Chapter 5

A VERY FAST TABU SEARCH ALGORITHM FOR JOB SHOP PROBLEM

Józef Grabowski[1], and Mieczyslaw Wodecki[2]

[1]*Wroclaw University of Technology, Institute of Engineering Cybernetics, Janiszewskiego 11-17, 50-372 Wroclaw, Poland, grabow@ict.pwr.wroc.pl;*

[2]*University of Wroclaw, Institute of Computer Science, Przesmyckiego 20, 51-151 Wroclaw, Poland, mwd@ii.uni.wroc.pl*

Abstract This paper deals with the classic job-shop scheduling problem with makespan criterion. Some new properties of the problem associated with blocks are presented and discussed. These properties allow us to propose a new, very fast local search procedure based on a tabu search approach. The central concepts are lower bounds for evaluations of the moves, and perturbations that guide the search to the more promising areas of solution space, where "good solutions" can be found. Computational experiments are given and compared with the results yielded by the best algorithms discussed in the literature. These results show that the algorithm proposed solves the job-shop instances with high accuracy in a very short time. The presented properties and ideas can be applied in many local search procedures.

Keywords: Job-Shop Scheduling, Makespan, Heuristics, Tabu Search

1. Introduction

The paper deals with the job-shop problem, which can be briefly presented as follows. There is a set of jobs and a set of machines. Each job consists of a number of operations, which are to be processed in a given order, each on a specified machine for a fixed duration. The processing of an operation can not be interrupted. Each machine can process at most one operation at a time. We want to find the schedule (the assignment of operations to time intervals on machines) that minimizes the *makespan*.

The job-shop scheduling problem, although relatively easily stated, is NP-hard, and is considered one of the hardest problems in the area of combinatorial optimization. This is illustrated by the fact that a classical benchmark problem

(FT10) of 10 jobs and 10 machines, proposed by Fisher and Thompson (1963), remained unsolved (to optimality) for more than a quarter of a century. Many various methods have been proposed, ranging from simple and fast dispatching rules to sophisticated branch-and bound algorithms. For the literature on job-shop scheduling, see Carlier and Pinson (1989), Morton and Pentico (1993), Nowicki and Smutnicki (1996b), Vaessens, Aarts and Lenstra (1996), Aarts and Lenstra (1997), Balas and Vazacopoulos (1998), and Pezzela and Merelli (2000), and their references. In this paper, we present new properties and techniques which allows us to solve the large-size job-shop instances with high accuracy in a relatively short time

The paper is organized as follows. In Section 2, the notations and basic definitions are introduced. Section 3 presents the new properties of the problem, moves and neighbourhood structure, methods to evaluate the moves, search strategy, dynamic tabu list, perturbations, and algorithm based on a tabu search approach. The central concepts are lower bounds for evaluations of the moves, and perturbations used during the performance of the 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 Formulation and Preliminaries

The job-shop problem can be formally defined as follows, using the notation by Nowicki and Smutnicki (1996b). There are: a set of jobs $J = \{1, 2, ..., n\}$, a set of machines $M = \{1, 2, ..., m\}$, and a set of operations $O = \{1, 2, ..., o\}$. Set O decomposes into subsets (chains) corresponding to the jobs. Each job j consists of a sequence of o_j operations indexed consecutively by $(l_{j-1} + 1, ..., l_{j-1} + o_j)$, which are to be processed in order, where $l_j = \sum_{i=1}^{j} o_i$, is the total number of operations of the first j jobs, $j = 1, 2, ..., n$, $(l_0 = 0)$, and $o = \sum_{i=1}^{n} o_i$. Operation x is to be processed on machine $\mu_x \in M$ during processing time p_x, $x \in O$. The set of operations O can be decomposed into subsets $M_k = \{x \in O | \mu_x = k\}$, each containing the operations to be processed on machine k, and $m_k = |M_k|$, $k \in M$. Let permutation π_k define the processing order of operations from the set M_k on machine k, and let Π_k be the set of all permutations on M_k. The processing order of all operations on machines is determined by m-tuple $\pi = (\pi_1, \pi_2, ..., \pi_m)$, where $\pi \in \Pi_1 \times \Pi_2 \times ... \times \Pi_m$.

It is useful to present the job-shop problem by using a graph. For the given processing order π, we create the graph $G(\pi) = (N, R \cup E(\pi))$ with a set of nodes N and a set of arcs $R \cup E(\pi)$, where:

- $N = O \cup \{s, c\}$, where s and c are two fictitious operations representing dummy "start" and "completion" operations, respectively. The weight of node $x \in N$ is given by the processing time p_x, $(p_s = p_c = 0)$.

- $R = \bigcup_{j=1}^{n} \left[\bigcup_{i=1}^{o_j - 1} \{(l_{j-1} + i, l_{j-1} + i + 1)\} \cup \{(s, l_{j-1} + 1)\} \right.$
  $\left. \cup \{(l_{j-1} + o_j, c)\} \right]$.

  Thus, R contains arcs connecting consecutive operations of the same job, as well as arcs from node s to the first operation of each job and from the last operation of each job to node c.

- $E(\pi) = \bigcup_{k=1}^{m} \bigcup_{i=1}^{m_k - 1} \{(\pi_k(i), \pi_k(i+1))\}$.

  Thus, arcs in $E(\pi)$ connect operations to be processed by the same machine.

Arcs from set R represent the processing order of operations in jobs, whereas arcs from set $E(\pi)$ represent the processing order of operations on machines. The processing order π is feasible if and only if graph $G(\pi)$ does not contain a cycle.

Let $C(x, y)$ and $L(x, y)$ denote the longest (critical) path and length of this path, respectively, from node x to y in $G(\pi)$. It is well-known that makespan $C_{max}(\pi)$ for π is equal to length $L(s, c)$ of critical path $C(s, c)$ in $G(\pi)$. Now, we can rephrase the job-shop problem as that of finding a feasible processing order $\pi \in \Pi$ that minimizes $C_{max}(\pi)$ in the resulting graph.

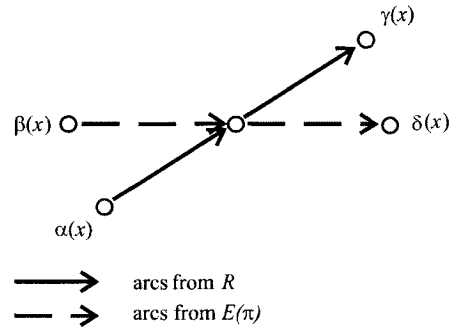

Figure 5.1. Operation predecessors and successors.

We use a notation similar to the paper of Balas and Vazacopoulos (1998). For any operation $x \in O$, we will denote by $\alpha(x)$ and $\gamma(x)$ the job-predecessor and job-successor (if it exists), respectively, of x, i.e. $(\alpha(x), x)$ and $(x, \gamma(x))$ are arcs from R. Further, for the given processing order π, and for any operation $x \in O$, we will denote by $\beta(x)$ and $\delta(x)$ the machine-predecessor and machine-successor (if it exists), respectively, of x, i.e. the operation that precedes x, and

succeeds x, respectively, on the machine processing operation x. In other words, $(\beta(x), x)$ and $(x, \delta(x))$ are arcs from $E(\pi)$, see Figure 5.1.

Denote the critical path in $G(\pi)$ by $C(s, c) = (s, u_1, u_2, \ldots, u_w, c)$, where $u_i \in O$, $1 \leq i \leq w$, and w is the number of nodes (except fictitious s and c) in this path. The critical path $C(s, c)$ depends on π, but for simplicity in notation we will not express it explicitly. The critical path is decomposed into subsequences $B_1, B_2, \ldots, B_r$ called *blocks* in π on $C(s, c)$ (Grabowski, 1979; Grabowski, Nowicki, and Smutnicki, 1988), where

1 $B_k = (u_{f_k}, u_{f_k+1}, ..., u_{l_k-1}, u_{l_k}), \quad 1 \leq f_k \leq l_k \leq w, \quad k = 1, 2, ..., r.$

2 B_k contains operations processed on the same machine, $k = 1, 2, \ldots, r$.

3 two consecutive blocks contain operations processed on different machines.

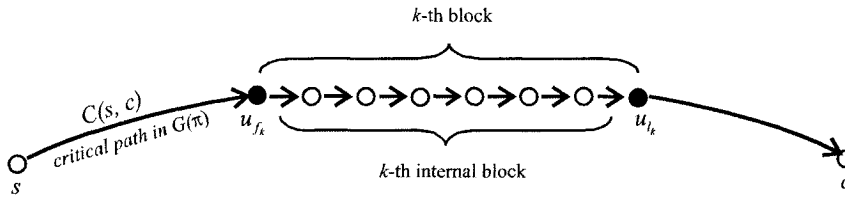

Figure 5.2. Block on critical path.

In other words, the block is a maximal subsequence of $C(s, c)$ and contains successive operations from the critical path processed consecutively on the same machine. In the further considerations, we will be interested only in *non-empty* block, i.e. such that $|B_k| > 1$, or alternatively $f_k < l_k$. Operations u_{f_k} and u_{l_k} in B_k are called the *first* and *last* ones, respectively. The k-th block, exclusive of the first and last operations, is called the k-th *internal block*, see Figure 5.2.

A block has advantageous so-called *elimination properties*, introduced originally in the form of the following theorem (Grabowski, 1979; Grabowski, Nowicki, and Smutnicki, 1988).

THEOREM 5.1 . *Let $G(\pi)$ be an acyclic graph with blocks B_k, $k = 1, 2, ..., r$. If acyclic graph $G(\omega)$ has been obtained from $G(\pi)$ through the modifications of π so that $C_{max}(\omega) < C_{max}(\pi)$, then in $G(\omega)$*

(i) at least one operation $x \in B_k$ precedes job u_{f_k}, for some $k \in \{1, 2, ..., r\}$, or

(ii) at least one operation $x \in B_k$ succeeds job u_{l_k}, for some $k \in \{1, 2, ..., r\}$.

3. Tabu Search Algorithm (TS)

Currently, TS is one of the most effective methods using local search techniques to find near-optimal solutions to combinatorial optimization problems, see Glover (1989, 1990). The basic idea in our context involves starting from an initial basic job processing order and searching through its neighbourhood, for the processing order with the lowest makespan (in our case, the processing order with the lowest lower bound on the makespan). The search then repeats using the chosen neighbor as a new basic processing order.

