by using the multimoves in some specific situations.If permutation $\pi_{\overline{v}}$ is obtained by performing a multimove $\overline{v}$, then the next one is made when *Piter* of the consecutive non-improving 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.

If a multimove is performed, then a pair of jobs corresponding to the move $v^* \in \overline{v}$ with the smallest value of $C_{max}(\pi_{v^*})$ is added to tabu list T. The *Piter* is a tuning parameter which is to be chosen experimentally.

3.3 Tabu list and tabu status of move

In our algorithm we use the cyclic tabu list defined as a finite list (set) *T* with length $LengthT$ containing ordered pairs of jobs. The list *T* is a realization of the short-term search memory. If a multimove $\overline{v}$ is performed on permutation π, then, for the

”best” move $v^* = (\pi(x), \pi(y)) \in \overline{v}$, 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 $A(\pi(x)) \cap \{\pi(x+1), \pi(x+2), ..., \pi(y)\} \neq \emptyset$ if $x < y$, and $B(\pi(x)) \cap \{\pi(y), \pi(y+1), ..., \pi(x-1)\} \neq \emptyset$ 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 algorithm uses a tabu list with dynamic length. This length is changed cyclically, as the current iteration number $iter$ of TS+M increases, using the “pick“ in order to avoid being trapped at a local optimum (see (Grabowski and Wodecki, 2004) for details). This kind of tabu list can be viewed as a specific disturbance and was employed on that very fast tabu search algorithm proposed by Grabowski and Wodecki (Grabowski and Wodecki, 2004), where it was successfully applied to the classical flow shop problem.

4 COMPUTATIONAL RESULTS

In this section we report the results of empirical tests to evaluate the relative effectiveness of the proposed tabu search algorithm. So far the best heuristic algorithms for a permutation flow-shop problem with blocking were presented in papers by (Abadi et al., 2000), (Caraffa et al., 2001), (Leistein, 1990), (McCormick et al., 1989), (Nowicki, 1999), (Ronconi, 2004). It is showed that the best heuristic algorithm is that proposed by Nowicki (Nowicki, 1999), denoted here as TSN. Therefore, we compare our algorithm TS+M with TSN which is also based on the tabu search approach.

Both algorithms TS+M and TSN were coded in C++, run on a PC with Pentium IV 1000MHz processor and tested on the benchmark instances provided by (Taillard, 1993) 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.

The algorithms based on the tabu search method, needs an initial permutation, which can found by any method. In our tests, we use algorithm NEH (Nawaz et al., 1983) in its original version, which is considered to be the best one (champion) among simple constructive heuristics for flow-shop scheduling. In our tests, for each instance, TS+M algorithm is terminated after performing *Maxiter* = 1 000, 3 000 and 30 000 iterations, whereas TNS performed *Maxiter* = 30 000 iterations. The value of tuning parameter *Piter* is drawn from (Grabowski and Wodecki, 2004) equal to 3. The effectiveness of our algorithms was analyzed in both terms of CPU time and solution quality.

For each test instance, we collected the following values:

- $PRI = 100\%(C^{NEH} - C^A)/C^{NEH}$ – the value of the percentage relative improvements of the makespan C^A obtained by algorithm $A = \{TSN, TS+M\}$ with respect to the makespan C^{NEH} obtained by algorithm NEH.
- CPU – the computer time (in seconds).

Then, for each size (group) $n \times m$, the following measures of the heuristic quality were calculated

- $APRI$ – the average (for 10 instances) percentage relative improvements of the makespans.
- $ACPU$ – the average (for 10 instances) computer time.

The computational results presented in Table 1 show that, in terms of APRI values, our algorithm TS+M, for $Maxiter$ =30 000 iterations, performs significantly better than TSN in comparable the CPU times. The TS+M found makespans with the overall average APRI equal to 3.05, whereas TSN found the ones with 2.21. The superiority of TS+M over TSN increases with the size of instances. And so, for the largest instances with $n \times m = 500 \times 20$, the TS+M found makespans with average APRI equal to 1.12,

