Journal of the Operational Research Society (2001) 52, 210–220
www.palgrave.com/jors

New block properties for the permutation flow shop problem with application in tabu search

J Grabowski* and J Pempera
Wroclaw University of Technology, Wrocław, Poland

This paper deals with the classic flow-shop scheduling problem with the make-span criterion. Some new properties of the problem associated with the so-called blocks have been presented and discussed. The properties allow us to skip some non-perspective solutions during the search of the solution space. Applied to local search algorithms, they result in a significant reduction of neighbourhood size and quickly direct the search trajectory to promising regions of the solution space. The implementation of the proposed properties in a tabu search algorithm is also presented. Computational experiments (up to 500 jobs and 20 machines) are given and compared with the results yielded by the best known algorithms.

Keywords: sequencing; heuristics; tabu search

Introduction

The notorious permutation flow-shop scheduling problem can be considered as a permanent indicator of the practical applicability of scheduling theory. This problem, which is relatively simply formulated and successfully applied in industry, has a finite but large number of solutions, but it is unfortunately NP-hard. For this reason many various algorithms have been proposed and tested to solve the problem in a short time. Skipping the review of a large number of papers dealing with this problem, we only mention the most recent and representative ones.[1–8] This paper deals with properties and techniques, which allow one to solve flow-shop instances up to 500 jobs and 20 machines with high accuracy in a reasonable time.

The permutation flow-shop problem 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. Job j, $j \in J$, 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}. We want to find a schedule such that the processing order is the same on each machine and the maximum completion time is minimal.

Each schedule of jobs can be represented by a permutation $\pi = (\pi(1), \ldots, \pi(n))$ on set J. Let Π denote the set of all such permutations. We wish to find such a 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 well known that $C_{\max}(\pi) = C_{\pi(n)m}$, where completion time $C_{\pi(j)k}$ of job $\pi(j)$ on machine k can be found using the recursive formula

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

with $\pi(0) = 0$, $C_{0k} = 0$, $k = 1, \ldots, m$, $C_{j0} = 0$, $j = 1, \ldots, n$. For a more sophisticated analysis, however, we refer to the known alternative in the literature, the non-recursive expression for the value of $C_{\max}(\pi)$:

$$C_{\max}(\pi) = \max_{1 \leqslant t_1 \leqslant t_2 \leqslant \ldots \leqslant t_{m-1} \leqslant m} \left(\sum_{j=1}^{t_1} p_{\pi(j)_1} + \sum_{j=j_1}^{t_2} p_{\pi(j)2} + \ldots + \sum_{j=t_{m-1}}^{n} p_{\pi(j)m} \right). \tag{2}$$

Despite the complex structure of formula (2), $C_{\max}(\pi)$ for given π can be found in time $O(nm)$ from equation (1).

Classic block properties

Block properties (BPs) are some general properties associated with the graph representation of scheduling problems with $C_{\max}$, $L_{\max}$ and $f_{\max}$ criteria. They date back to the early 1980s.[9,10] Starting from the basic single-machine case[11] with the criterion $L_{\max}$, they were quickly extended to

*Correspondence: J Grabowski, Institute of Engineering Cybernetics, Wrocław University of Technology, Janiszewskiego 11–17, 50-372 Wrocław, Poland.
E-mail: grabow@ict.pwr.wroc.pl

cover more general scheduling problems, namely to flow-shop and job-shop problems with $L_{\max}$ and $f_{\max}$ criteria.[12–15] Since that time they have been employed in many other algorithms,[16–18] to mention but a few. Initially, due to the elimination power, they were used in branch-and-bound algorithms (B&B), implying a very characteristic branching rule.[9,10,12–16,18] Another reported application refers to the acceleration of some flow-shop heuristics.[19] In the 1990s, when B&B schemes became less popular because of its excessive complexity, the significance of these properties for modern approximate methods was appreciated. The spectacular success of TS with built-in block properties[1,20–22] proved that local search methods can provide quite good algorithms; however, just the application of specific theoretical properties may lead to excellent algorithms. BPs were then incorporated into some other excellent algorithms[17,18] and they contributed to their success. Unfortunately, all the above-mentioned papers on approximate methods refer to very simple basic versions of BPs, whereas there exist more powerful BPs, which so far have not been applied in search methods. Hence several questions can be posed: (a) what do the extended block properties look like? (b) how can they be incorporated into known solution techniques such as tabu search (TS), simulated annealing (SA), etc.? (c) is this profitable? To answer these questions we undertook our research.

Let us start from elementary definitions and properties. A sequence of integers $t = (t_1, t_2, \ldots, t_{m-1})$, satisfying $1 \leqslant t_1 \leqslant t_2 \leqslant, \ldots, \leqslant t_{m-1} \leqslant n$, defines a *path* in π. The *length* $C(\pi, t)$ of this path is defined by

$$C(\pi, t) = \sum_{j=1}^{t_1} p_{\pi(j)1} + \sum_{j=t_1}^{t_2} p_{\pi(j)2} + \cdots + \sum_{j=t_{m-1}}^{n} p_{\pi(j)m}.$$

It follows from (2) that, for any π, we have

$$C_{\max}(\pi) \geqslant C(\pi, t) \tag{3}$$

where $t = (t_1, t_2, \ldots, t_{m-1})$ is any path in π. Path $u = (u_1, u_2, \ldots, u_{m-1})$ such that

$$C_{\max}(\pi) = C(\pi, u) = \sum_{j=1}^{u_1} p_{\pi(j)1} + \sum_{j=u_1}^{u_2} p_{\pi(j)2} + \cdots + \sum_{j=u_{m-1}}^{n} p_{\pi(j)m} \tag{4}$$

is called a *critical path* with respect to π. A sequence of jobs

$$B_k = (\pi(u_{k-1}), \pi(u_{k-1}+1), \ldots, \pi(u_k))$$

is called the kth *block* in π, $k = 1, 2, \ldots, m$, where $u_0 \stackrel{\text{def}}{=} 1$ and $u_m \stackrel{\text{def}}{=} n$. Jobs $\pi(u_k - 1)$ and $\pi(u_k)$ in B_k of π are the *first* and *last* ones, respectively. The locations of jobs in B_k are defined by the sequence of appropriate indices

$$P_k = (u_{k-1}, u_{k-1}+1, \ldots, u_k).$$

Next, we define the kth *internal block* of B_k as the subsequence

$$B_k^* = \begin{cases} B_k \backslash \{\pi(u_1)\} & \text{if } k = 1 \\ B_k \backslash \{\pi(u_{k-1}), \pi(u_k)\} & \text{if } 1 < k < m \\ B_k \backslash \{\pi(u_{m-1})\} & \text{if } k = m \end{cases}.$$

By analogy, let P_k^* be a suitable sequence of positions of jobs located in B_k^*. To simplify, the *sets* of elements from the sequences B_k, B_k^*, P_k, P_k^* we will also denote by B_k, B_k^*, etc. Although the notions B_k, B_k^*, P_k, P_k^* depend on π, we do not express this fact explicitly for the sake of simplicity of notation. By definition, the last job in B_k is simultaneously the first in B_{k+1}, $k = 1, 2, \ldots, m-1$. This implies that the set of all job indices $N = \{1, 2, \ldots, n\}$ can be presented as follows:

$$N = \bigcup_{k=1}^{m} (\{u_{k-1}\} \cup P_k^*) \tag{5}$$

or

$$N = \bigcup_{k=1}^{m} (P_k^* \cup \{u_k\}). \tag{6}$$

Using the notions introduced, we can present each permutation $\pi \in \Pi$ in the form shown in Figure 1.