The neighbourhood of a basic processing order is generated by the moves. A move changes the location of some operations in the basic processing order. In order to avoid cycling, becoming trapped at a local optimum, or continuing the search in a too narrow region, the mechanisms of a tabu list and a perturbation are utilized. The tabu list records the performed moves, for a chosen span of time, treating them as forbidden for possible future moves, i.e. it determines forbidden processing orders in the currently analyzed neighbourhood. The list content is refreshed each time a new basic processing order is found: the oldest elements are deleted and new ones added. The search stops when a given number of iterations has been reached without improvement of the best current makespan, the algorithm has performed a given total number of iterations, time has run out, the neighbourhood is empty, or a processing order with a satisfying makespan has been found, etc. In practice, the design of the particular components of a search algorithm is considered an art. The construction of the components influences the algorithm performance, speed of convergence, running time, etc.

3.1 Moves and Neighbourhood

In the literature are many types of moves based on interchanges of operations (or jobs) on a machine. The intuition following from Theorem 5.1 suggests that the "insertion" type move is the most proper for the problem considered. In general, the insertion move operates on a sequence of operations by removing an operation from its position in a sequence and inserting it in to another position in the same sequence. More precisely, let $v = (x, y)$ be a pair of operations on a machine, $x, y \in O$, $x \neq y$. With respect to graph $G(\pi)$, **the pair $v = (x, y)$ defines a move that consists in removing operation x from its original position and inserting it in the position immediately after (or before) operation y if operation y succeeds (or precedes) operation x in** $G(\pi)$. This move v generates a new graph $G(\pi_v)$ from $G(\pi)$. All graphs $G(\pi_v)$, which can be obtained by performing moves from a given move set U, create the neighbourhood $N(U, \pi) = \{G(\pi_v) \mid v \in U\}$ of graph $G(\pi)$.

The proper definition of the move and selection of U, i.e. the neighbourhood $N(U, \pi)$, is very important in constructing an effective algorithm. The set U

should be neither too "big" nor too "small". The large set requires a great computational effort for the search of $N(U, \pi)$ at a given iteration of the algorithm, whereas the small one needs a large number of iterations for finding a "good" solution.

For the job-shop problem there are several definitions of moves based on the interchanges of adjacent and non-adjacent pairs of operations on a machine.

The interchange moves of the adjacent pairs have been used earlier by Balas (1969), while the non-adjacent ones by Grabowski (1979), Grabowski and Janiak, (1987), and Grabowski, Nowicki and Smutnicki (1988). The latter moves were widely employed for the flow-shop problem by Grabowski (1980, 1982), and Grabowski, Skubalska and Smutnicki (1983), and for the one-machine scheduling by Carlier (1982), Grabowski, Nowicki and Zdrzalka (1986), Adams, Balas and Zawack (1988), and Zdrzalka and Grabowski (1989). All these moves were applied in branch-and-bound procedures.

Recently, in the heuristics for the job-shop problem, the adjacent moves have been employed by Matsuo, Suh and Sullivan (1988), Laarhoven, Aarts and Lenstra (1992), and Nowicki and Smutnicki (1996b), while the non-adjacent ones by DellAmico and Trubian (1993), Balas and Vazacopoulos (1998), and Pezzela and Merelli (2000). Besides, the latter moves were used by Nowicki and Smutnicki (1996a), Smutnicki (1998), and Grabowski and Pempera (2001) in the tabu search algorithms for the flow-shop problem.

The second component of the local search algorithms is a selection (construction) of "effective" neighbourhood $N(U, \pi)$. Amongst many types of neighbourhoods considered (and connected with the chosen definition of the move), two appear to be very interesting.

The first is that proposed by Nowicki and Smutnicki (1996b). In point of the computational results, it seems that their neighbourhood used in the tabu search procedure with built-in block properties (and based on interchanges of some adjacent pairs only) is "optimal". However, we believe this neighbourhood is "too small", that is, their TS needs too many iterations. Despite out criticism, the computational results obtained by Nowicki and Smutnicki (1996b) are excellent. Their spectacular success encourages further explorations in that area.

The second neighbourhood (based on interchanges of non-adjacent pairs), presented by Balas and Vazacopoulos (1998), is employed in their local search (tree search) algorithm, denoted GLS. This algorithm is based on the branch-and-bound procedure with an enumeration tree whose size is bounded in a guided manner, so that GLS can be treated as an approximation algorithm. It is clear that the largest size of the neighbourhood is at the root of the tree, and while searching, the size decreases with increasing levels of the tree. Additionally, GLS consists of several such procedures (each of them starting with various initial solutions). As a consequence, it is difficult to compare the neighbour-

hood size of GLS with those given in the literature. However, with regard to the neighbourhood at the root of the tree, GLS investigates a considerably larger neighbourhood than the heuristics based on interchanges of adjacent pairs of operations. Computational results obtained by GLS confirm an advantage over other heuristics for the job-shop problem. In our algorithm TS, the neighbourhood is larger than that at the root of the tree in GLS, and is based on the block approach. Besides, in order to reduce calculations for the search of neighbourhood, we propose to use a lower bound on the makespan instead of calculating the makespan explicitly, as a basis for choosing the best move.

For any block B_k in $G(\pi)$ (acyclic), let us consider the set of moves $W_k(\pi)$ which can be performed inside this block, i.e. on operations $u_{f_k+1}, ..., u_{l_k-1}$, $k = 1, 2, ..., r$. Precisely, each $W_k(\pi)$ is defined by the formula

$$W_k(\pi) = \{(x, y) \mid x, y \in \{u_{f_k+1}, ..., u_{l_k-1}\},\ x \neq y\}.$$

All these moves create the set $W(\pi) = \bigcup_{k=1}^{r} W_k(\pi)$.

Immediately from Theorem 5.1 we obtain the following Corollary which provides the basis for elimination (Grabowski, 1979; Grabowski, Nowicki, and Smutnicki, 1988).

COROLLARY 1 . *If acyclic graph $G(\pi_v)$ has been generated from acyclic graph $G(\pi)$ by a move $v \in W(\pi)$, then $C_{max}(\pi_v) \geq C_{max}(\pi)$.*

This Corollary states that the moves from the set $W(\pi)$ defined above are not interesting, taking into account the possibility of an immediate improvement of the makespan after making a move.

Next, we will give a detailed description of the moves and neighbourhood structure considered in this paper. Let us consider the sequence of operations on critical path $C(s, c)$ in $G(\pi)$ and blocks $B_1, B_2, \ldots, B_r$ determined for $C(s, c)$. For each fixed operation x belonging to the critical path $C(s, c)$, we consider at most one move to the right and at most one to the left. Moves are associated with blocks. Let us take the block $B_k = \{u_{f_k}, u_{f_k+1}, ..., u_{l_k-1}, u_{l_k}\}$, $k = 1, 2, ..., r$. Then, we define the sets of *candidates* (Grabowski, 1979; Grabowski, Nowicki, and Smutnicki, 1988).

$$E_{ka} = \{u_{f_k}, u_{f_k+1}, ..., u_{l_k-1}\} = B_k - \{u_{l_k}\},$$
$$E_{kb} = \{u_{f_k+1}, ..., u_{l_k-1}, u_{l_k}\} = B_k - \{u_{f_k}\}.$$

Each set E_{ka} (or E_{kb}) contains the operations in the k-th block of $G(\pi)$ that are candidates for being moved to a position *after* (or *before*) all other operations in the k-th block. More precisely, we move operation x, $x \in E_{ka}$, to the right in to the position immediately after operation u_{l_k}, and this move takes the form $v = (x, u_{l_k})$, so Corollary 1 can not be applied to this move, i.e. $v \notin W_k$. By symmetry, operation x, $x \in E_{kb}$, is moved to the left in the position immediately before operation u_{f_k}, and this move takes the form $v = (x, u_{f_k})$, so $v \notin W_k$. Note that after performing a move $v = (x, u_{l_k})$, $x \in E_{ka}$ (or $v = (x, u_{f_k})$, $x \in E_{kb}$), operation x, in $G(\pi_v)$, is to be processed as the last (or first) operation of the k-th block of $G(\pi)$. It is easy to observe that in order to obtain the graph $G(\pi_v)$ by performing a move $v = (x, u_{l_k})$, $x \in E_{ka}$ (or $v = (x, u_{f_k})$, $x \in E_{kb}$), we should remove the arcs $(\beta(x), x)$, $(x, \delta(x))$ and $(u_{l_k}, \delta(u_{l_k}))$ from $G(\pi)$ and add to $G(\pi)$ the arcs $(\beta(x), \delta(x))$, (u_{l_k}, x) and $(x, \delta(u_{l_k})$ (or remove the arcs $(\beta(x), x)$, $(x, \delta(x))$ and $(\delta(u_{f_k}), u_{f_k})$, and add the arcs $(\beta(x), \delta(x))$, (x, u_{f_k}) and $(\beta(u_{f_k}), x)$). For illustration, performing the move $v = (x, u_{l_k})$ is shown in Figure 5.3.

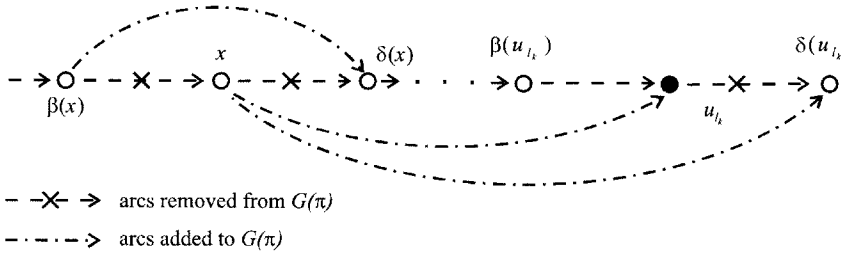

Figure 5.3. Move performance.

