Journal of Scheduling **8:** 145–159, 2005.

AN ADVANCED TABU SEARCH ALGORITHM FOR THE JOB SHOP PROBLEM

EUGENIUSZ NOWICKI AND CZESŁAW SMUTNICKI

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

ABSTRACT

The job shop scheduling problem with the makespan criterion is a certain NP-hard case from OR theory having excellent practical applications. This problem, having been examined for years, is also regarded as an indicator of the quality of advanced scheduling algorithms. In this paper we provide a new approximate algorithm that is based on the big valley phenomenon, and uses some elements of so-called path relinking technique as well as new theoretical properties of neighbourhoods. The proposed algorithm owns, unprecedented up to now, accuracy, obtainable in a quick time on a PC, which has been confirmed after wide computer tests.

KEY WORDS: job shop scheduling, makespan, tabu search, path relinking

1. INTRODUCTION

The job shop scheduling problem with the makespan criterion comes from the practice of OR and models many real production processes very well on the basis of the flow of tasks. The problem has been known for years in the scheduling theory as a particularly hard combinatorial optimisation case. Although in the last three decades a lot of effort has been put into research in this area, good results have appeared quite recently, see state-of-the-art papers by Błażewicz, Domschke, and Pesch (1996), Vaessens, Aarts, and Lenstra (1996), and Jain and Meeran (1999), supplemented by a few recent studies (Jain, Rangaswamy, and Meeran, 2000; Pezella and Merelli, 2000; Nowicki and Smutnicki, 2001).

Up to the eighties, the majority of the research focused on optimisation methods, among which B&B scheme played the central role. At the beginning of the nineties, it became evident that pure optimisation methods reached the limit; they cannot solve instances larger than 250 operations within a reasonable time (hours, days). Following this, approximate approaches (developed so far in the background) became foreground tasks. Many studies have been carried out to improve efficacy of these algorithms, measured by the accuracy being opposed to the running time. We refer here to various GA, TS, SA approaches, see review part in Vaessens, Aarts, and Lenstra (1996), and CS methods given in Dorndorf, Pesch, and Phan-Huy (2000, 2002).

Tabu search (TS) approach, proposed by Glover (Glover and Laguna, 1997), seems to be especially promising for the problem considered. Among few individual TS methods, algorithm TSAB (Nowicki and Smutnicki, 1996), designed originally in 1993, introduced a real breakthrough in thinking about efficient approximate approaches for the job shop problem because it offers a simple implementation, very short running time and good accuracy.

Correspondence to: Czesław Smutnicki. E-mail: smutnick@ict.pwr.wroc.pl

Although much has been done, the quality of solutions obtainable in a reasonable time remains unsatisfactory. In this paper we provide new algorithm i-TSAB, which by the use of *big valley* phenomenon, some elements of *path relinking* philosophy, and new theoretical properties offers unprecedented, until now, accuracy in a time span of minutes on a PC. Proved properties are general enough to be applied in many other algorithms. As the immediate practical result of the proposed approach, we have found 91 better upper bounds among 112 yet unsolved instances from the common benchmark set, attacked by all job shop algorithms designed up to now all over the world.

The paper is organized as follows. After the presentation of the problem, its model and denotations (Section 2) are discussed, followed by introduction of some theoretical properties (Section 3). New algorithm i-TSAB has been shown in Section 4, its implementation and tuning in Section 5, whereas its comparison with other algorithms in Section 6.

2. PROBLEM, MODEL AND NOTATIONS

The job shop problem is defined formally as follows. There is a set of jobs $J = \{1, \ldots, n\}$, a set of machines $M = \{1, \ldots, m\}$ and a set of operations $O = \{1, \ldots, o\}$. Set O is decomposed into subsets corresponding to the jobs. Job j consists of a sequence of o_j operations indexed consecutively by $(l_{j-1} + 1, \ldots, 1_{j-1} + o_j)$ which should be processed in that order, where $l_j = \sum_{i=1}^{j} o_i$ is the total number of operations of the first j jobs, $j = 1, \ldots, n$ ($l_0 = 0$) and $\sum_{i=1}^{n} o_i = o$. Operation i must be processed on a machine $\mu_i \in M$ during an uninterrupted processing time $p_i > 0$, $i \in O$. Thus, the set of operations O can by naturally decomposed into subsets $M_k = \{i \in O : \mu_i = k\}$, each of which corresponding to operations that should be processed on machine k; let $m_k = |M_k|$, $k \in M$. We assume that any successive operations of the same job are going to be processed on different machines. Each machine can process one operation at a time, at most. A *feasible schedule* is defined by starting times $S_i \geq 0$, $i \in O$, such that the above constraints are satisfied. The problem is to find a feasible schedule that minimizes the makespan $\max_{i \in O}(S_i + p_i)$.