Let us consider the process of modifying a given permutation π in order to make it better (or closer) to the optimal permutation. This approach has been used in approximate algorithms based on local search techniques and in some B&B methods which use the complete solution for branching and updating the upper bound. For permutation-type problems a few manipulations (perturbations)—adjacent pair-wise interchange ($\mathcal{A}$), non-adjacent pair-wise interchange ($\mathcal{N}$) and insertions or shifts ($\mathcal{U}$)—are usually considered. Let $\mathcal{A}(\pi), \mathcal{N}(\pi), \mathcal{U}(\pi)$ denote all permutations that can be obtained from π using appropriate techniques. A

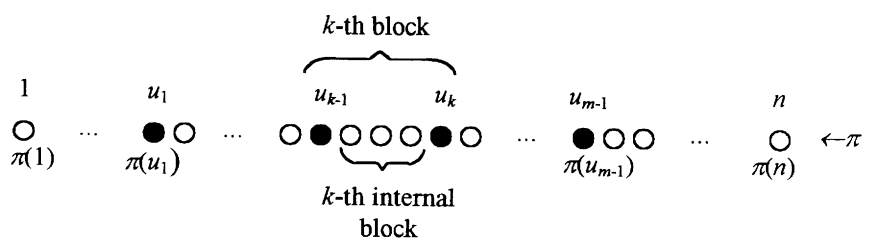

Figure 1 Flow shop permutations.

transition from π to a target permutation can be defined just by a few numbers called a *move*. In this case, a move is pair $v = (a, b)$ of positions, which indicate moved or shifted jobs. Precisely, for $\mathcal{A}(\pi)$, move (a, b), $|a - b| = 1, a, b \in \{1, 2, \ldots, n\}$, means that jobs $\pi(a)$ and $\pi(b)$ should be swapped. By analogy, for $N(\pi)$, move (a, b), $a, b \in \{1, 2, \ldots, n\}, a \neq b$, means that jobs $\pi(a)$ and $\pi(b)$ should be exchanged. For $\mathcal{U}(\pi)$, move (a, b), $a, b \in \{1, 2, \ldots, n\}, a \neq b$, means that job $\pi(a)$ should be removed from its original position and inserted so that b will be its new position in the resulting permutation. Let us denote by $A(\pi), N(\pi), U(\pi)$ the move sets needed for getting permutations from sets $\mathcal{A}(\pi), \mathcal{N}(\pi), \mathcal{U}(\pi)$, respectively. One can say that there may exist *redundant moves* leading to the same permutations. For the sake of simplicity, we allow move sets $A(\pi), N(\pi), U(\pi)$ to contain such moves, although they really do not increase the size of the respective sets $\mathcal{A}(\pi), \mathcal{N}(\pi), \mathcal{U}(\pi)$. In practice, to prevent redundancy in calculations we assume that each move set is refined but immediately before its use, and afterwards it does not contain redundant moves. Let us denote by π_v the permutation obtained from π by applying move $v = (a, b) \in \{A(\pi), N(\pi), U(\pi)\}$.

The fundamental BPs are derived from the following property.

Theorem 1.[9,10] Let $\pi \in \Pi$ be any permutation with blocks $B_k, k = 1, 2, \ldots, m.$. If the permutation β has been obtained from π and $C_{\max}(\beta) < C_{\max}(\pi)$, then in β

(i) at least one job $j \in B_k$ precedes job $\pi(u_{k-1})$, for some $k \in \{2, 3, \ldots, m\}$, or
(ii) at least one job $j \in B_k$ succeeds job $\pi(u_k)$, for some $k \in \{1, 2, \ldots, m-1\}$.

Negation of Theorem 1 leads immediately to the following corollary for adjacent and non-adjacent interchanges:

Corollary 1. *We have* $C_{\max}(\pi_v) \geqslant C_{\max}(\pi)$ *for* $v \in G(\pi) \stackrel{\text{def}}{=} \bigcup_{k=1}^{m} (P_k^* \times P_k^*) \cap N(\pi)$ and the next corollary for inserts:

Corollary 2. *We have* $C_{\max}(\pi_v) \geqslant C_{\max}(\pi)$ *for* $v \in W(\pi) \stackrel{\text{def}}{=} \bigcup_{k=1}^{m} (P_k^* \times P_k^*) \cap U(\pi)$.

Up to now, nobody has used the former corollary since the moves of type $\mathcal{A}$ and $\mathcal{N}$ are less popular for the problems with the $C_{\max}$ criterion. Thus, its practical application is still an open question. Let us concentrate next on the moves of type $\mathcal{U}$. The latter corollary states that the moves $V(\pi) \stackrel{\text{def}}{=} \mathcal{U}(\pi) \backslash W(\pi)$ are 'more interesting' than those in $W(\pi)$, taking into account the possibility of an immediate improvement of $C_{\max}(\pi)$. This is one of the fundamental observations to the tabu search algorithm proposed and developed in reference 1. The second fundamental fact used there is based on the experimental conclusion that the set of potentially *profitable* moves $V(\pi)$ remains still too large considering the high search cost. Therefore, two reduction methods were proposed in reference 1. The first (discussed in detail below) allows us to decompose the set of moves and to restrict arbitrarily its size using a control parameter ϵ. The second, exploiting some theoretical properties, skilfully aggregates the calculation process, which speeds up finding of $C_{\max}(\pi_v)$ for a collection of moves, but not for a separate move.

Move set $U(\pi)$ and its descendants are decomposed using the set of moves to the left that insert job $\pi(a)$ into positions $x \leqslant b \leqslant y, y < a$,

$$L_a(x, y) = \{(a, b) : x \leqslant b \leqslant y\} \tag{7}$$

and the 'symmetrical' set of moves to the right, $a < x$,

$$R_a(x, y) = \{(a, b) : x \leqslant b \leqslant y\} \tag{8}$$

defined for each job $\pi(a), a \in N$. To prevent a misunderstanding, we supplement this definition putting $L_a(x, y) = \emptyset$ if only $a \leqslant x$ or $x > y$ and $R_a(x, y) = \emptyset$ if only $a \geqslant x$ or $x > y$. The notations introduced enable natural decomposition of the whole set of moves into

$$U(\pi) = \bigcup_{a \in N} L_a(1, a-1) \cup \bigcup_{a \in N} R_a(a+1, n) \tag{9}$$

which, in turn, applying (5) and (6), may be re-written in the following form:

$$U(\pi) = \bigcup_{k=1}^{m} \bigcup_{a \in P_k^* \cup \{u_k\}} L_a(1, a-1) \cup \bigcup_{k=1}^{m} \bigcup_{a \in \{u_{k-1}\} \cup P_k^*} R_a(a+1, n). \tag{10}$$