According to the description given, for any block B_k in $G(\pi)$, $k = 1, 2, \ldots, r$, we define the following set of moves to the right

$$ZR_k(\pi) = \{(x, u_{l_k}) | x \in E_{ka}\}$$

and the set of moves to the left

$$ZL_k(\pi) = \{(x, u_{f_k}) | x \in E_{kb}\}.$$

Set $ZR_k(\pi)$ contains all moves of operations of E_{ka} to the right after the last operation u_{l_k} of the k-th block. Similarly, set $ZL_k(\pi)$ contains all moves of operations of E_{kb} to the left before the first operation u_{f_k} of the k-th block. Of

course, Corollary 1 does not hold for moves from the sets $ZR_k(\pi)$ and $ZL_k(\pi)$. For illustration, the moves performed to the right and left are shown in Figure 5.4.

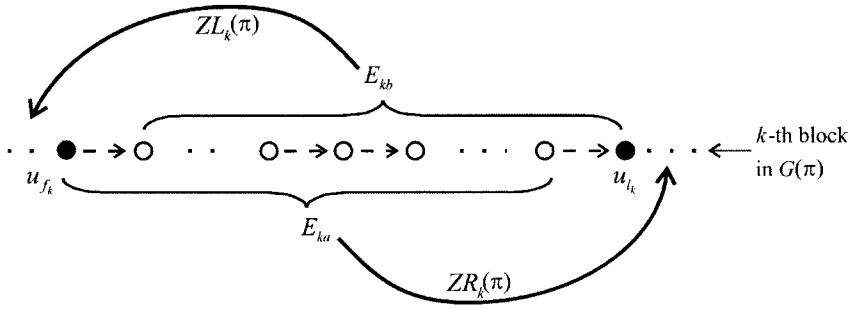

Figure 5.4. Operation movements.

Note that if $|B_k| = 2$, for some $k \in \{1, 2, \ldots, r\}$, then $E_{ka} = \{u_{f_k}\}$, $E_{kb} = \{u_{l_k}\}$, and $ZR_k(\pi) = ZL_k(\pi)$, and one of these sets can be eliminated. If we assume that $E_{ka} = \{u_{l_k-1}\}$ in $ZR_k(\pi)$ and $E_{kb} = \{u_{f_k+1}\}$ in $ZL_k(\pi)$ then $ZR_k(\pi) \cup ZL_k(\pi)$ is similar to that presented by Nowicki and Smutnicki (1996b), denoted as $V_k(\pi)$.

As a consequence of the above considerations, in TS, we should take the set of moves

$$M(\pi) = \bigcup_{k=1}^{r} (ZR_k(\pi) \cup ZL_k(\pi))$$

and the resulting neighbourhood $N(M(\pi), \pi)$.

A set of moves similar to $M(\pi)$ has been proposed by Grabowski (1980, 1982) and Grabowski, Skubalska and Smutnicki (1983) for the flow-shop problem. However, for the job-shop problem, the neighbourhood $N(M(\pi), \pi)$ contains processing orders which can be infeasible. It should be noticed that if a move $v = (x, u_{l_k}) \in ZR_k(\pi)$, (or $v = (x, u_{f_k}) \in ZL_k(\pi)$) contains an adjacent pair of operations, i.e. $x = u_{l_k-1} \in E_{ka}$, (or $x = u_{f_k+1} \in E_{kb}$), then the resulting graph $G(\pi_v)$ is acyclic (Balas, 1969; Laarhoven, Aarts, and Lenstra, 1992).

In sequel, we consider conditions under which performing a move $v = (x, u_{l_k}) \in ZR_k(\pi)$, $x \neq u_{l_k-1}$, (or $v = (x, u_{f_k}) \in ZL_k(\pi)$, $x \neq u_{f_k+1}$) in an acyclic $G(\pi)$, generates the acyclic graph $G(\pi_v)$.

THEOREM 5.2 *For each acyclic $G(\pi)$, if $G(\pi_v)$ has been generated by a move $v = (x, u_{l_k}) \in ZR_k(\pi)$, $x \neq u_{l_k-1}$, $k = 1, 2, \ldots, r$, and if in $G(\pi)$*

$$L(u_{l_k}, c) + min(p_{\alpha(u_{l_k})}, p_{u_{l_k}-1}) + p_{\gamma(x)} > L(\gamma(x), c), \qquad (5.1)$$

and $\gamma(x) \neq \alpha(u_{l_k})$, then $G(\pi_v)$ is acyclic.

Proof (by contradiction).

For simplicity, the index k will be dropped. For a move $v = (x, u_l)$, $x \in E_a$, $x \neq u_{l-1}$, we suppose that there is created a cycle C in $G(\pi_v)$. It is obvious that C contains some arcs that are added to graph $G(\pi)$ (see Figure 5.5).

If $(\beta(x), \delta(x)) \in C$, then $G(\pi)$ contains a path from $\delta(x)$ to $\beta(x)$, which contradicts the assumption that $G(\pi)$ is acyclic. Therefore, C can contain $(x, \delta(u_l))$ or (u_l, x). If C contains both these arcs, then there is a path in $G(\pi)$ from $\delta(u_l)$ to u_l, contrary to the assumption that $G(\pi)$ is acyclic. Hence, C contains either $(x, \delta(u_l))$, or (u_l, x). If $(x, \delta(u_l)) \in C$, then there is a path in $G(\pi)$ from $\delta(u_l)$ to x, again contrary to the assumption. Finally, if $(u_l, x) \in C$, then C contains

a) a path $d_1(x, u_l) = ((x, \gamma(x), d(\gamma(x), \alpha(u_l)), (\alpha(u_l), u_l))$, or

b) a path $d_2(x, u_l) = ((x, \gamma(x), d(\gamma(x), u_{l-1}), (u_{l-1}, u_l))$,

(a) In this case if C contains path $d_1(x, u_l)$, then this path is in $G(\pi)$ and, since $\gamma(x) \neq \alpha(u_l)$, we obtain

$$L(\gamma(x), c) \geq L(u_l, c) + p_{\alpha(u_l)} + p_{\gamma(x)}. \qquad (5.1a)$$

(b) But if C contains path $d_2(x, u_l)$, then this path is in $G(\pi)$, and now we obtain

$$L(\gamma(x), c) \geq L(u_l, c) + p_{u_l - 1} + p_{\gamma(x)}. \qquad (5.1b)$$

Together (5.1a) and (5.1b) imply

$$L(\gamma(x), c) \geq L(u_l, c) + min(p_{\alpha(u_l)}, p_{u_l - 1}) + p_{\gamma(x)},$$

which contradicts the assumption 5.1.

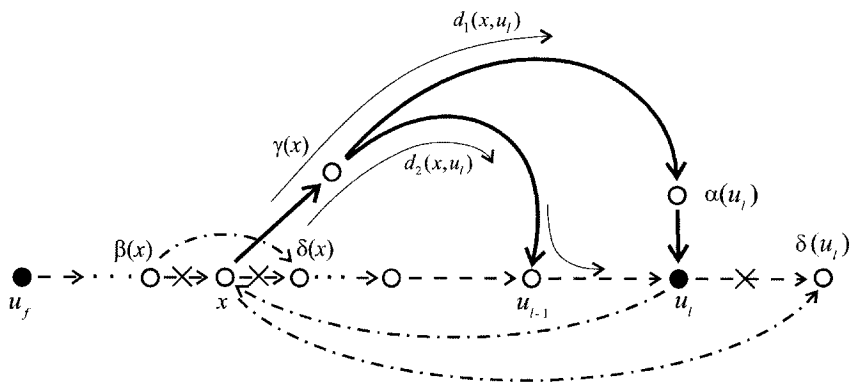

Figure 5.5. Paths $d_1(x, u_l)$ and $d_2(x, u_l)$ in $G(\pi)$.

The considerations in the proof of Theorem 5.2 suggest the following property.

PROPERTY 1 *For each acyclic $G(\pi)$, if $G(\pi_v)$ has been generated by a move $v = (x, u_{l_k}) \in ZR_k(\pi)$, $k = 1, 2, \ldots, r$ and if x has no job-successor $\gamma(x)$, then $G(\pi_v)$ is acyclic.*

By symmetry, we have

THEOREM 5.3 *For each acyclic $G(\pi)$, if $G(\pi_v)$ has been generated by a move $v = (x, u_{f_k}) \in ZL_k(\pi)$, $x \neq u_{f_k+1}$, $k = 1, 2, \ldots, r$, and if in $G(\pi)$*

$$L(s, u_{f_k}) + min(p_{\gamma(u_{f_k})}, p_{u_{f_k}+1}) + p_{\alpha(x)} > L(s, \alpha(x)), \qquad (5.2)$$

and $\alpha(x) \neq \gamma(u_{f_k})$, then $G(\pi_v)$ is acyclic.

The proof of Theorem 5.3 can be obtained by using similar considerations to Theorem 5.2, but with the set of moves $ZL_k(\pi)$.
By analogy, we have

PROPERTY 2 *For each acyclic $G(\pi)$, if $G(\pi_v)$ has been generated by a move $v = (x, u_{f_k}) \in ZL_k(\pi)$, $k = 1, 2, \ldots, r$, and if x has no job-predecessor $\alpha(x)$, then $G(\pi_v)$ is acyclic.*

Note that the conditions 5.1 and 5.2 are both less restrictive than those given by Balas and Vazacopoulos (1998) for procedure GLS, so that our neighbourhood is larger than that proposed by Balas and Vazacopoulos (1998), but it is slightly smaller than that of DellAmico and Trubian (1993).

Let

$$ZR_k^*(\pi) = \{v \in ZR_k(\pi) | v \text{ satisfies 5.1 and } \gamma(x) \neq \alpha(u_{l_k}), \text{ or } x = u_{l_k-1}\},$$
$$ZL_k^*(\pi) = \{v \in ZL_k(\pi) | v \text{ satisfies 5.2 and } \alpha(x) \neq \gamma(u_{f_k}), \text{ or } x = u_{f_k+1}\},$$

be the sets of the moves from $ZR_k(\pi)$ and $ZL_k(\pi)$, the performance of which generates acyclic $G(\pi_v)$ from acyclic $G(\pi)$. Finally, in our TS, we will employ the set of moves

$$M^*(\pi) = \bigcup_{k=1}^{r} (ZR_k^*(\pi) \cup ZL_k^*(\pi)),$$

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

As a consequence of the above considerations, let

$$E_{ka}^* = \{x \in E_{ka} \mid (x, u_{l_k}) \in ZR_k^*(\pi)\},$$
$$E_{kb}^* = \{x \in E_{kb} \mid (x, u_{f_k}) \in ZL_k^*(\pi)\},$$

be the sets of operations whose movement generates acyclic $G(\pi_v)$ from acyclic $G(\pi)$.