In this paper we refer to the permutation-and-graph model, introduced originally by us in Nowicki and Smutnicki (1996), which is simpler and of smaller size than the commonly used disjunctive graph model of Roy and Sussman (1964). In our model, the schedule is represented by the *processing order* of operations on machines, i.e. by m-tuple $\pi = (\pi_1, \ldots, \pi_m)$, where $\pi_k = (\pi_k(1), \ldots, \pi_k(m_k))$ is a permutation on M_k, $k \in M$; $\pi_k(i)$ denotes the element of M_k, which is in position i in π_k. Let Π_k be the set of all permutations on M_k, then $\pi \in \Pi = \Pi_1 \times \Pi_2 \times \cdots \times \Pi_m$. For the processing order π, we create a direct graph $G(\pi) = (O, R \cup E(\pi))$ with a set of nodes O and a set of arcs $R \cup E(\pi)$, where $R = \bigcup_{j=1}^{n} \bigcup_{i=1}^{o_j - 1} \{(l_{j-1} + i, l_{j-1} + i + 1)\}$ and $E(\pi) = \bigcup_{k=1}^{m} \bigcup_{i=1}^{m_k - 1} \{(\pi_k(i), \pi_k(i+1))\}$. 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. Each node $i \in O$ in the digraph has weight p_i and each arc has weight zero.

The processing order $\pi \in \Pi$, such that graph $G(\pi)$ does not contain a cycle, will be called the *feasible processing order*. For the feasible processing order $\pi \in \Pi$, we denote the length of the longest path in $G(\pi)$ among the paths going to node i (including the node weight) by $r_\pi(i)$. By symmetry, we denote the length of the longest path among the paths going out from node i (including the node weight) by $q_\pi(i)$. It is well known that the schedule $S_i = r_\pi(i) - p_i$, $i \in O$ (represented by π) is the feasible schedule. Makespan $C_{\max}(\pi)$ for this schedule equals the length

of the longest path (critical path) 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 sequel, we also use the alternative representation of $G(\pi)$. For operation i, we denote its job-successor by $Aj(i)$ (i.e. the next operation to i in the job to which operation i belongs) and job-predecessor by $Bj(i)$. By symmetry, we denote the the machine-successor (next operation processed on the same machine that operation i) by $Am_\pi(i)$ and machine-predecessor of operations i by $Bm_\pi(i)$. We complete these definitions by putting zero if the required operation does not exist. For operation i, the sets of successors and predecessors are denoted briefly by $B_\pi(i) = \{Bj(i), Bm_\pi(i)\}$ and $A_\pi(i) = \{Aj(i), Am_\pi(i)\}$.

3. NEW PROPERTIES OF NEIGHBOURHOODS

We start from the most commonly used technique of generating neighbours in local search (LS) methods, see Laarhoven, Aarts, and Lenstra (1992) and Taillard (1994). Let π be the feasible processing order, and $u^\pi = (u_1^\pi, \ldots, u_s^\pi)$, where $u_i^\pi \in O$, be the critical path in $G(\pi)$ (s is the number of nodes in this path). We refer to neighbourhoods generated by *swap moves* performed from the processing order π, in such sense that two *adjacent operations* $(x, y) = v$, $x, y \in u^\pi$, on a machine from the critical path u^π in $G(\pi)$ are swapped, providing the new processing order, denoted hereinafter by $\pi_{(v)}$. This technique uses so-called *elimination property of the critical path* and ensures feasibility for all resulting processing orders. We denote by $N(\pi)$ the set of all such moves from π, thus processing orders $\mathcal{N}(\pi) = \{\pi_{(v)} : v \in N(\pi)\}$ constitute the *basic swap neighbourhood* of π.