Next, the set $W(\pi)$ can be presented as follows:

$$W(\pi) = \bigcup_{k=2}^{m} \bigcup_{a \in P_k^*} L_a(u_{k-1}+1, a-1) \cup \bigcup_{a \in P_1^*} L_a(1, a-1) \cup \bigcup_{k=1}^{m-1} \bigcup_{a \in P_k^*} R_a(a+1, u_k-1) \cup \bigcup_{a \in P_m^*} R_a(a+1, n). \tag{11}$$

Indeed, set $\bigcup_{a \in P_k^*} L_a(u_{k-1}+1, a-1)$ corresponds to the moves to the left, made inside the kth block $(k > 1)$, whereas $\bigcup_{a \in P_1^*} L_a(1, a-1)$ corresponds to those inside the first block. By symmetry, set $\bigcup_{a \in P_k^*} R_a(a+1, u_k-1)$ corresponds to the moves to the right inside the kth block $(k < m)$, whereas $\bigcup_{a \in P_m^*} R_a(a+1, n)$ corresponds to those inside the last

block. It is easy to verify that subtracting (11) from (10) we obtain

$$V(\pi) = \bigcup_{k=2}^{m} \bigcup_{a \in P_k^*} L_a(1, u_{k-1}) \cup \bigcup_{k=1}^{m} L_{u_k}(1, u_k - 1) \cup \bigcup_{k=1}^{m-1} \bigcup_{a \in P_k^*} R_a(u_k, n) \cup \bigcup_{k=1}^{m} R_{u_k}(u_k + 1, n). \quad (12)$$

The moves to the left: $\bigcup_{a \in P_k^*} L_a(1, u_{k-1})$ insert job $\pi(a)$ from the inside of the kth block into any position before this block. Moves $L_{u_k}(1, u_k - 1)$ insert job $\pi(u_k)$ into any other position to the left. By symmetry, the moves to the right: $\bigcup_{a \in P_k^*} R_a(u_k, n)$ insert job $\pi(a)$ from the inside of the kth block into any position after this block. Similarly moves $R_{u_k}(u_k + 1, n)$ insert job $\pi(u_k)$ into any other position to the right.

Since set $V(\pi)$ is too large, therefore, for practical application a method which limits the size of this set was proposed in ref. 1. A scheme of its reduction is shown in Table 1 where $H_\epsilon(\pi)$ denotes the resulting set of moves. Non-decreasing, non-negative functions $c(\epsilon)$ and $d(\epsilon)$ allow the user to control the size of $H_\epsilon(\pi)$, and their particular forms are presented in reference 1. Functions $c(\epsilon)$ and $d(\epsilon)$ quickly decrease, which prevents a high calculation cost—in practice they are set to zero for ratio $n/m \geqslant 3$. Note that the reduction technique is purely arbitrary in the first stage (the selection of the moves considered) and fully democratic in the second stage (the choice between the moves selected).

Let us denote by $H_\infty(\pi)$ the move set containing both sets $L_a(1, u_{k-1})$, $R_a(u_k, n)$ which are extremely large (original) and sets $L_{u_k}(1, u_k - 1)$ and $R_{u_{k-1}}(u_{k-1} + 1, n)$ but reduced as shown in Table 1. It is clear that

$$\mathcal{H}_0(\pi) \subseteq \mathcal{H}_\epsilon(\pi) \subseteq \mathcal{H}_\infty(\pi) \subseteq \mathcal{V}(\pi) \subseteq \mathcal{U}(\pi). \quad (13)$$

It is recommended to apply $\mathcal{H}_0(\pi)$ for the fastest method, whereas $\mathcal{H}_\infty(\pi)$ for the best accuracy.[15]

New block properties

In order to introduce the approach proposed, we start with (12) and apply another decomposition oriented towards new BPs. First, using (5) and (6), we decompose some sets necessary for (12), namely

$$\{1, \ldots, u_{k-1}\} = \{1, \ldots, u_1\} \cup \bigcup_{l=2}^{k-1} \{u_{l-1} + 1, \ldots, u_l\} \quad (14)$$

and

$$\{u_k, \ldots, n\} = \bigcup_{l=k+1}^{m-1} \{u_{l-1}, \ldots, u_l - 1\} \cup \{u_{m-1}, \ldots, n\}. \quad (15)$$

Then, component sets in (12) can be rewritten. So, for $a \in P_k^*$ we get from (14)

$$L_a(1, u_{k-1}) = L_a(1, u_1) \cup \bigcup_{l=2}^{k-1} L_a(u_{l-1} + 1, u_l) \quad (16)$$

whereas from (15) we get

$$R_a(u_k, n) = \bigcup_{l=k+1}^{m-1} R_a(u_{l-1}, u_l - 1) \cup R_a(u_{m-1}, n). \quad (17)$$

Substituting (16) and (17) into (12) we conclude that $V(\pi)$ is fundamentally formed by the above two elementary types of move sets. The first contains moves of jobs from the inside of the kth block to the left into the inside of the lth block, ie $L_a(u_{l-1} + 1, u_l), a \in P_k^*, l < k$. The second contains moves of jobs from the inside of the kth block to the right into the inside of the lth block, ie $R_a(u_{l-1}, u_l - 1), a \in P_k^*, l > k$. The moves performed to the right and left are shown in Figure 2.

In order to evaluate these moves, we introduce some new BPs. At first, we define a local evaluation

$$\Delta_{kl}(j) \stackrel{\text{def}}{=} P_{jl} - P_{jk},\ j \in J, k, l = 1, \ldots, m, l \neq k$$

which can be found for fixed k, l, j in time $O(1)$. Now we are able to formulate the next theoretical property.

Theorem 2. *Let $\pi \in \Pi$ be any permutation with blocks $B_k, k = 1, 2, \ldots, m$. Let π_v be the permutation obtained from π by a move $v = (a, b)$ from either $L_a(u_{l-1} + 1, u_l)$, $a \in P_k^*$, $l < k$ or $R_a(u_{l-1}, u_l - 1), a \in P_k^*, l > k$. Then we have*

$$C_{\max}(\pi_v) \geqslant C_{\max}(\pi) + \Delta_{kl}(\pi(a)). \quad (18)$$

The proof of the theorem is given in the appendix.