In order to decrease the total computational effort for the search, we propose calculation of a lower bound on the makespans instead of computing the makespans explicitly for use in selecting the best solution, though doing so can increase the number of iterations in TS. The makespan resulting from performing a move can be calculated by using the standard Bellman's algorithm in $O(o)$ time, however, doing this for every solution becomes too expensive, so that we propose a less costly lower bound. In fact, this lower bound is used for evaluating and selecting the "best" move.

Next, we present a method to indicate a move to be performed, i.e. an operation which should be moved after the last operation u_{l_k} (or before the first operation u_{f_k}) of the k-th block. According to the search strategy in TS, we want to choose a move v which will generate graph $G(\pi_v)$ with the smallest possible makespan $L_v(s,c)$. To evaluate all moves from the sets $ZR_k^*(\pi)$ and $ZL_k^*(\pi)$, (i.e. all operations from E_{ka}^* and E_{kb}^*) we introduce the formula

$$\Delta_{ka}(x) = max(L_1^a,\ L_2^a,\ L_3^a,\ L_4^a,\ L_5^a), \qquad x \in E_{ka}^*,$$

where:

$$\begin{aligned}
L_1^a &= -p_x, \\
L_2^a &= L(\gamma(x),c) - L(u_{l_k},c) + p_{u_{l_k}}, \\
L_3^a &= L(s,\alpha(\delta(x))) - L(s,x), \\
L_4^a &= L_2^a + L_3^a + p_x, \\
L_5^a &= L(\gamma(\beta(x)),c) - L(x,c), \qquad x \neq u_{f_k}.
\end{aligned}$$

And

$$\Delta_{kb}(x) = max(L_1^b,\ L_2^b,\ L_3^b,\ L_4^b,\ L_5^b), \qquad x \in E_{kb}^*,$$

where:

$$\begin{aligned}
L_1^b &= -p_x, \\
L_2^b &= L(s,\alpha(x)) - L(s,u_{f_k}) + p_{u_{f_k}}, \\
L_3^b &= L(\gamma(\beta(x)),c) - L(x,c), \\
L_4^b &= L_2^b + L_3^b + p_x, \\
L_5^b &= L(s,\alpha(\delta(x))) - L(s,x), \qquad x \neq u_{l_k}.
\end{aligned}$$

The complexity of $\Delta_{k\lambda}(x)$, $\lambda \in \{a,b\}$, having the components $L(s,i)$ and $L(i,c)$, $i \in N$, is $O(1)$. Note that these components are obtained during the calculation of the makespan $L(s,c)$ in $G(\pi)$. Here, if there does not exist $\gamma(x)$ (or $\alpha(x)$) for some x, then $\gamma(x) = c$ and $L(\gamma(x),c) = 0$ (or $\alpha(x) = s$ and $L(s,\alpha(x)) = 0$). Further, if there does not exist $\alpha(\delta(x))$ (or $\gamma(\beta(x))$ for some x, then $\alpha(\delta(x)) = s$ and $L(s,\alpha(\delta(x))) = 0$ (or $\gamma(\beta(x)) = c$ and $L(\gamma(\beta(x)),c) = 0$). Note that if $x = u_{f_k}$ (or $x = u_{l_k}$), then L_5^a (or L_5^b)

is not used during the calculation of $\Delta_{ka}(x)$ (or $\Delta_{kb}(x)$). The usefulness of these values $\Delta_{k\lambda}(x)$, $\lambda \in \{a, b\}$, for the choice of a move is illustrated by the following Theorems.

THEOREM 5.4 *For each acyclic $G(\pi)$, if $G(\pi_v)$ has been generated by a move $v = (x, u_{l_k}) \in ZR_k^*(\pi)$, $k = 1, 2, \ldots, r$, then*

$$L_v(s, c) \geq L(s, c) + \Delta_{ka}(x)$$

where $L_v(s, c)$ is the length of a critical path in $G(\pi_v)$.

Proof.

For simplicity, the index k will be dropped, then we have

$$\begin{aligned}
&L(s,c) + \Delta_a(x) = L(s,c) + max(L_1^a,\ L_2^a,\ L_3^a,\ L_4^a,\ L_5^a)\\
&= L(s,c) + max(-p_x,\ L(\gamma(x),c) - L(u_l,c) + p_{u_l},\ L(s,\alpha(\delta(x)))\\
&-L(s,x),\ L(\gamma(x),c) - L(u_l,c) + p_{u_l} + L(s,\alpha(\delta(x))) - L(s,x)\\
&+p_x,\ L(\gamma(\beta(x)),c) - L(x,c)).
\end{aligned}$$

Since $G(\pi)$ is acyclic, then there exists a critical path. And for each node $i \in N$ which belongs to the critical path, we have

$$C(s,c) = (C(s,i), C(i,c)),$$

and

$$L(s,c) = L(s,i) + L(i,c) - p_i. \tag{5.3}$$

Further, since the nodes $\beta(x)$, x, $\delta(x)$ and u_l belong to the critical path $C(s,c)$, then, using 5.3, we get

$$\begin{aligned}
&L(s,c) + \Delta_a(x) = max(L(s,c) - p_x,\ L(s,c) + L(\gamma(x),c)\\
&-L(u_l,c) + p_{u_l},\ L(s,c) + L(s,\alpha(\delta(x))) - L(s,x),\ L(s,c) + L(\gamma(x),c)\\
&-L(u_l,c) + p_{u_l} + L(s,\alpha(\delta(x))) - L(s,x) + p_x,\ L(s,c) - L(x,c)\\
&+L(\gamma(\beta(x)),c)) = max(L(s,x) + L(x,c) - 2p_x,\ L(s,u_l) + L(u_l,c)\\
&+p_{u_l} + L(\gamma(x),c) - L(u_l,c) + p_{u_l},\ L(s,x) + L(\delta(x),c)\\
&+L(s,\alpha(\delta(x))) - L(s,x),\ L(s,x) + L(\delta(x),u_l) + L(u_l,c) - p_{u_l}\\
&+L(\gamma(x),c) - L(u_l,c) + p_{u_l} + L(s,\alpha(\delta(x))) - L(s,x)\\
&+p_x,\ L(s,\beta(x)) + L(x,c) + L(\gamma(\beta(x)),c) - L(x,c))\\
&= max(L(s,x) + L(x,c) - 2p_x,\ L(s,u_l) + L(\gamma(x),c),\ L(\delta(x),c)\\
&+L(s,\alpha(\delta(x))),\ L(s,\alpha(\delta(x))) + L(\delta(x),u_l) + p_x\\
&+L(\gamma(x),c),\ L(s,\beta(x)) + L(\gamma(\beta(x)),c)).
\end{aligned} \tag{5.4}$$

Now, let us consider certain paths $d_1(s,c)$, $d_2(s,c)$, $d_3(s,c)$, $d_4(s,c)$ and $d_5(s,c)$ from s to c in $G(\pi_v)$ generated by the move $v = (x, u_l) \in ZR_k^*(\pi)$ (see Figure 5.6),

$$
\begin{aligned}
d_1(s,c) &= (C(s,\beta(x)),(\beta(x),\delta(x)),C(\delta(x),c)),\\
d_2(s,c) &= (C(s,\beta(x)),(\beta(x),\delta(x)),C(\delta(x),u_l),(u_l,x),(x,\gamma(x)),\\
&\quad C(\gamma(x),c)),\\
d_3(s,c) &= (C(s,\alpha(\delta(x))),(\alpha(\delta(x)),\delta(x)),C(\delta(x),c)),\\
d_4(s,c) &= (C(s,\alpha(\delta(x))),(\alpha(\delta(x)),\delta(x)),C(\delta(x),u_l),(u_l,x),\\
&\quad (x,\gamma(x)),C(\gamma(x),c)),\\
d_5(s,c) &= (C(s,\beta(x)),(\beta(x),\gamma(\beta(x))),C(\gamma(\beta(x)),c)).
\end{aligned}
$$

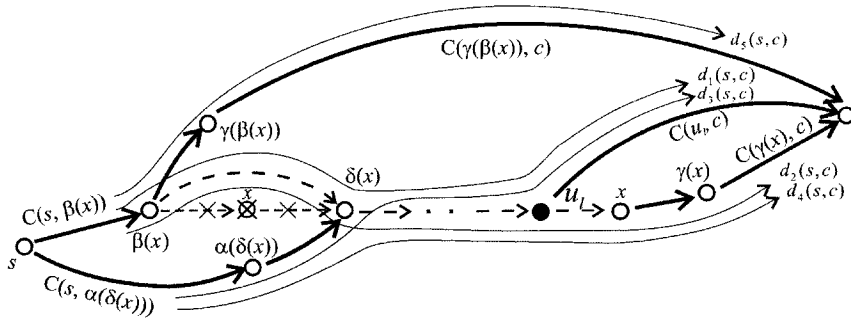

Figure 5.6. Paths $d_1(s,c)$, $d_2(s,c)$, $d_3(s,c)$, $d_4(s,c)$ and $d_5(s,c)$ in $G(\pi_v)$.