Repetitive process, which chooses the solution by a neighbourhood exploration, is usually the most time-consuming part of LS algorithms; this refers to TS as well as to other methods, as an example SA. Total calculation cost depends on the number of neighbours and the amount of calculations per neighbour. To reduce them several approaches have been proposed. For example, our algorithm TSAB from Nowicki and Smutnicki (1996) uses so-called *reduced* set of moves $N^*(\pi) \subseteq N(\pi)$, $|N^*(\pi)| \ll |N(\pi)|$ (reduced neighbourhood $\mathcal{N}^*(\pi) \subseteq \mathcal{N}(\pi)$, respectively), which refers to *block properties* from Grabowski, Nowicki, and Smutnicki (1988). Other authors have evaluated neighbours by inexpensive lower bound, rather than by the makespan, see for example Taillard (1994) and Ten Eikelder et al. (1999). Unfortunately, it has been found in Nowicki and Smutnicki (2001) that the latter technique frequently deteriorates the quality of the algorithm. Therefore, we prefer a completely new approach heading towards the reduction of complexity $O(o)$ stemmed from the standard method of makespan calculation.

In the sequel we will provide new properties applicable to $\mathcal{N}(\pi)$, where $\pi \in \Pi$ is a feasible processing order. To this order we introduce two additional notions for the given feasible $\tau \in \Pi$, graph $G(\tau)$ and any two of its nodes a, b. By $P_\tau(a \vee b)$ and $P_\tau(\bar{a})$ we denote the set of all paths 'containing nodes a or b' and 'not containing node a', respectively. For the given set of paths P, we denote the length of the longest path in P by $D[P]$.

Property 1. For any $(\sigma) = \pi_{(v)}$, *where* $v = (x, y) \in N(\pi)$, *we have*

$$C_{\max}(\sigma) = \max\{D[P_\sigma(x \vee y)], \quad D[P_\pi(\bar{x})]\}. \tag{1}$$

Proof of the property is given in Appendix A.

Equation (1) states that $C_{\max}(\sigma)$ can be calculated using two components. The former term $D[P_\sigma(x \vee y)]$ follows from 'new' graph $G(\sigma)$ and is the lower bound (further denoted by LB^T) proposed by Taillard (1994). He showed that $LB^T = D[P_\sigma(x \vee y)]$ can be calculated in a time $O(1)$, using values $r_\pi(i), q_\pi(i), i \in O$, on the basis of the following equation

$$LB^T = \max\{\Delta + q_\pi(Aj(y)), \quad \max\{\Delta, r_\pi(Bj(x))\} + p_x + \max\{q_\pi(Aj(x)), q_\pi(Am_\pi(y))\}\} \quad (2)$$

where

$$\Delta = \max\{r_\pi(Bj(y)), r_\pi(Bm_\pi(x))\} + p_y.$$

The latter term in (1) derives from the 'old' graph $G(\pi)$. Its calculation is slightly more complex, but seldom needs to be performed. Indeed, if $LB^T \geq C_{\max}(\pi)$, then from the inequality $C_{\max}(\pi) \geq D[P_\pi(\bar{x})]$, we have $C_{\max}(\sigma) = LB^T$, thus second term mentioned in (1) can be skipped. Otherwise, if $LB^T < C_{\max}(\pi)$, then its use is necessary.

The subsequent property supports finding the second term in (1). It employs the list $F_\pi = (F_\pi(1), \ldots, F_\pi(o))$ of all operations from O, being the *topological order* of nodes from graph $G(\pi)$. Precisely, we require $f_\pi(a) < f_\pi(b)$ for any pair $(a, b) \in R \cup E(\pi)$, where $f_\pi(i)$ is the position in F_π occupied by the operation i; $F_\pi(f_\pi(i)) = i, i \in O$.