Theorem 2 allows us to detect some 'additional' non-promising moves, which can be then eliminated. These are moves from sets

$$L_a^+(u_{l-1} + 1, u_l) = \{(a, b) \in L_a(u_{l-1} + 1, u_l) : \Delta_{kl}(\pi(a)) \geqslant 0\} \quad (19)$$

$$R_a^+(u_{l-1}, u_l - 1) = \{(a, b) \in R_a(u_{l-1}, u_l - 1) : \Delta_{kl}(\pi(a)) \geqslant 0\}. \quad (20)$$

Table 1 Size reduction of component move sets from $V(\pi)$ to $H_\epsilon(\pi)$

Original set	*Reduced set*	*Comment*
$L_a(1, u_{k-1})$	$L_a(u_{k-1} - c(\epsilon), u_{k-1})$	Controlled by ϵ
$L_{u_k}(1, u_k - 1)$	$L_{u_k}(u_{k-1} - c(\epsilon), u_{k-1})$	Cut and controlled by ϵ
$R_a(u_k, n)$	$R_a(u_k, u_k + d(\epsilon))$	Controlled by ϵ
$R_{u_{k-1}}(u_{k-1} + 1, n)$	$R_{u_{k-1}}(u_k, u_k + d(\epsilon))$	Cut and controlled by ϵ

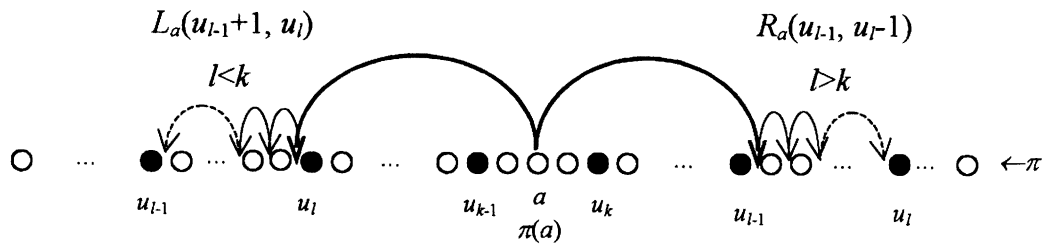

Figure 2 Job movements.

Directly from the definition and Theorem 2 we obtain

Corollary 3. $C_{\max}(\pi_v) \geqslant C_{\max}(\pi)$ for any $v = (a, b) \in L_a^+(u_{l-1} + 1, u_l)$, $a \in P_k^*$, $l < k$.

Corollary 4. $C_{\max}(\pi_v) \geqslant C_{\max}(\pi)$ for any $v = (a, b) \in R_a^+(u_{l-1}, u_l - 1)$, $a \in P_k^*$, $l > k$.

The sets of promising moves can be found by the contradiction in (19) and (20), namely

$$L_a^*(u_{l-1} + 1, u_l) = \{(a, b) \in L_a(u_{l-1} + 1, u_l) : \Delta_{kl}(\pi(a)) < 0\} \tag{21}$$

$$R_a^*(u_{l-1}, u_l - 1) = \{(a, b) \in R_a(u_{l-1}, u_l - 1) : \Delta_{kl}(\pi(a)) < 0\}. \tag{22}$$

This contradiction fulfils the necessary condition of obtaining a 'better' permutation than π. In this context, we propose the 'intelligently' reduced move set $M(\pi)$ based on the scheme shown in Table 2. We found this move set very promising for the search.

The extension of the idea proposed suggests that 'more promising' moves should have some preferences in the choice. The intuition following from Theorem 2 states that moves with the smallest value of $\Delta_{kl}(\pi(a))$ should be considered first. Then, for the fixed $a \in P_k^*$ and moves to the left $(a, b) \in L_a(1, u_1) \cup \bigcup_{l=2}^{k-1} L_a^*(u_{l-1} + 1, u_l)$, let $l(a)$ be the index of block l that provides the minimal value of $\Delta_{kl}(\pi(a))$. By symmetry, for the fixed $a \in P_k^*$ and moves to the right $(a, b) \in \bigcup_{l=k+1}^{m-1} R_a(u_{l-1}, u_l - 1) \cup R_a(u_{m-1}, n)$, let $r(a)$ be the index of block l that provides the minimal value of $\Delta_{kl}(\pi(a))$. The last move set: $M^*(\pi)$ is obtained from $M(\pi)$ preserving only the part of $M(\pi)$, which is most promising for the immediate improvement of the $C_{\max}(\pi_v)$. This is done according to the scheme shown in Table 3.

Table 2 Size reduction of component move sets from $V(\pi)$ to $M(\pi)$

Original set	*Reduced set*	*Comment*
$L_a(u_{l-1} + 1, u_l)$	$L_a^*(u_{l-1} + 1, u_l)$	Promising only
$L_{u_k}(1, u_k - 1)$	$L_{u_k}(u_{k-1}, u_{k-1})$	Reduced to singleton
$R_a(u_{l-1}, u_l - 1)$	$R_a^*(u_{l-1}, u_l - 1)$	Promising only
$R_{u_{k-1}}(u_{k-1} + 1, n)$	$R_{u_{k-1}}(u_k, u_k)$	Reduced to singleton

It can be proved that

$$\mathcal{M}^*(\pi) \subseteq \mathcal{M}(\pi) \subseteq \mathcal{H}_\infty(\pi) \subseteq \mathcal{V}(\pi) \subseteq \mathcal{U}(\pi). \tag{23}$$

Comparing (23) with (13) we should note certain facts. First, there are no relationships between $\mathcal{H}_\epsilon(\pi)$ for $\epsilon < \infty$ and $\mathcal{M}(\pi), \mathcal{M}^*(\pi)$. Second, $\mathcal{M}(\pi)$ and $\mathcal{M}^*(\pi)$ contain some carefully selected best permutations, whereas $\mathcal{H}_\epsilon(\pi)$ contains some arbitrarily selected permutations. Third, the size of $\mathcal{M}^*(\pi)$ is approximately of the order of $|\mathcal{H}_{0.5}(\pi)|$, which is smaller than the size of $\mathcal{H}_1(\pi)$ used in TSAB for some harder instances; the size of $\mathcal{M}(\pi)$ is approximately of the order of $0.5|\mathcal{H}_\infty(\pi)|$. Fourth, the largest set $\mathcal{M}(\pi)$ does not exceed $\mathcal{H}_\infty(\pi)$ and it will reach the size of $\mathcal{H}_\infty(\pi)$ only if $L_a^-(*) = \emptyset = R_a^-(*)$ for all $a \in N$, which, in turn, seems unlikely to happen.

Implementation in tabu search

Although BPs can be applied in any local search algorithm, we consider here only their implementation in the tabu search environment, which allows us to make a comparative study.

TS is a meta-heuristic capable of handling any sequencing problem.[23,24] Roughly speaking, the basic idea of this approach consists in starting from an initial basic permutation of n jobs and searching through its neighbourhood for a permutation with the lowest makespan. Then the search is repeated starting from the best permutation, as a new basic permutation, and the process is continued. The neighbourhood of a permutation is generated by the moves. A move changes the location of some jobs in the basic permutation. A mechanism of a tabu list is used to avoid cycling and

Table 3 Size reduction of component move sets from $V(\pi)$ to $M(\pi)$

Original set	*Reduced set*	*Comment*
$\bigcup_{l=1}^{k-1} L_a(u_{l-1} + 1, u_l)$	$L_a^*(u_{l(a)-1} + 1, u_{l(a)})$	Super promising only
$L_{u_k}(1, u_k - 1)$	$L_{u_k}(u_{k-1}, u_{k-1})$	Reduced to singleton
$\bigcup_{l=k+1}^{m} R_a(u_{l-1}, u_l - 1)$	$R_a^*(u_{r(a)-1}, u_{r(a)} - 1)$	Super promising only
$R_{u_{k-1}}(u_{k-1} + 1, n)$	$R_{u_{k-1}}(u_k, u_k)$	Reduced to singleton

becoming trapped at a local optimum as well as to guide the search to a 'good region' of the solution space. The list records the moves, treating them as forbidden for future moves, ie it determines forbidden permutations in the currently analysed neighbourhood. A move having prohibited attributes is forbidden, although it can be performed if it is sufficiently profitable. The list content is refreshed each time a new basic permutation is found: the oldest element is removed and the new one is added. The search stops when a given number of iterations has been reached without improving the best current solution or the algorithm has performed a given number of iterations, or the neighbourhood is empty, etc.