The lengths of these paths are

$$
\begin{aligned}
l_1(s,c) &= L(s,\beta(x)) + L(\delta(x),c) = L(s,x) - p_x + L(x,c) - p_x\\
&= L(s,x) + L(x,c) - 2p_x,\\
l_2(s,c) &= L(s,\beta(x)) + L(\delta(x),u_l) + p_x + L(\gamma(x),c) = L(s,u_l)\\
&+ L(\gamma(x),c),\\
l_3(s,c) &= L(s,\alpha(\delta(x))) + L(\delta(x),c),\\
l_4(s,c) &= L(s,\alpha(\delta(x))) + L(\delta(x),u_l) + p_x + L(\gamma(x),c),\\
l_5(s,c) &= L(s,\beta(x)) + L(\gamma(\beta(x)),c).
\end{aligned}
$$

Since $G(\pi_v)$ is acyclic, then there exists a critical path $C_v(s,c)$, the length of which can not be shorter than the length of any path from s to c in $G(\pi_v)$. Therefore, we have

$$
\begin{aligned}
L_v(s,c) &\geq l_1(s,c),\quad L_v(s,c) \geq l_2(s,c),\quad L_v(s,c) \geq l_3(s,c),\\
L_v(s,c) &\geq l_4(s,c),\quad L_v(s,c) \geq l_5(s,c).
\end{aligned}
$$

Hence, using 5.4, we get

$$\begin{aligned}
&L_v(s,c) \geq max(l_1(s,c),\ l_2(s,c),\ l_3(s,c),\ l_4(s,c),\ l_5(s,c)) \\
&= max(L(s,x) + L(x,c) - 2p_x,\ L(s,u_l) \\
&+L(\gamma(x),c),\ L(\delta(x),c) + L(s,\alpha(\delta(x))),\ L(s,\alpha(\delta(x))) \\
&+L(\delta(x),u_l) + p_x + L(\gamma(x),c),\ L(s,\beta(x)) + L(\gamma(\beta(x)),c)) \\
&= L(s,c) + \Delta_a(x).
\end{aligned}$$

An analogous result holds for moves from the set $ZL_k^*(\pi)$.

THEOREM 5.5 *For each acyclic $G(\pi)$, if $G(\pi_v)$ has been generated by a move $v = (x, u_{f_k}) \in ZL_k^*(\pi)$, $k = 1, 2, \ldots, r$, then*

$$L_v(s,c) \geq L(s,c) + \Delta_{kb}(x),$$

where $L_v(s,c)$ is the length of the critical path in $G(\pi_v)$.

Proof. Parallels that of Theorem 5.4.

Hence, by moving operation $x \in E_{ka}^*$ (or $x \in E_{kb}^*$) after operation u_{l_k} (or before operation u_{f_k}) in $G(\pi)$, a lower bound on value $L_v(s,c)$ of acyclic graph $G(\pi_v)$ is $L(s,c) + \Delta_{ka}(x)$ (or $L(s,c) + \Delta_{kb}(x)$). Thus, the values $\Delta_{k\lambda}(x)$, $\lambda \in \{a,b\}$, can be used to decide which operation should be moved, i.e. the operation should have the smallest value of $\Delta_{k\lambda}(x)$. The smallest value of $\Delta_{k\lambda}(x)$ corresponds to the "best" move $v = (x, u_{l_k}) \in ZR_k^*(\pi)$, (or $v = (x, u_{f_k}) \in ZL_k^*(\pi)$) if $\Delta_{ka}(x)$ (or $\Delta_{kb}(x)$) reaches this value. From Theorems 5.4 and 5.5 it follows that if $\Delta_{k\lambda}(x) > 0$, then in the resulting graph $G(\pi_v)$ we have $L_v(s,c) > L(s,c)$.

Generally, in our TS, for the given graph $G(\pi)$, we calculate the critical path $C(s,c)$ (if there is more than one critical path, any one of them can be used), and the length of this path $C_{max}(\pi)$ ($= L(s,c)$). We then identify the blocks $B_1, B_2, \ldots, B_r$, create the set of moves $M^*(\pi)$, compute the values $\Delta_{k\lambda}(x)$, $x \in E_{k\lambda}^*$, $\lambda \in \{a,b\}$, $k = 1, 2, \ldots, r$, choose the "best" move v (corresponding to the smallest value of $\Delta_{k\lambda}(x)$) from set $M^*(\pi)$ and create the graph $G(\pi_v)$ by removing some arcs from $G(\pi)$ and adding other ones to $G(\pi)$ (see beginning of this section). Next, the search process of TS is repeated for the resulting graph $G(\pi_v)$ until $Maxiter$ of iterations is reached. Of course, according to the philosophy of TS, there are some exceptions while choosing the "best" move:

A. If the chosen move has a status tabu (see next section for details), the move is not allowed.

B. If $MaxretP$ ($MaxretP < Maxiter$) of the consecutive non-improving iterations pass in TS, then, instead of a single ("best") move, we choose several ones to be performed simultaneously (see section **Perturbations** for details).

Exception (B) gives assistance in addition to the tabu list to avoid being trapped at a local optimum.

3.2 Tabu List and Tabu Status of Move

In our algorithm we use the tabu list defined as a finite list (set) T with dynamic length $LengthT$ containing ordered pairs of operations. The list is initiated by introducing $LengthT$ empty elements. If a move $v = (x, u_{t_k}) \in ZR_k^*(\pi)$, (or move $v = (x, u_{f_k}) \in ZL_k^*(\pi)$) is performed on graph $G(\pi)$ generating graph $G(\pi_v)$, then the pair of operations $(\delta(x), x)$ (or pair $(x, \beta(x))$), representing a precedence constraint, is added to T. Each time before adding a new pair to T, we must delete the oldest one.

With respect to graph $G(\pi)$, a move $(x, u_{t_k}) \in ZR_k^*(\pi)$, (or a move $(x, u_{f_k}) \in ZL_k^*(\pi)$) has the *tabu* status (it is forbidden) if $A(x) \cap B_k \neq \emptyset$ (or $B(x) \cap B_k \neq \emptyset$), where:

$$A(x) = \{y \in O \mid (x, y) \in T\},$$
$$B(x) = \{y \in O \mid (y, x) \in T\}.$$

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

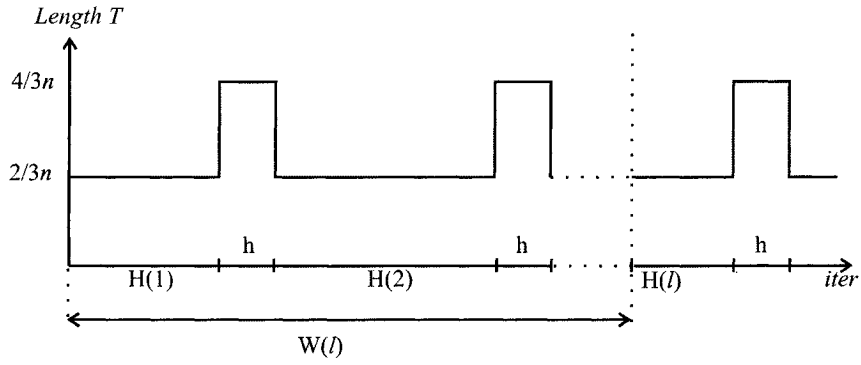

Figure 5.7. Dynamic tabu list.

As mentioned above, our algorithm uses a tabu list with dynamic length. This length is changed, as the current iteration number *iter* of TS increases. The length change is used as a "pick" intended to carry the search to another

area of the solution space. $LengthT$ is a cyclic function shown in Figure 5.7 and defined by the expression

$$LengthT = \begin{cases} \left\lceil \frac{2}{3}n \right\rceil, & \text{if } W(l) < iter \leq W(l) + H(l), \\ \left\lceil \frac{4}{3}n \right\rceil, & \text{if } W(l) + H(l) < iter \leq W(l) + H(l) + h, \end{cases}$$

where: $l = 1, 2, ...$ is the number of the cycle, $W(l) = \sum_{s=1}^{l} H(s-1) + (l-1) * h$, (here $H(0) = 0$), and h is the width of the pick equal to n. Interval $H(l)$ is the parameter which is not constant, but it depends on the structure of graph $G(\pi)$ currently considered. More precisely, let $G(\pi)$ be the graph obtained at the beginning of the interval $H(l)$, i.e. in $W(l)+1$ iteration (see expression on $LengthT$). Then the next pick is begun when $H(l) = 2 \times |C|$ iterations pass in TS, where $|C|$ is the number of nodes in the critical path of $G(\pi)$. The one exception is for the first cycle when we take $H(1) = 3 \times |C|$.

If $LengthT$ decreases then a suitable number of the oldest elements of tabu list T is deleted and the search process is continued.

3.3 Search Strategy

We employ a specific searching strategy which yields very good computational results. A move $v = (x, u_{l_k}) \in M^*(\pi)$ (or $v = (x, u_{f_k}) \in M^*(\pi)$) is unforbidden (UF), if it does not have the tabu status. For a given graph $G(\pi)$, the neighbourhood is searched in the following manner. First, the sets of unforbidden moves are defined

$$UR_k = \{v \in ZR_k^*(\pi) \mid \text{ move } v \text{ is UF}\},$$

$$UL_k = \{v \in ZL_k^*(\pi) \mid \text{ move } v \text{ is UF}\}.$$

For the k-th block, the "best" moves $v_{R(k)} \in UR_k$ and $v_{L(k)} \in UL_k$ are chosen (respectively):

$$DELTA(v_{R(k)}) = \min_{v=(x,u_{l_k}) \in UR_k} \Delta_{ka}(x), \quad k = 1, 2, ..., r,$$
$$DELTA(v_{L(k)}) = \min_{v=(x,u_{f_k}) \in UL_k} \Delta_{kb}(x), \quad k = 1, 2, ..., r.$$

Next, the following sets of moves are created

$$RB = \{v_{R(k)} \mid k = 1, 2, ..., r\},$$

$$LB = \{v_{L(k)} \mid k = 1, 2, ..., r\},$$

and

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

Note that the move $v_k \in BB$ belongs either to RB or to LB. The move v to be performed is selected amongst those in BB with the lowest value of $DELTA(v)$, i.e. $DELTA(v) = \min_{v_k \in BB} DELTA(v_k)$, and which gives the lowest bound on value $C_{max}(\pi_v)$, that is $C_{max}(\pi) + DELTA(v)$ (see Theorems 5.4 and 5.5). If the move v is selected, then the resulting graph $G(\pi_v)$ is created, and a pair of operations corresponding to the move v is added to the tabu list T (see section **Tabu list and tabu status of move** for details). If set BB is empty, then the oldest element of tabu list T is deleted, and the search is repeated until non-empty set BB is found.

3.4 Perturbations

The main handicap of a local search procedure is its myopic nature: it looks only one single move ahead, and any move can lead to a "bad" solution where the search becomes trapped in a local optimum that may be substantially "worse" than the global optimum, even in the tabu search approach where a tabu list is used. In this paper, we use a certain perturbation technique in addition to the tabu list for overcoming this drawback of traditional local search algorithms.