Property 2. Let F_π denotes the topological order of nodes from $G(\pi)$. Let $FJ = \{j \in O : Bj(j) = O\}$ and $LJ = \{j \in O : Aj(j) = 0\}$ be sets of the first and last operations of jobs, respectively. For any $\sigma = \pi_{(v)}$, $v = (x, y) \in N(\pi)$, we have

$$D[p_\pi(\bar{x})] = \max\{RQ, R', R'', Q', Q''\}, \quad (3)$$

where

$$RQ = \max\{r_\pi(a) + q_\pi(b) : f_\pi(a) < f_\pi(x) < f_\pi(b), (a, b) \in R \cup E(\pi)\},$$
$$R' = \max\{r_\pi(a) : f_\pi(a) < f_\pi(x), a \in LJ\}, \quad R'' = \max\{r_\pi(a) : a \in B_\pi(x)\},$$
$$Q' = \max\{q_\pi(a) : f_\pi(x) < f_\pi(a), a \in FJ\}, \quad Q'' = \max\{q_\pi(a) : a \in A_\pi(x)\}$$

and additionally

$$D[P_\sigma(x \vee y)] = LB^T \geq \max\{R'', Q''\}. \quad (4)$$

Proof of the property is given in Appendix B.

From Properties 1 and 2 we conclude that the makespan $C_{\max}(\sigma), \sigma = \pi_{(v)}$, for fixed move $v = (x, y) \in N(\pi)$, can be found using the following formula

$$C_{\max}(\sigma) = \begin{cases} LB^T, & \text{if } LB^T \geq C_{\max}(\pi), \\ \max\{LB^T, RQ, R', Q'\}, & \text{otherwise.} \end{cases} \quad (5)$$

Because of (5) we can formulate the following surprising theorem.

Theorem 1. Having $r_\pi(i), q_\pi(i), i \in O$ and F_π, the makespan $C_{\max}(\sigma)$ for fixed $\sigma \in \Pi, \sigma = \pi_{(v)}$, $v = (x, y) \in N(\pi)$, can be found in the time $O(\max\{\sum_{j=1}^{n} \log o_j, \sum_{k=1}^{m} \log m_k\})$.

Proof of the theorem is given in Appendix C.

From the theorem and its proof it follows that the exact value of $C_{\max}(\sigma)$ can be found in a time strictly less than $O(o)$; please note, we have $o = \sum_{j=1}^{n} o_j = \sum_{k=1}^{m} m_k$. This result can be immediately applied in TS or SA, if only one uses a neighbourhood included in $\mathcal{N}(\pi)$. Indeed, assuming that in order to select a solution from the neighbourhood $\mathcal{N}(\pi)$, we have to find $h \leq |\mathcal{N}(\pi)|$ makespans $C_{\max}(\sigma)$, the running time has been shortened from $O(h{\cdot}o)$ to $O(h{\cdot}\max\{\sum_{j=1}^{n} \log o_j, \sum_{k=1}^{m} \log m_k\})$. We have tested benefits from Theorem 1 using algorithm TSAB, on instances from Taillard (1993), having $o_j = m$, $j = 1, \ldots, n$, $m_k = n$, $k = 1, \ldots, m$, $n \geq m$. For them, the complexity of finding the best solution in the neighbourhood gets reduced theoretically from $O(h{\cdot}nm)$ to $O(h{\cdot}n \log m)$, where $h = |N^*(\pi)|$. In experiments, the running time was reduced 8 times for instances with $n = 15$, $m = 15$, 11.8 times for instances with $n = 20$, $m = 20$ and 13.6 times for instances with $n = 30$, $m = 20$.

4. ALGORITHM *i*-TSAB

The positive correlation between the distance among local extremes and the makespan value for all Taillard's instances has been detected (Nowicki and Smutnicki, 2001). This fact suggests the presence of *big valley* (BV) phenomenon in the solution space. BV comprises the best *elite solutions* dispersed over its area, constitutes an unusually small part of this space, and contains a huge number of solutions. Then, we are looking for a method with at least: (1) diversification, on the basis of various initial solutions, broadly and uniformly dispersed over BV area; (2) tracing, to localize the centre of BV, probably close to the global minimum; (3) perfect exploration, of any subarea of BV, concentrated around a chosen initial solution.