Some of the components used in our algorithm, namely the form of the tabu list, the tabu status of a move, and the search strategy, were taken from, refs 1 and 20 where they had been successfully applied. In this paper, we use some components in their original forms or in the modified forms according to the approach considered.

Moves and neighbourhoods

We implement basically two types of the new neighbourhoods developed by us, namely $\mathcal{M}(\pi)$ and $\mathcal{M}^*(\pi)$ which are both based on insertion moves. The size of each neighbourhood can be additionally reduced by using an *aspiration function* described in the following. In order to prevent too high computational cost, we employ the idea for speeding up calculation presented in references 1 and 5.

Tabu list and tabu status of move

In our algorithm (given below), we use a tabu list defined as finite cyclic list T of length *LengthT* containing ordered pairs of jobs. The list is initiated by introducing *LengthT* empty elements. If move (a, b) is performed from the permutation π, then, similar to reference 1, pair of jobs $(\pi(a), \pi(a+1))$ if $a < b$, or the pair $(\pi(a-1), \pi(a))$, if $a > b$ 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, the move (a, b) has tabu status (it is forbidden) if at least one pair $(\pi(j), \pi(a))$ with $a+1 \leqslant j \leqslant b$ belongs to T if $a < b$, and at least one pair $(\pi(a), \pi(j))$ with $b \leqslant j \leqslant a-1$ is in T, otherwise.

Aspiration function

Aspiration function $AF(v)$ decides whether unforbidden move v should be considered, since it gives hope that a better permutation than the best known up to now, C^*, will be found, otherwise the move should be rejected. This allows us to eliminate also those moves v for which

$$AF(v) \geqslant C^*$$

and thus to reduce the size of the neighbourhood searched. We propose the following function

$$AF_s(v) = LB(v)/(\alpha + s\beta)$$

where $\alpha \geqslant 1$ and $\beta \geqslant 0$ are some constants determined experimentally, and s is the current iteration number. As $LB(v)$ we take the bound on $C_{\max}(\pi_v)$ formulated in Theorem 2, namely $LB(v) = C_{\max}(\pi) + \Delta_{kl}(\pi(a))$. The complexity of calculating $AF(v)$ is $O(1)$ per one move.

Search strategy

We employ a specific two-level search strategy, similar to refs 1 and 20, which yields very good computational results. Move $v \in M(\pi)$ can belong to one of three categories: (1) unforbidden (UF) if it does not have the tabu status, (2) forbidden but profitable (FP) if it has the tabu status and satisfies condition $C_{\max}(\pi_v) < C_{\max}(\pi^*)$, or (3) forbidden and non-profitatble (FN) if it has the tabu status and satisfies condition $C_{\max}(\pi_v) \geqslant C_{\max}(\pi^*)$, where $C_{\max}(\pi^*)$ is the lowest value of the makespan obtained in the search process up to the time moment when π^* is generated.

For given permutation π the neighbourhood of π is searched in the following manner. First, sets of moves are defined

$$\overline{M}(\pi) = \{v \in M(\pi) : \text{move } v \text{ is UF or FP}\}$$

Next, a family of move subsets is created:

$$C_{\max}(\pi_{v_a}) = \min_{(a,b) \in F_a} C_{\max}(\pi_{(a,b)}).$$

where $F_a = \overline{M}(\pi) \cap (L_a(1, a-1) \cup R_a(a+1, n))$ contains all moves from position a of π in neighbourhood $M(\pi)$ that are UF or FN. For each $F_a \in F$ the best move in F_{v_a}, called a *representative* of F_a, is chosen:

$$F = \{F_1, F_2, \ldots, F_n\}$$

Selected representative moves form *representative set* $BR = \{v_1, \ldots, v_n\}$. The move to be performed v^* is selected from amongst those in BR with the lowest value of the objective function, ie $C_{\max}(\pi_{v^*}) = \min_{v \in BR} C_{\max}(\pi_v)$. If the set of representatives is empty, the oldest element of tabu list T is removed and the search process is repeated.

For the algorithm which uses $\mathcal{M}^*(\pi)$, the description of the search strategy is the same.

Algorithm

In the algorithm, the asterisk (*) refers to the best values found, the zero superscript (^{0}) refers to initial value and its lack denotes current values. The algorithm starts from given initial permutation π^0 (π^0 can be found by any algorithm). In each run, similar to in TSAB,[1] the algorithm stores the last best profitable region of the search space,

enabling additional restarts (backtrack) of the search from this region if at most *Maxret* successive iterations have passed without improving the makespan. The algorithm stops when *Maxiter* iterations have been performed.

Algorithm GP

INITIALISATION.

Set $\pi^* := \pi^0$, $\pi := \pi^0$, $C^* := C_{\max}(\pi^*)$, $T := \emptyset$, $Z := \emptyset$, $iter := 0$, $ret := 0$.

SEARCHING.

Set $iter := iter + 1$, $ret := ret + 1$. For moves $v \in \overline{M}(\pi)$ satisfying the condition $AF(v) < C^*$ create set of representatives $BR = \{v_1, v_2, \ldots, v_n\}$.

SELECTION.

If $BR = \emptyset$ then remove the oldest element of the tabu list and go to SEARCHING. Find the best move $v \in BR$, ie $C_{\max}(\pi_v) = \min_{1 \leqslant a \leqslant n} C_{\max}(\pi_{v_a})$. If $C_{\max}(\pi_v) < C^*$ then store the profitable search region (ie $Z := BR \setminus \{v\}$, $\pi^0 := \pi$, $T^0 := T$) and save the best values $C^* := C_{\max}(\pi^*)$, $\pi^* := \pi_v$, $ret := 0$.

MAKE MOVE

Modify the tabu list according to the method described earlier and set $\pi := \pi_v$.

STOP CRITERIA

If $iter \geqslant Maxiter$ then STOP. If $ret < Maxret$ then go to Searching.

BACKTRACK.

Restore the profitable region, ie set $\pi := \pi^0$, $T := T^0$ and set $ret := 0$. If $Z \neq \emptyset$ then find the best move $v \in Z$, such that $C_{\max}(\pi_v) = \min_{r \in Z} C_{\max}(\pi_r)$, set $Z := Z \setminus \{v\}$ and go to MAKE MOVE. Otherwise STOP.

Algorithm GP has four tuning parameters, ie *LengthT*, α, β and *Maxret*, that are to be chosen experimentally.

Computational results

Algorithm GP was coded in C++ and run on an IBM RS6000/43P, 200 MHz, assuming two different neighbourhoods $M(\pi)$ and $M^*(\pi)$, denoted hereafter by GP+M and GP+M*, respectively.

Among the known papers dealing with approximate algorithms for the flow-shop problem with the $C_{\max}$ criterion,[1–8] so far the best speed and accuracy have been obtained by algorithm TSAB[1] based on the TS approach and the best accuracy by algorithm RY[17] based on the GA approach. Since we implement our algorithm in the TS environment, most comparisons are made with TSAB, but there are a few made with RY.