The generic key idea of a perturbation is to consider a search which allows us several moves to be made simultaneously in a single iteration and carry the search to the more promising areas of solution space.

In our algorithm, the set of promising moves can be found as follows

$$BB^{(-)} = \{v_k \in BB \mid DELTA(v_k) < 0\} = \{v_1, v_2, ..., v_z\}, \quad z \leq 2r.$$

The intuition following from Theorems 5.4 and 5.5 suggests that each move $v \in BB^{(-)}$ can provide a graph $G(\pi_v)$ that is "better" than $G(\pi)$. Therefore, as a perturbation, we decided to perform simultaneously **all** moves from $BB^{(-)}$ in $G(\pi)$, obtaining the resulting graph, denoted $G(\pi_{\overline{v}})$, where $\overline{v} = (v_1, v_2, ..., v_z)$. While performing simultaneously all moves from $BB^{(-)}$, the different moves of $BB^{(-)}$ operate in different blocks of $G(\pi)$. Therefore, graph $G(\pi_{\overline{v}})$ is acyclic (it follows from the proofs of Theorems 5.2 and 5.3).

Note that if $|BB^{(-)}| = 1$, then the perturbation is equivalent to the selection from BB the single ("best") move to be performed, thus, in this case, it is not treated as a perturbation. Furthermore, if set $BB^{(-)}$ is empty then the perturbation can not be performed. Therefore, in both cases, the search process is continued (according to the description given in section **Search strategy**) until the graph with $|BB^{(-)}| > 1$ is obtained, and then the perturbation can be made.

If a perturbation is performed, then a pair of operations corresponding to the move v with the smallest value of $DELTA(v)$ is added to tabu list T (see section **Tabu list and tabu status of move** for details).

A perturbation is used when at least $MaxretP$ **consecutive non-improving iterations** pass in the algorithm. More precisely, if graph $G(\pi_{\overline{v}})$ is obtained after

performing a perturbation, then the next one is made when $MaxretP$ of the iterations will pass in TS. In other words, the perturbation is made periodically, where $MaxretP$ is the number of the iterations between the neighbouring ones.

3.5 Algorithm TSGW

In the algorithm, the asterisk (*) refers to the best values found, the zero superscript (^{o}) refers to initial values, and its lack denotes the current values. The algorithm starts from a given initial graph $G(\pi^o)$ (π^o can be found by any algorithm). The algorithm stops when $Maxiter$ iterations have been performed.

INITIALISATION.

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

SEARCHING.

Set $iter := iter + 1$, modify (if it is appropriate) $LengthT$ of the tabu list according to the method described earlier, and for graph $G(\pi)$ create a set of representatives BB.

SELECTION.

If $BB = \emptyset$, then remove the oldest element of the tabu list and go to SEARCHING.

Find the "best" move $v \in BB$, i.e.

$$DELTA(v) = \min_{v_k \in BB} DELTA(v_k),$$

create the graph $G(\pi_v)$, calculate $C_{max}(\pi_v)$, and modify the tabu list according to the method described earlier. If $C_{max}(\pi_v) < C^*$, then save the best values $C^* := C_{max}(\pi_v)$, and $\pi^* := \pi_v$. If $C_{max}(\pi_v) \geq C_{max}(\pi)$, then set $retp := retp + 1$, otherwise set $retp := 0$.

Next set $G(\pi) := G(\pi_v)$.

STOP CRITERIA

If $iter \geq Maxiter$ then STOP.

If $retp < MaxretP$ then go to SEARCHING.

PERTURBATION

For graph $G(\pi)$ create the sets BB and $BB^{(-)}$. If $BB = \emptyset$, then remove the oldest element of the tabu list and go to SEARCHING. Perform the perturbation according to the method described earlier generating graph $G(\pi_{\overline{v}})$, and calculate $C_{max}(\pi_{\overline{v}})$. If $C_{max}(\pi_{\overline{v}}) < C^*$, then save the best

values $C^* := C_{max}(\pi_{\overline{v}})$, $\pi^* := \pi_{\overline{v}}$ and set $retp := 0$. If $|BB^{(-)}| \leq 1$ and $C_{max}(\pi_{\overline{v}}) \geq C_{max}(\pi)$, then set $retp := retp + 1$. If $|BB^{(-)}| \leq 1$ and $C_{max}(\pi_{\overline{v}}) < C_{max}(\pi)$, then set $retp := 0$. If $|BB^{(-)}| > 1$, then set $retp := 0$. Modify the tabu list according to the method described earlier. Next set $G(\pi) := G(\pi_{\overline{v}})$, and go to SEARCHING.

Algorithm TSGW has one tuning parameter $MaxretP$ which is to be chosen experimentally.

4. Computational Results

Algorithm TSGW was coded in C++, run on a personal computer Pentium 333 MHz, and tested on benchmark problems taken from the literature. The results obtained by our algorithm were then compared with results from the literature.

So far, the best approximation algorithms for the job-shop problem with the makespan criterion were proposed in papers by Matsuo, Suh and Sullivan (1988), Laarhoven, Aarts and Lenstra (1992), DellAmico and Trubian (1993), Nowicki and Smutnicki (1996b), Balas and Vazacopoulos (1998), and Pezzela and Merelli (2000). Pezzela and Merelli reported that their algorithm, denoted as TSSB, provides better results than the ones proposed by other authors. Therefore we compare our algorithm TSGW with TSSB, which is also based on the tabu search approach.

Algorithm TSGW, similarly as TSSB, was tested on 133 commonly used problem instances of various sizes and difficulty levels taken from the OR-Library.

(a) Five instances denoted as ORB1–ORB5 with $n \times m = 10 \times 10$ due to Applegate and Cook (1991), three instances FT6, FT10, FT20 with $n \times m = 6 \times 6$, 10×10, 5×20 due to Fisher and Thompson (1963), and five instances ABZ5–ABZ9 with $n \times m = 10 \times 10$, 20×15 due to Adams, Balas and Zawack (1988).

(b) Forty instances of eight different sizes LA01–LA40 with $n \times m = 10 \times 5$, 15×5, 20×5, 10×10, 15×10, 20×10, 30×10, 15×15 due to Lawrence (1984). The optimal solution of the instance LA29 is thus far unknown.

(c) Eighty instances of eight different sizes TA1–TA80 with $n \times m = 15 \times 15$, 20×15, 20×20, 30×15, 30×20, 50×15, 50×20, 100×20 due to Taillard (1993). For this class, the optimal solution is known only 32 out of 80 instances.

The effectiveness of our algorithm was analysed in both terms of CPU time and solution quality. There are some complications involving the speed of computers used in the tests. Algorithm TSGW was run on Pentium 333 MHz, whereas TSSB was run on Pentium 133 MHz. Regarding the speed of the performance, it is becoming very difficult to compare the CPU times of algorithms tested on different computers. An attempt is made to compare the CPU times

for different algorithms using conversion factors for different machines given in a report by Dongarra (2004). Although, the benchmark results reported in Dongarra tests can be used to give a rough estimate on the relative performance of different computers, these results refer to floating-point operations and therefore may not be representative when computations are essentially with integers, as in the case of our algorithms. Besides, the architecture, configurations, cache, main memory and compilers also affect the CPU times. Therefore, in order to avoid discussion about the conversion factors and speed of computers used in the tests, we enclosed for each compared algorithm the original name of computer on which it has been tested, as well as the original running time.

Algorithm TSGW needs an initial solution, which can be found by any heuristic method. In our tests, we use the procedure INSA which is based on an insertion technique, see Nowicki and Smutnicki (1996b). The computational complexity of this heuristic is $O(n^3m^2)$.

At the initial stage, TSGW was run several times, for small-size instances in order to find the proper value of tuning parameter $MaxretP$. This was chosen experimentally as a result of the compromise between the running time and solution quality and we set $MaxretP = 3$.

For each test instance, we collected the following values:

C^A – the makespan found by the algorithm $A \in \{\text{TSGW, TSSB }\}$.

$Time$ – CPU in seconds.

Then two measures of the algorithms quality were calculated

$PRD(A) = 100(C^A - LB)/LB$ – the value (average) of the percentage relative difference between makespan C^A and the best known lower bound LB (or the optimum value OPT, if it is known).

$CPU(A)$ – the computer time (average) of algorithm A.

For TSSB, there are some problems concerning the interpretation of the results in CPU times for the instances of class (c). In the paper of Pezzela and Merelli (2000), it is reported that for each instance, TSSB performs $Maxiter$ iterations equal to $100n$. So that, the average CPU for the instances with size $n \times m = 20 \times 20$ should be shorter than for the ones with $n \times m = 100 \times 20$, whereas in Table 6 of the paper we have found that for the former instances, the CPU is, in approximation, 150 times longer than for the latter ones. Similar problems are in Table 3 of the paper for the instances of class (b).

Therefore, we conclude that for the instances of classes (b) and (c), $Maxiter$ is not equal to $100n$, but it is different for different instances. Instead, the analysis of the results in Table 5.1 for class (a) suggests that there these inconveniences are avoided. Hence, we have assumed that the CPU times of TSSB obtained for both classes (b) and (c) are those for which the C_{max} values (or PRD values)

presented in the paper of Pezzela and Merelli (2000) are reached. And, since these values are reported for each instance, it is possible to detect $Maxiter$ and/or CPU time to be correspondent to the C_{max} value produced by TSSB, for an individual instance.

As a consequence of the above, while testing our algorithm, for each instance of classes (b) and (c), we detect the CPU time at which TSGW has reached the C_{max} value not greater than that obtained by TSSB. Then it was possible to compare the CPU times of the algorithms.

In Table 5.1, we present the results obtained for the test problems of class (a) ORB1–ORB5, FT6, FT10, FT20, and ABZ5–ABZ9. For these instances, TSGW was tested for $Maxiter$ equal to $300n$.