Table 1: Computational results

	TSN		TS+M					
$Maxiter$	30000		1000		3000		30000	
$n \times m$	APRI	ACPU	APRI	ACPU	APRI	ACPU	APRI	ACPU
20×5	2.93	2.4	2.87	0.1	3.08	0.3	4.08	2.5
20×10	4.51	4.1	3.27	0.1	4.11	0.4	4.75	4.4
20×20	2.83	7.2	2.33	0.3	2.39	0.7	2.89	7.3
50×5	1.69	6.0	2.04	0.2	2.34	0.6	3.05	6.1
50×10	3.13	10.8	2.63	0.4	3.18	1.0	4.04	10.9
50×20	3.70	19.1	2.01	0.7	2.47	1.9	4.42	20.1
100×5	0.79	12.3	0.98	0.4	1.18	1.3	1.78	12.8
100×10	1.98	21.9	1.78	0.7	2.06	1.8	3.00	23.1
100×20	2.56	39.5	1.76	1.3	2.12	2.2	3.04	40.9
200×10	0.73	44.1	1.03	1.5	1.28	4.0	1.93	46.3
200×20	1.35	79.4	1.30	2.7	1.68	8.0	2.52	82.1
500×20	0.36	213	0.49	7.0	0.60	20.0	1.12	205
all	2.21		1.87		2.21		3.05	

whereas TSN found the ones with APRI = 0.36. For the instances with $n \times m = 200 \times 20$, respective values are equal to 2.52 and 1.35.

Analysing the performance of our algorithm, we have observed that TS+M converges to the good solutions significantly faster, on the average, than TSN. The APRI values for all instances equal to 2.21 was found by TSN in 30 000 iterations whereas for TS+M the respective number is 3 000 iterations. Furthermore, about 85 % improvements obtained by TS+M (i.e. APRI = 1,87 - for all instances) has been reached in 1 000 iterations.

Generally speaking TS+M produces better results than TSN in a significantly shorter time.

ACKNOWLEDGEMENTS

This research has been supported by KBN Grant 4 T11A 016 24. The authors are due to anonymous referees for their useful comments and suggestions.

REFERENCES

Abadi, I. N. K., Hall, N., and Sriskandarayah, C. (2000). Minimizing cycle time in a blocking flowshop. *Operations Research*, 48:177–180.

Caraffa, V., Ianes, S., Bagchi, T., and Sriskandarayah, C. (2001). Minimizing makespan in a blocking flowshop using genetic algorithms. *International Journal of Production Economics*, 70:101–115.

Gilmore, P. and Gomory, R. (1964). Sequencing a state-variable machine: a solvable case of the traveling salesman problem. *Operations Research*, 12:655–679.

Gilmore, P., Lawler, E., and Shmoys, D. (1985). Well-solved special cases. *E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys (Eds), The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization. Wiley: Chichester*, pages 87–143.

Glover, F. (1989). Tabu search. part i. *ORSA Journal of Computing*, 1:190–206.

Glover, F. (1990). Tabu search. part ii. *ORSA Journal of Computing*, 2:4–32.

Grabowski, J. and Pempera, J. (2000). Sequencing of jobs in some production system. *European Journal of Operational Research*, 125:535–550.

Grabowski, J. and Wodecki, M. (2004). A very fast tabu search algorithm for the flow shop problem with makespan criterion. *Computers and Operations Research*, 11:1891–1909.

Hall, N. and Sriskandarajah, C. (1996). A survey of machine scheduling problems with blocking and no-wait in process. *Operations Research*, 44:510–525.

Leistein, R. (1990). Flowshop sequencing with limited buffer storage. *International Journal of Production Research*, 28:2085–2100.

McCormick, M., Pinedo, M., Shenker, S., and Wolf, B. (1989). Sequencing in an assembly line with blocking to minimize cycle time. *Operations Research*, 37:925–935.

Nawaz, M., Enscore, E., and Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flowshop sequencing problem. *OMEGA The International Journal of Management Science*, 11:91–95.

Nowicki, E. (1999). The permutation flow shop with buffers: A tabu search approach. *European Journal of Operational Research*, 116:205–219.

Peng, Y., Soong, B., and Wang, L. (2004). *Electronics Letters*, 40:375–376.

Reddi, S. and Ramamoorthy, C. (1972). On flowshop sequencing problems with no-wait in process. *Operational Research Quarterly*, 23:323–331.

Ronconi, D. (2004). A note on constructive heuristics for the flowshop problem with blocking. *International Journal of Production Economics*, 87:39–48.

Ronconi, D. and Armentano, V. (2001). Lower bounding schemes for flowshops with blocking in-process. *Journal of the Operational Research Society*, 52:1289–1297.

Smutnicki, C. (1983). Some properties of scheduling problem with storing constraints. *Zeszyty Naukowe AGH: Automatyka (in Polish)*, 34:223–232.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64:278–285.