Algorithms GP+M and GP+M*, similar to in TSAB, were tested on benchmark problems from reference 5 (see also OR Library[25] for more details). 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.

Tuning

Algorithm GP needs an initial permutation, which can by found by any method. Similar to in TSAB, we set this permutation applying algorithm NEH[5] considered as the champion among simple constructive heuristics for flow-shop scheduling. The computational complexity of this heuristic is $O(n^2m)$.

At the initial stage, GP+M and GP+M* were run several times, chiefly for small-size instances because of running time, in order to find proper values of tuning parameters. These were chosen experimentally as a result of the compromise between the running time and solution quality. We set $\alpha = 1.2$ and $\beta = 0.000\,05$ in aspiration function $AF(v)$. Values of the parameters *LengthT* and *Maxret* depend on ratio n/m and for the best result they should be fitted individually to each size of instances and to each neighbourhood used. Nevertheless, we set them in a way common for all the instances as follows: $LengthT = 6 + [n/(10m)]$, where $[x]$ represents the integer of x, $Maxret = 800|200$ for $\mathcal{M}(\pi)$ and $Maxret = 600|100$ for $\mathcal{M}^*(\pi)$, where the symbol X|Y stands for $Maxret = X$ if the profitable search region is visited for the first time (the region is virgin, no backtrack to *this* region has been performed) and $Maxret = Y$ otherwise (after restoring the search in the region which has already been visited).

The last important parameter: *Maxiter* varies in the tests performed. Algorithms GP+M and GP+M* were tested under various limits on the iteration number. Algorithm TSAB was originally run in two versions: TSAB^a and TSAB^b with limits 1000 and 2000 iterations, respectively.

Performance measures

For each test problem we collected the following values:

C^H—the makespan found by algorithm H, H $\in$ {GP+M, GP+M*}

Time—CPU time in s.

Then two measures of the algorithm quality were calculated:

$PRD(\mathrm{H}) = 100\%(C^H - C^T)/C^T$—the mean value of the percentage relative difference between makespan C^H and reference makespan C^T given by algorithm T[6]

$CPU(\mathrm{H})$—the mean computer time of algorithm H.

The mean value is obtained from 10 instances of the given size.

There are some problems with the speed of computers used in the tests for TSAB and GP. In order to compare the results, we need to normalise calculation speeds with the

help of common benchmarks (see for example, references 26 and 27). Unfortunately, no clock was specified for 486DX used in reference 1, whereas only the specified parameter of 10 Mips does not fit to any 486DX model from the list. Therefore, we arbitrarily chose 486DX/33 MHz, one of the slowest models. Then, we assumed that both algorithms had been implemented using integer variables and using SPECint95 test[26] and Wintune97 database[27] we found that IBM RS6000/43P/200 MHz was 10–12 times faster than IBM PC 486DX used in reference 1, for TSAB. Finally, coefficient 11 will be used hereafter. We found the DEC Alpha 600 5/226 computer used in reference 17 for RY comparable with IBM RS6000/43/200 for SPECint95 test.

Convergence test

In this test, we observed the behaviour of PRD values in iterations for the different algorithms. The results are given in Table 4. The results prove that GP converges faster (in iterations) to good solutions than TSAB. The superiority of GP+M and GP+M* over TSAB increases with the size of instances. For $n = 500$ and $m = 20$ and *Maxiter* = 2000, GP+M and GP+M* found the makespans with PRD equal to -0.56 and -0.50, respectively, whereas TSAB found the makespans with PRD equal to -0.23. For $n = 200$ and $m = 20$, the respective values are equal to -1.00, -0.87, and -0.15. These solutions yielded by TSAB after 1000 iterations were found by GP+M in approximately 250 iterations, and by GP+M* in 450 iterations. For 2000 iterations, the respective numbers are 550 and 750 iterations. Hence it can be concluded that mean number of iterations necessary to find a good solution is approximately three times smaller for GP+M* (four times smaller for GP+M) than for TSAB.

Quality test

One can say that the number of iterations is not an appropriate measure, because the size of $\mathcal{M}(\pi)$ and $\mathcal{M}^*(\pi)$ is greater than the size of the neighbourhood employed in TSAB. In such a case, the number of iterations should be reduced due to the greater neighbourhood applied. In order to check this and to compare the algorithms, we applied the next test. As already mentioned, the size of $\mathcal{M}^*(\pi)$ is of the order of $|\mathcal{H}_{0.5}(\pi)|$ whereas the size of $\mathcal{M}(\pi)$ is of the order of $0.5\,|\mathcal{H}_{\infty}(\pi)|$. Theoretically, it is a difficult task to compare the results since the authors of TSAB use for some instances $\mathcal{H}_1(\pi)$, for others $\mathcal{H}_0(\pi)$[1] and additionally $\mathcal{H}_{\infty}(\pi)$ in some further tests.[27] Moreover, the limits of iterations for TSAB were set individually for each group of instances and varied from 3000 to 30 000 or even more. Therefore, we compare the results obtained by GP in the convergence test (GP+M on 2000 iterations) with those published for TSAB in the long-time test (TSAB^c with iterations given in ITER), see Table 5. In order to ensure a common base for comparison, we adjusted the CPU of TSAB to our computer speed (see column ACPU of TSAB). We used two relative measures: CPU FRAC—the ratio of GP+M CPU to TSAB ACPU, and PRD SUB—the subtraction TSAB PRD minus GP+M PRD. We arrive at uncertain conclusions. Most of the results were obtained by us in a time shorter than that for TSAB (the mean CPU FRAC value is 0.76). Unfortunately not all of them are better in terms of the PRD value. Excluding group 50|20, which in our opinion needs an individual comment, we obtained the mean value of CPU FRAC equal to 0.74, whereas the mean value of PRD SUB is -0.01. This means that GP+M yields slightly better results than TSAB in a relatively shorter time. Group 50|20 is commonly considered as particularly hard (see the limit of ITER and CPU for TSAB in Table 5). Unsatisfactory PRD value for GP+M can be explained, in our opinion, by two independent

Table 4 Convergence test results

	500 iterations			*1000 iterations*			*2000 iterations*		
n|m	*TSAB*	*GP+M*	*GP+M**	*TSAB*	*GP+M*	*GP+M**	*TSAB*	*GP+M*	*GP+M**
20|5	—	0.03	0.00	0.00	0.00	0.02	0.00	0.00	0.00
20|10	—	0.41	0.45	0.26	0.25	0.45	0.20	0.06	0.22
20|20	—	0.33	0.46	0.28	0.18	0.32	0.18	0.08	0.17
50|5	—	−0.01	0.01	0.00	−0.02	0.01	−0.01	−0.02	−0.02
50|10	—	−0.16	−0.05	0.11	−0.31	−0.22	0.05	−0.39	−0.34
50|20	—	0.85	1.36	0.78	0.39	1.02	0.49	0.17	0.55
100|5	—	−0.03	−0.03	−0.01	−0.03	−0.03	−0.03	−0.03	−0.03
100|10	—	−0.05	0.03	−0.03	−0.10	−0.08	−0.07	−0.21	−0.17
100|20	—	0.17	0.27	0.59	−0.35	−0.16	0.37	−0.66	−0.39
200|10	—	−0.14	−0.14	−0.16	−0.15	−0.17	−0.18	−0.15	−0.18
200|20	—	−0.41	−0.37	0.15	−0.67	−0.65	−0.15	−1.00	−0.87
500|20	—	−0.28	−0.18	−0.04	−0.45	−0.34	−0.23	−0.56	−0.50
all	—	0.06	0.15	0.16	−0.11	0.01	0.05	−0.23	−0.14