We propose algorithm *i*-TSAB satisfying desired requirements; its general idea follows from our paper, Nowicki and Smutnicki (2001). Goal (1) is realized by the special generator of initial solutions, based on some elements of path relinking philosophy (Glover and Laguna, 1997). Goal (2) by an original method of updating the collection of elite solutions. Formally, for aim (3), one can use any LS method; however, we recommend TSAB due to its amazing exploration properties.

Shortly speaking, the proposed algorithm *i*-TSAB operates on the set of dispersed elite solutions from BV, further transformed by New Initial Solution (NIS) generator, to provide successive solutions for local explorations of BV with the help of TSAB.

4.1. New initial solution generator

Each new initial solution ϕ is generated on the basis of some two elite solutions γ, δ and the *reference makespan* C^R. The value C^R denotes the best makespan found during the search; we have $C^R \leq \min\{C_{\max}(\gamma), C_{\max}(\delta)\}$. The generator has the form of function *NIS*, shown in Fig. 1. Its fundamental aim is to provide the solution $\phi = NIS(\gamma, \delta, C^R)$, located 'between' γ and δ, to run local exploration (TSAB in our case). In order to control the distance between any two solutions α and β, we use the measure $D(\alpha, \beta)$ to represent a minimal number of swap moves of adjacent operations on a machine (not necessarily from the critical path) to get β from α. It is already known that

$$D(\alpha, \beta) = |\{(x, y) \in E^T(\alpha) : (y, x) \in E^T(\beta)\}|, \tag{6}$$

Function, for two processing orders γ, δ and the reference makespan C^R, returns the processing order ϕ as $NIS(\gamma, \delta, C^R)$ (located between γ and δ) and updates the reference makespan C^R.

Set $\pi := \gamma$ and $iter := 0$. Find δ^{-1} and $D(\gamma, \delta)$. Construct $G(\pi)$. Find $r_\pi(j)$, $q_\pi(j)$, $j \in O$, F_π and u^π.

repeat

Set $iter := iter + 1$. Find $N(\pi)$ on the basis of u^π.

For any $v \in N(\pi)$ calculate and store $C_{\max}(\pi_{(v)})$ using $r_\pi(j)$, $q_\pi(j)$, $j \in O$, F_π and equation (5).

Find $N^+ = \{v = (x, y) \in N(\pi) : \delta^{-1}(y) < \delta^{-1}(x)\}$. Set $K := N^+$, if $N^+ \neq \emptyset$ and $K := N(\pi)$, otherwise. Select the move $w \in K$ such, that $C_{\max}(\pi_{(w)}) = \min_{v \in K} C_{\max}(\pi_{(v)})$. Denote $\pi_{(w)}$ by α.

Modify: (*i*) the graph $G(\pi)$ to get $G(\alpha)$; (*ii*) the list F_π to get F_α and (*iii*) the $r_\pi(j)$, $q_\pi(j)$, $j \in O$ to get $r_\alpha(j)$, $q_\alpha(j)$, $j \in O$ on the basis of F_α. Find u^α on the basis of $q_\alpha(j)$, $j \in O$. Set $\pi := \alpha$ and $\phi := \pi$.

If $C_{\max}(\pi) < C^R$ then set $C^R := C_{\max}(\pi)$ and EXIT.

until $iter \geq maxV \cdot D(\gamma, \delta)$.

Figure 1. Function $NIS(\gamma, \delta, C^R)$.

where $E^T(\alpha)$ and $E^T(\beta)$ are transitive closure of $E(\alpha)$ and $E(\beta)$, respectively. The measure (6) can be written in the equivalent form, more suitable for calculations,

$$D(\alpha, \beta) = |\{(x, y) \in E^T(\alpha) : \beta^{-1}(y) < \beta^{-1}(x)\}|, \tag{7}$$

where $\beta^{-1}(z)$ denotes the position of operation z in permutation β_{μ_z}; $\beta_{\mu_z}(\beta^{-1}(z)) = z$.