Table 5.1. Detailed results for the problem instances of class (a)

Problem	$n \times m$	OPT or (LB-UB)	TSGW $Maxiter = 300 * n$ C_{max}	PRD	CPU	CPU to opt (or to best)	TSSB $Maxiter = 100 * n$ C_{max}	PRD	CPU
ORB1	10 × 10	1059	1059	0.00	0.9	0.6	1064	0.47	82
ORB2	10 × 10	888	888	0.00	0.9	0.6	890	0.23	75
ORB3	10 × 10	1005	1005	0.00	1.1	0.7	1013	0.80	87
ORB4	10 × 10	1005	1005	0.00	0.8	0.6	1013	0.80	75
ORB5	10 × 10	887	887	0.00	0.9	0.2	887	0.00	81
FT6	6 × 6	55	55	0.00	0.1	0.0	55	0.00	-
FT10	10 × 10	930	930	0.00	1.2	0.2	930	0.00	80
FT20	20 × 5	1165	1165	0.00	2.3	0.7	1165	0.00	115
ABZ5	10 × 10	1234	1236	0.16	1.1	(0.2)	1234	0.00	75
ABZ6	10 × 10	943	943	0.00	1.0	0.2	943	0.00	80
ABZ7	20 × 15	656	656	0.00	14.8	3.8	666	1.52	200
ABZ8	20 × 15	(647-669)	671	3.71	14.6	(5.7)	678	5.12	205
ABZ9	20 × 15	(661-679)	682	3.18	14.9	(3.9)	693	4.84	195
all				0.54				1.06	

CPU represents the CPU time:
TSGW on Pentium 333MHz,
TSSB on Pentium 133MHz (Pezzella and Merelli 2000)

Our algorithm finds an optimal solution to ten out of thirteen problems in relatively very short times. For very famous FT10 with $n \times m = 10 \times 10$, it finds an optimal solution in 0.2 second. Nevertheless, for ABZ5, we could not find any optimal solution reported in the literature, equal to 1234. Besides, note that for $Maxiter$ equal to $300n$, TSGW needs a very small amount of CPU times. The longest CPU time of TSGW is equal to 14.9 seconds (on computer Pentium 333), whereas TSSB needs 205 seconds (on Pentium 133) for $Maxiter$ equal to $100n$. Finally, note that in the terms of PRD values, TSGW produces significantly better results than TSSB.

Table 5.2. Detailed results for the problem instances of class (b)

LA	OPT or (LB-UB)	TSGW $C_{\max}$	TSGW PRD	TSGW CPU	TSSB $C_{\max}$	TSSB PRD	LA	OPT or (LB-UB)	TSGW $C_{\max}$	TSGW PRD	TSGW CPU	TSSB $C_{\max}$	TSSB PRD
10×5							15×10						
1	666	666	0.00	0.0	666	0.00	21	1046	1046	0.00	3.4	1046	0.00
2	655	655	0.00	0.0	655	0.00	22	927	927	0.00	2.7	927	0.00
3	597	597	0.00	0.2	597	0.00	23	1032	1032	0.00	0.2	1032	0.00
4	590	590	0.00	0.0	590	0.00	24	935	936	0.10	0.9	938	0.32
5	593	593	0.00	0.0	593	0.00	25	977	978	0.10	3.7	979	0.20
15×5							20×10						
6	926	926	0.00	0.0	926	0.00	26	1218	1218	0.00	1.0	1218	0.00
7	890	890	0.00	0.0	890	0.00	27	1235	1235	0.00	3.9	1235	0.00
8	863	863	0.00	0.0	863	0.00	28	1216	1216	0.00	4.4	1216	0.00
9	951	951	0.00	0.0	951	0.00	29	1142-1153	1160	1.57	0.9	1168	2.28
10	958	958	0.00	0.0	958	0.00	30	1355	1355	0.00	0.2	1355	0.00
20×5							30×10						
11	1222	1222	0.00	0.0	1222	0.00	31	1784	1784	0.00	0.0	1784	0.00
12	1039	1039	0.00	0.0	1039	0.00	32	1850	1850	0.00	0.0	1850	0.00
13	1150	1150	0.00	0.0	1150	0.00	33	1719	1719	0.00	0.0	1719	0.00
14	1292	1292	0.00	0.0	1292	0.00	34	1721	1721	0.00	0.0	1721	0.00
15	1207	1207	0.00	0.0	1207	0.00	35	1888	1888	0.00	0.1	1888	0.00
10×10							15×15						
16	945	945	0.00	0.6	945	0.00	36	1268	1268	0.00	0.1	1268	0.00
17	784	784	0.00	0.0	784	0.00	37	1397	1411	1.00	2.4	1411	1.00
18	848	848	0.00	3.2	848	0.00	38	1196	1198	0.17	2.4	1201	0.42
19	842	842	0.00	2.1	842	0.00	39	1233	1233	0.00	3.4	1240	0.57
20	902	902	0.00	0.5	902	0.00	40	1222	1225	0.25	4.5	1233	0.90
							all			0.08			0.14

CPU represents the CPU time on Pentium 333MHz.

Table 5.3. Average results for the instance groups of class (b)

Problem	$n \times m$	TSGW PRD (aver.)	TSGW CPU (aver.)	TSSB PRD (aver.)	TSSB CPU (aver.)
LA01-05	10 × 5	0.00	0.1	0.00	9.8
LA06-10	15 × 5	0.00	0.0	0.00	-
LA11-15	20 × 5	0.00	0.0	0.00	-
LA16-20	10 × 10	0.00	1.3	0.00	61.5
LA21-25	15 × 10	0.04	2.2	0.10	115
LA26-30	20 × 10	0.31	2.2	0.46	105
LA31-35	30 × 10	0.00	0.0	0.00	-
LA36-40	15 × 15	0.28	2.6	0.58	141
all		0.08		0.14	

CPU represents the CPU time:
TSGW on Pentium 333MHz,
TSSB on Pentium 133MHz (Pezzella and Merelli 2000)

Table 5.4. Detailed results for the problem instances of class (c)

TA	OPT or (LB-UB)	TSGW C_{max}	TSGW PRD	TSGW riptsize	TSGW CPU	TSSB C_{max}	TSSB PRD	TA	OPT or (LB-UB)	TSGW C_{max}	TSGW PRD	TSGW CPU	TSSB C_{max}	TSSB PRD
15×15								30×20						
1	1231	1239	0.649		7.9	1241	0.812	41	1859-2023	2033	9.359	3.9	2045	10.005
2	1244	1244	0.000		7.2	1244	0.000	42	1867-1961	1976	5.839	3.2	1979	5.999
3	1218	1218	0.000		4.7	1222	0.328	43	1809-1879	1898	4.920	8.2	1898	4.920
4	1175	1175	0.000		6.6	1175	0.000	44	1927-1998	2031	5.397	44.1	2036	5.656
5	1224	1228	0.327		9.6	1229	0.408	45	1997-2005	2021	1.202	7.1	2021	1.202
6	1238	1238	0.000		9.4	1245	0.565	46	1940-2029	2046	5.464	6.6	2047	5.515
7	1227	1227	0.000		4.5	1228	0.081	47	1789-1913	1937	8.272	4.4	1938	8.329
8	1217	1218	0.082		9.1	1220	0.246	48	1912-1971	1986	3.870	9.2	1996	4.393
9	1274	1287	1.020		9.7	1291	1.334	49	1915-1984	2007	4.804	6.9	2013	5.117
10	1241	1249	0.645		7.1	1250	0.725	50	1807-1937	1971	9.076	5.7	1975	9.297
20×15								50×15						
11	1321-1364	1370	3.709		7.5	1371	3.785	51	2760	2760	0.000	1.7	2760	0.000
12	1321-1367	1376	4.164		3.9	1379	4.391	52	2756	2756	0.000	3.2	2756	0.000
13	1271-1350	1355	6.609		4.7	1362	7.160	53	2717	2717	0.000	5.7	2717	0.000
14	1345	1345	0.000		7.6	1345	0.000	54	2839	2839	0.000	3.1	2839	0.000
15	1293-1342	1355	4.795		10.1	1360	5.182	55	2679	2681	0.075	5.3	2684	0.187
16	1300-1362	1369	5.307		11.9	1370	5.385	56	2781	2781	0.000	7.6	2781	0.000
17	1458-1464	1477	1.303		4.7	1481	1.578	57	2943	2943	0.000	3.8	2943	0.000
18	1369-1396	1418	3.579		6.9	1426	4.164	58	2885	2885	0.000	2.8	2885	0.000
19	1276-1341	1350	5.800		9.1	1351	5.878	59	2655	2655	0.000	4.1	2655	0.000
20	1316-1353	1361	3.419		4.8	1366	3.799	60	2723	2723	0.000	3.6	2723	0.000
20×20								50×20						
21	1539-1645	1658	7.739		12.0	1659	7.797	61	2868	2868	0.000	3.9	2868	0.000
22	1511-1601	1620	7.213		13.9	1623	7.412	62	2869-2872	2937	2.370	3.4	2942	2.544
23	1472-1558	1567	6.454		18.8	1573	6.861	63	2755	2755	0.000	5.8	2755	0.000
24	1602-1651	1656	3.371		11.9	1659	3.558	64	2702	2702	0.000	10.7	2702	0.000
25	1504-1597	1604	6.649		14.0	1606	6.782	65	2725	2725	0.000	2.7	2725	0.000
26	1539-1651	1666	8.252		16.7	1666	8.252	66	2845	2845	0.000	4.6	2845	0.000
27	1616-1687	1693	4.765		22.1	1697	5.012	67	2825	2861	1.274	46.1	2865	1.416
28	1591-1615	1622	1.948		32.2	1622	1.948	68	2784	2784	0.000	7.3	2784	0.000
29	1514-1625	1635	7.992		57.0	1635	7.992	69	3071	3071	0.000	4.8	3071	0.000
30	1473-1585	1602	8.758		5.9	1614	9.572	70	2995	2995	0.000	2.7	2995	0.000
30×15								100×20						
31	1764	1769	0.283		11.1	1771	0.397	71	5464	5464	0.000	4.8	5464	0.000
32	1774-1803	1836	3.495		17.3	1840	3.720	72	5181	5181	0.000	3.0	5181	0.000
33	1778-1796	1831	2.981		25.2	1833	3.093	73	5568	5568	0.000	3.6	5568	0.000
34	1828-1832	1842	0.766		46.9	1846	0.985	74	5339	5339	0.000	4.3	5339	0.000
35	2007	2007	0.000		7.9	2007	0.000	75	5392	5392	0.000	5.8	5392	0.000
36	1819	1820	0.055		14.7	1825	0.330	76	5342	5342	0.000	7.1	5342	0.000
37	1771-1784	1808	2.089		23.3	1813	2.372	77	5436	5436	0.000	3.0	5436	0.000
38	1673-1677	1694	1.255		17.6	1697	1.435	78	5394	5394	0.000	2.8	5394	0.000
39	1 795	1812	0.947		19.2	1815	1.114	79	5358	5358	0.000	3.5	5358	0.000
40	1631-1686	1724	5.702		17.2	1725	5.763	80	5183	5183	0.000	6.5	5183	0.000
								all			2.30			2.43

CPU represents the CPU time on Pentium 333MHz.