Table 5 Quality test results

	TSAB				GP+M			
n|m	*ITER*	*CPU*	*ACPU*	*PRD*	*CPU*	*PRD*	*CPU FRAC*	*PRD SUB*
20|5	3000	5	0.45	0.00	0.50	0.00	1.10	−0.00
20|10	10,000	33	3.00	0.00	1.30	0.06	0.43	0.06
20|20	15,000	75	6.82	0.00	3.60	0.08	0.53	0.08
50|5	3,000	12	1.09	−0.01	0.90	−0.02	0.83	−0.01
50|10	6,000	49	4.45	−0.41	3.70	−0.39	0.83	0.02
50|20	30,000	594	54.00	−0.36	12.50	0.17	0.23	0.53
100|5	4,000	30	2.73	−0.03	1.90	−0.03	0.70	−0.00
100|10	10,000	175	15.91	−0.22	6.20	−0.21	0.39	0.01
100|20	15,000	581	52.82	−0.66	29.00	−0.66	0.55	−0.00
200|10	4,000	131	11.91	−0.19	12.40	−0.15	1.04	0.04
200|20	10,000	887	80.64	−0.80	67.90	−1.00	0.84	−0.20
500|20	6,000	1445	131.36	−0.50	220.50	−0.56	1.68	−0.06

TSAB CPU on a PC 486DX, TSAB ACPU adjusted to IBM RS6000.

factors. One of them is the fact that GP+M was run in a time *four times shorter* than that of TSAB and only on 2000 iterations, hence the worse result is intuitively clear. Indeed, running GP+M on 8000 iterations (GP+M CPU approximately equal to TSAB ACPU) we find for group 50|20 that the GP+M PRD value was equal to −0.14, which confirms our hypothesis. The other factor follows from some observations made during the tests: for instances with low n/m, the mean length of blocks is small and the evaluations introduced by Theorem 2 become rough. This, in turn, implies an incorrect guiding of TS and other search mechanisms must be used in order to find a reasonably good result.

Therefore we incline to draw the following conclusion. Taking into account the CPU time, we can insist that the results of GP+M are roughly comparable to those obtained by TSAB with one exception: TSAB was run for a large number of iterations, which, in our opinion, shows that its initial dynamics were lost and so it has no potential for further development, whereas GP+M has been run on 2000 iterations only (still inside the initial phase), which means that the results obtained for a larger number of iterations may be better.

Performance test

The promising conclusion drawn at the end of the previous section induced us to run an additional test in order to evaluate the quality of GP under a number of iterations larger than 2000. A general aim was to obtain reference makespans better than those published recently in OR Library.[25] We ran GP+M assuming *Maxiter* = 30 000 and *Maxret* = 2000|1000. Results are shown in Table 6. In this table, we present the makespans found in reference 1 (TSAB) in the long-time test, the makespans found in reference 28 ($TSAB_\infty$) for the neighbourhood $\mathcal{H}_\infty(\pi)$, the best makespans found in reference 17 (RY) which are better than the results in reference 1, and only those best makespans found by GP which are better than RY. The RY results are chosen as the best from 30 runs (assuming various random start points) on DEC Alpha 600 5/226 within 12, 21 and 47 min per each run and instance, for 50|20, 100|20 and 200|20, respectively. The corresponding CPU times for GP+M was 3, 7 and 17 min per run; the best makespan was chosen from three runs with size of tabu list *LengthT* of (a) 6, (b) 7 and (c) 8. For $TSAB_\infty$, the CPU times have not been specified in reference 28. The makespans in bold are also better than those for $TSAB_\infty$. The results marked by asterisk (*) are even better than the best results known so far (LB-UB), see OR Library.[25] The reference values LB-UB are considered as a concatenation of partial results obtained by many authors, including the results obtained by the very expensive B&B technique.

Table 6 List of new reference makespans obtained by algorithm GP

50 × 20	*GP+M*	*RY*	*TSAB*	*$TSAB_\infty$*	*LB-UB*
1		3861	3875	3856	3771–3875
2		3709	3715	3714	3661–3715
3		3651	3668	3658	3591–3668
4		3726	3752	3737	3631–3752
5		3614	3635	3619	3551–3635
6		3690	3698	3690	3667–3687
7		3711	3716	3709	3672–3706
8		3699	3709	3700	3627–3700
9	3756[c]	3760	3765	3760	3645–3755
10		3767	3777	3767	3696–3757
100 × 20	*GP+M*	*RY*	*TSAB*	*$TSAB_\infty$*	*LB-UB*
1	***6221**[a]	6242	6286	6238	6106–6228
2	6211[b]	6217	6241	6210	6183–6210
3	**6283**[a]	6299	6329	6296	6252–6271
4		6288	6306	6278	6254–6269
5		6329	6377	6351	6262–6319
6		6380	6437	6406	6302–6403
7	***6287**[b]	6302	6346	6298	6184–6292
8	6431[a]	6433	6481	6423	6315–6423
9	**6286**[c]	6297	6358	6291	6204–6275
10	**6441**[c]	6448	6465	6441	6404–6434
200 × 20	*GP+M*	*RY*	*TSAB*	*$TSAB_\infty$*	*LB-UB*
1	11 216[c]	11 272	11 294	11 213	11 152–11 195
2	11 256[c]	11 299	11 420	11 249	11 143–11 223
3	**11 370**[b]	11 410	11 446	11 382	11 281–11 337
4	***11 295**[a]	11 347	11 347	11 309	11 275–11 299
5	11 273[a]	11 290	11 311	11 265	11 259–11 260
6	11 224[b]	11 250	11 282	11 212	11 176–11 189
7	11 401[c]	11 438	11 456	11 387	11 337–11 386
8	**11 354**[b]	11 395	11 415	11 362	11 301–11 334
9	**11 227**[b]	11 263	11 343	11 241	11 145–11 192
10	11 316[b]	11 335	11 422	11 313	11 284–11 313

Conclusions

In this paper we have presented and discussed some new properties of blocks in the flow-shop problem. These properties allow us to reduce consciously and 'intelligently' the neighbourhood size in tabu search and to direct the search trajectory into a promising region of the solution space.

The computational results show that the algorithms proposed provide better results than recent modern approaches. A particular superiority of our algorithms is observed for large-size problems. Nevertheless, some improvements in our algorithms are possible. For instance, attempts to define some new properties of the blocks should reflect a further essential reduction of the neighbourhoods. Also, the development of new more sophisticated methods of makespan calculations in order to decrease GP running time seems to be the next interesting topic for future research. Finally, certain subset of jobs, instead of one job, may be moved in the permutation while generation the neighbourhoods. This modification would allow a good solution to be obtained quickly which is, in the end, the main purpose of the algorithms.