Tables 5.2 and 5.3 report the computational results for the Lawrence's test problems (LA01–LA40) of class (b). Table 5.2 shows the detailed results obtained for each instance tested. Instances LA01–LA15 and LA31–LA35 are "easy" because the number of jobs is several times larger than the number of machines. They were solved to optimality by TSGW in less than 0.4 seconds. The more difficult instances LA16–LA30 and LA36–LA40 were solved in less than 4.5 seconds.

Table 5.3 lists the average results for each size (group) $n \times m$ of the instances. For all groups, CPU times of TSGW are very small (on the average). And so, for group with the largest instances LA36–LA40, TSGW needs 2.6 seconds (on Pentium 333), whereas TSSB needs 141 seconds (on Pentium 133). While, for group with the smallest instances LA01–LA05, the respective CPU times are 0.1 and 9.8 seconds. Besides, note that in the terms of PRD values, TSGW produces substantially better results than TSSB.

Tables 5.4 and 5.5 present the results on 80 test problems of class (c) proposed by Taillard (TA01–TA80). It is reported that for 32 out of 80 instances optimal solutions are not known.

Table 5.4 lists detailed results for TA01–TA80. Instances TA51–TA80 are "easy" because the number of jobs is several times larger than the number of machines. Most of them (i.e. 28 out of 30) were solved to optimality by TSGW in less than 10 seconds. For more difficult instances TA31–TA40 and TA41–TA50, the best C_{max} values of TSSB were produced by TSGW in less than 50 seconds. While, for the most difficult instances TA21–TA30 the values were produced in less than 60 seconds. Most of them (i.e. 8 out of 10) were obtained in less than 25 seconds. The longest CPU is reached for TA29 and is equal to 57 seconds.

Table 5.5. Average results for the instance groups of class (c)

Problem	$n \times m$	TSGW		TSSB	
		PRD (aver.)	CPU (aver.)	PRD (aver.)	CPU (aver.)
TA01-10	15 × 15	0.27	7.6	0.45	2175
TA11-20	20 × 15	3.87	7.1	4.13	2526
TA21-30	20 × 20	6.31	20.4	6.52	34910
TA31-40	30 × 15	1.75	20.1	1.92	14133
TA41-50	30 × 20	5.82	9.9	6.04	11512
TA51-60	50 × 15	0.01	4.1	0.02	421
TA61-70	50 × 20	0.36	9.2	0.39	6342
TA71-80	100 × 20	0.00	4.4	0.00	231
all		2.30		2.43	

CPU represents the CPU time:
TSGW on Pentium 333MHz,
TSSB on Pentium 133MHz (Pezzella and Merelli 2000)

Finally, Table 5.5 shows the average results for each size (group) $n \times m$ of instances. For all groups, CPU times of TSGW are extremely small (on the average). And so, for group with the smallest instances TA01–TA10, our algorithm needs 7.6 seconds (on Pentium 333), whereas TSSB needs 2175 seconds (on Pentium 133). While, for the most difficult group TA21-TA30, the respective CPU times are 20.4 and 34910 seconds. Besides, It is noteworthy that in the terms of PRD values, TSGW produces slightly better results than TSSB.

All these results confirm the favorable performance of TSGW in the terms of CPU times and PRD values as well.

5. Conclusions

In this paper we have presented and discussed some new properties of blocks in the job-shop problem. These properties allow us to propose a new, very fast algorithm based on the tabu search approach. In order to decrease the computational effort for the search in TS, we propose calculation of the lower bounds on the makespans instead of computing makespans explicitly for use in selecting the best solution. These lower bounds are used to evaluate the moves for selecting the "best" one. Also, we propose a tabu list with dynamic length which is changed cyclically as the current iteration number of TS increases, using a "pick" in order to carry the search to another area of the solution space. Finally, some perturbations associated with block properties are periodically applied. Computational experiments are given and compared with the results yielded by the best algorithms discussed in the literature. These results show that the algorithm proposed provides much better results than the recent modern approaches. A particular superiority of our algorithm is observed for so-called "hard" problems for which the number of jobs is close to the number of machines. Nevertheless, some improvements in our algorithm are possible. For instance, attempts to refine the lower bounds and perturbations may induce a further reduction of the computational times.

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

Acknowledgements

This research was supported by KBN Grant 4 T11A 016 24. The authors are grateful to Cesar Rego and the referees for their useful comments and suggestions.

References

Aarts, E. and J.K. Lenstra (1997) Local Search in Combinatorial Optimization. Wiley, New York.

Adams, J., E. Balas and D. Zawack (1988) "The Shifting Bottleneck Procedure for Job-Shop Scheduling," *Management Science*, 34(6):391–401.

Applegate, D. and W. Cook (1991) "A Computational Study of the Job-Shop Scheduling Problem," *ORSA Journal of Computing*, 3:149–156.

Balas, E. (1969) "Machine Sequencing via Disjunctive Graphs: An Implicit Enumeration Algorithm," *Operations Research*, 17:941–957.

Balas, E. and A. Vazacopoulos (1998) "Guided Local Search with Shifting Bottleneck for Job-Shop Scheduling," *Management Science*, 44(2):262–275.

Carlier, J. (1982) "The One-Machine Sequencing Problem," *European Journal of Operational Research*, 1:42–47.

Carlier, J. and E. Pinson (1989) "An Algorithm for Solving the Job Shop Problem," *Management Science*, 35:164–176.

Dongarra, J.J. (2004) Performance of Various Computers using Standard Linear Equations Software. Working paper. Computer Science Department, University of Tennessee, USA. http://www.netlib.org/benchmark/performance.ps.

DellAmico, M. and M. Trubian (1993) "Applying Tabu Search to the Job-Shop Scheduling Problem," *Annals of Operations Research*, 4:231–252.

Fisher, H. and G.L. Thompson (1963) Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules. In J.F. Muth, G.L. Thompson, Editors, Industrial Scheduling, Prencite-Hall, Englewood Cliffs, New York.

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. (1979) Generalized problems of operations sequencing in the discrete production systems. (Polish), Monographs 9, Scientific Papers of the Institute of Technical Cybernetics of Wroclaw Technical University.

Grabowski, J. (1980) "On Two-Machine Scheduling with Release and Due Dates to Minimize Maximum Lateness," *Opsearch*, 17:133–154.

Grabowski, J. (1982) A new Algorithm of Solving the Flow-Shop Problem. In G. Feichtinger and P. Kall, Editors, *Operations Research in Progress*, Reidel Publishing Company, Dordrecht, 57–75.

Grabowski, J., E. Skubalska and C. Smutnicki (1983) "On Flow-Shop Scheduling with Release and Due Dates to Minimize Maximum Lateness," *Journal of the Operational Research Society*, 34:615–620.

Grabowski, J., E. Nowicki and S. Zdrzalka (1986) "A Block Approach for Single Machine Scheduling with Release Dates and Due Dates," *European Journal of Operational Research*, 26:278–285.

Grabowski, J. and J. Janiak (1987) "Job-Shop Scheduling with Resource-Time Models of Operations," *European Journal of Operational Research*, 28:58–73.

Grabowski, J., E. Nowicki and C. Smutnicki (1988) Block Algorithm for Scheduling of Operations in Job-Shop System. (Polish), *Przeglad Statystyczny*, 35:67–80.

Grabowski, J. and J. Pempera (2001) New Block Properties for the Permutation Flow-Shop Problem with Application in TS. *Journal of the Operational Research Society*, 52:210–220.

Internet, http://mscmga.ms.ic.ac.uk/info.html.

Laarhoven, P.V., E. Aarts and J.K. Lenstra (1992) "Job-Shop Scheduling by Simulated Annealing," *Operations Research*, 40:113–125.

Lawrence, S. (1984) Supplement to "Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques," Technical Report, GSIA, Carnegie Mellon University.

Matsuo, H., C.J. Suh and R.S. Sullivan (1988) Controlled Search Simulated Annealing Method for the General Job-Shop Scheduling Problem. Working Paper 03-04-88, Department of Management, Graduate School of Business, The University of Texas at Austin.

Morton, T. and D. Pentico (1993) *Heuristic Scheduling Systems*. Wiley, New York.

Nowicki, E. and C. Smutnicki (1996a) "A Fast Tabu Search Algorithm for the Permutation Flow-Shop Problem," *European Journal of Operational Research*, 91:160–175.

Nowicki, E. and C. Smutnicki (1996b) "A Fast Tabu Search Algorithm for the Job-Shop Problem," *Management Science*, 42(6):97–813.

Pezzella, F. and E. Merelli (2000) "A Tabu Search Method Guided by Shifting Bottleneck for the Job-Shop Scheduling Problem". *European Journal of Operational Research*, 120:297–310.

Smutnicki, C. (1998) A Two-Machine Permutation Flow-Shop Scheduling with Buffers. *OR Spectrum*, 20:229–235.

Taillard, E. (1993) "Benchmarks for Basic Scheduling Problems," *European Journal of Operational Research*, 64:278–285.

Vaessens, R., E. Aarts and J.K. Lenstra (1996) "Job Shop Scheduling by Local Search," *INFORMS Journal of Computing*, 8:303–317.

Zdrzalka, S. and J. Grabowski (1989) "An Algorithm for Single Machine Sequencing with Release Dates to Minimize Maximum Cost," *Discrete Applied Mathematics*, 23:73–89.