Although no suitable research has been carried out, proposed neighbourhoods $\mathcal{M}(\pi)$ and $\mathcal{M}^*(\pi)$ seem to be even better for the SA technique than for TS. They allow us to direct the search to the most promising region of the solution space without any increase in the calculation cost. Other useful applications of block properties remain open.

The results obtained encourage us to extend the ideas proposed to other hard problems of sequencing, for example, to the job-shop problem.

Appendix

Proof of Theorem 2

Without loss of generality and for the simplicity of denotation one can assume that $\pi(i) = i$. Thus, $\pi(u_k) = u_k$, $k = 1, 2, \ldots, m-1$. Note that formula (4) takes the form

$$C_{\max}(\pi) = \sum_{j=1}^{u_1} p_{j1} + \sum_{j=u_1}^{u_2} p_{j2} + \cdots + \sum_{j=u_{m-1}}^{n} p_{jm}.$$

Let us consider move $v = (a, b) \in R_a(u_{l-1}, u_l - 1), a \in P_k^*$, made from the permutation $\pi, \pi = (1, \ldots, u_1, \ldots, u_{k-1}, \ldots, a-1, a, a+1, \ldots, u_k, \ldots, u_{l-1}, \ldots, b-1, b, b+1, \ldots, u_l, \ldots, n)$. The resulting permutation takes the following form:

$$\pi = (1, \ldots, u_1, \ldots, u_{k-1}, \ldots, a-1, a+1, \ldots, u_k, \ldots, u_{l-1}, \ldots, b-1, b, a, b+1, \ldots, u_l, \ldots, n).$$

Let $t = (t_1, t_2, \ldots, t_{m-1})$ be a path with respect to π_v such that $\pi_v(t_k) = u_k$. Hence, using (3), we get

$$\begin{aligned} C_{\max}(\pi_v) \geqslant C(\pi_v, t) &= \sum_{j=1}^{u_1} p_{j1} + \cdots + \sum_{j=u_{k-1}}^{a-1} p_{jk} \\ &+ \sum_{j=a+1}^{u_k} p_{jk} \cdots + \sum_{j=u_{l-1}}^{b} p_{jl} + p_{al} \\ &+ \sum_{j=b+1}^{u_l} p_{jl} + \cdots + \sum_{j=u_{m-1}}^{n} p_{jm} = \sum_{j=1}^{u_1} p_{j1} + \cdots \\ &+ \sum_{j=u_{k-1}}^{u_k} p_{jk} - p_{ak} + \cdots \\ &+ \sum_{j=u_{l-1}}^{u_l} p_{jl} + \cdots + \sum_{j=u_{m-1}}^{n} p_{jm} + p_{al} \\ &= C_{\max}(\pi) + \Delta_{kl}(\pi(a)) \end{aligned}$$

which completes the proof of this case. The case, where we make a move to the left, can be proved by analogy.

References

1 Nowicki E and Smutnicki C (1996). A fast tabu search algorithm for the permutation flow-shop problem. *Eur J Oper Res* **91**: 160–175.

2 Ishubuchi M, Masaki S, and Tanaka H (1995). Modified simulated annealing for the flow shop sequencing problems. *Eur J Oper Res* **81**: 388–398.

3 Ogbu FA and Smith DK (1990). The application of the simulated annealing algorithm to the solution $n/m/P/C_{\max}$ flow-shop problem. *Comp Oper Res* **17**: 243–253.

4 Osman IH and Potts CN (1989). Simulated annealing for permutation flow shop scheduling. *OMEGA* **17**: 551–557.

5 Taillard E (1990). Some efficient heuristic methods for flow-shop sequencing. *Eur J Oper Res* **47**: 65–74.

6 Taillard E (1993). Benchmarks for basic scheduling problems. *Eur J Oper Res* **64**: 278–285.

7 Werner F (1993). On the heuristic solution of the permutation flow shop problem. *Comp Oper Res* **20**: 707–722.

8 Widmer M and Hertz A (1989). A new heuristic method for the flow-shop sequencing problem. *Eur J Oper Res* **42**: 186–193.

9 Grabowski J (1980). On two-machine scheduling with release and due dates to minimize maximum lateness. *Opsearch* **17**: 133–154.

10 Grabowski J (1982). A new algorithm of solving the flow-shop problem. In: Feichtinger G and Kall P (eds). *Operations Research in Progress*. D. Reidel: Dordrecht, pp 57–75.

11 Grabowski J, Nowicki E, and Zdrzałka S (1986). A block approach for single machine scheduling with release dates and due dates. *Eur J Oper Res* **26**: 278–285.

12 Grabowski J, Skubalska E and Smutnicki C (1983). On flow-shop scheduling with release and due dates to minimize maximum lateness. *J Opl Res Soc* **34**: 615–620.

13 Grabowski J, Nowicki E, and Smutnicki C (1988). Block algorithm for scheduling of operations in job shop system (Polish). *Przegląd Statystyczny* **35**: 67–80.

14 Zdrzałka S and Grabowski J (1989). An algorithm for single machine sequencing with release dates to minimize maximum cost. *Discr Appl Math* **23**: 73–89.

15 Grabowski J and Smutnicki C (1996). A block approach for flow shop scheduling to minimize maximum cost. In: Trappl R (ed). *Proc. XIII Conference of Cybernetics and Systems'96*, Austrian Society for Cybernetic Studies, Vienna, pp 826–831.

16 Brucker P, Jurish B and Sievers B (1994). A fast branch & bound algorithm for the job shop problem. *Discr Appl Math* **49**: 107–127.
17 Reeves CR and Yamada T (1998). Genetic algorithms, path relinking and the flowshop sequencing problem. *Evol Comp* **6**: 45–60.
18 Balas E and Vazacopoulos A (1998). A guided local search with shifting bottleneck for job-shop scheduling. *Mngt Sci* **44**: 262–275.
19 Nowicki E and Smutnicki C (1993). New results in the worst-case analysis for flow-shop scheduling. *Discr Appl Math* **46**: 21–41.
20 Nowicki E and Smutnicki C (1996). A fast tabu search algorithm for the job-shop problem. *Mngt Sci* **42**: 797–813.
21 Nowicki E and Smutnicki C (1998). Flow shop with parallel machines. A tabu search approach. *Eur J Oper Res* **106**: 226–253.
22 Smutnicki C (1998). A two-machine permutation flow-shop scheduling with buffers. *OR Spectrum* **20**: 229–235.
23 Glover F (1989). Tabu search. Part I. *ORSA J Comp* **1**: 190–206.
24 Glover F (1990). Tabu search. Part II. *ORSA J Comp* **2**: 4–32.
25 OR Library http://mscmga.ms.ic.ac.uk/info.html
26 Spec: http://open.specbench.org/osg/cpu95/results/cint95.html
27 Wintune97: http://wintune.winmag.com/
28 Nowicki E and Smutnicki C (1996). Permutation flow-shop scheduling. Benchmarks results. *Technical Report* PRE 15/96. Technical University of Wroclaw.

Received July 1999;
accepted August 2000 after two revisions