Function $NIS(\gamma, \delta, C^R)$ begins from the processing order $\pi = \gamma$ and performs a number of iterations, using neighbourhood $\mathcal{N}(\pi)$. At iteration $iter \geq 1$, we find for processing order π, the move set $N(\pi)$ and makespans $C_{\max}(\pi_{(v)})$, $v \in N(\pi)$. Next, it creates the auxiliary move set $N^+ = \{v = (x, y) \in N(\pi) : \delta^{-1}(y) < \delta^{-1}(x)\}$. By (7), each move $v \in N^+$ directs $\pi_{(v)}$ toward the processing order δ, i.e. $D(\pi_{(v)}, \delta) = D(\pi, \delta) - 1$, whereas each move $v \in N(\pi) \backslash N^+$ directs $\pi_{(v)}$ backward, i.e. $D(\pi_{(v)}, \delta) = D(\pi, \delta) + 1$. Since the set N^+ can be empty, we propose to use in the selection set K, defined as follows: $K := N^+$, if $N^+ \neq \emptyset$ and $K := N(\pi)$, otherwise. *NIS* chooses the move $w \in K$, such that $C_{\max}(\pi_{(w)}) = \min_{v \in K} C_{\max}(\pi_{(v)})$ and, then, putting $\pi := \pi_{(w)}$ it goes to the next iteration. Function stops at the iteration $iter = \lfloor maxV \cdot D(\gamma, \delta) \rfloor$ and returns the current processing order $\phi = \pi$; value $0 < maxV < 1$ is a tuning parameter. The case $N^+ = \emptyset$ appeared rarely in experiments; thus, in practice, we have $D(\gamma, \phi) \approx maxV \cdot D(\gamma, \delta)$ and $D(\phi, \delta) \approx (1 - maxV) \cdot D(\gamma, \delta)$.

Formally, the function *NIS* can perform less than $\lfloor maxV \cdot D(\gamma, \delta) \rfloor$ iterations, if only for some *iter* we found $C_{\max}(\pi) < C^R$. However, this case was observed unusually seldom, which means that the probability of finding a better solution *directly* on the path from one solution to another is very small, even though this path links elite solutions.

Algorithm starts with a processing order π^0 provided by INSA, returns the best found processing order π^* and its makespan C^*.

Set $(\pi^1, C^1) := TSAB(\pi^0)$ and $C^* := C^1$. For $i = 2, \ldots, maxE$ do: set $\phi := NIS(\pi^{i-1}, \pi^0, C^*)$, $(\pi^i, C^i) := TSAB(\phi)$ and $C^* = \min\{C^*, C^i\}$.

Find $1 \leq k \leq maxE$ such, that $C^k = C^*$. Set $\pi^* := \pi^k$.

repeat

Find $1 \leq l \leq maxE$ so that $D(\pi^k, \pi^l) = \max\{D(\pi^k, \pi^i) : 1 \leq i \leq maxE\}$.

Set $\phi := NIS(\pi^k, \pi^l, C^*)$ and $(\pi^l, C^l) := TSAB(\phi)$.

If $C^l < C^k$ then set $(\pi^*, C^*) := (\pi^l, C^l)$ and $k := l$.

until $\max\{D(\pi^k, \pi^i) : 1 \leq i \leq maxE\} < maxD$.

Figure 2. Algorithm i-TSAB.

4.2. Proper algorithm

Algorithm i-TSAB works with the set of dispersed elite solutions $ES = \{\pi^i : i = 1, \ldots, maxE\}$, where $maxE$ is a tuning parameter. Pairs of solutions from ES transformed by *NIS* provide promising initial solutions for further local explorations of BV, performed in our case by TSAB. To make denotations brief, we use symbol $(\pi^B, C^B) := TSAB(\phi)$, if π^B has been returned by TSAB, started from ϕ and $C^B = C_{\max}(\pi^B)$. The run of i-TSAB consists fundamentally of two phases: *initiation* and the *proper work* phase, see Fig. 2.

In the initiation phase the algorithm builds up, step-by-step, the primal set ES. At the beginning, the set contains only single elite solution π^1, where $(\pi^1, C^1) := TSAB(\pi^0)$ and π^0 is an outer starting solution (provided by INSA; Nowicki and Smutnicki, 1996). Let π^{i-1}, $2 \leq i < maxE$, be the last solution introduced to ES. The successive elite solution π^i entered to ES is derived from the local exploration $(\pi^i, C^i) = TSAB(\phi)$, started with the new initial solution $\phi := NIS(\pi^{i-1}, \pi^0, C^*)$ located between π^{i-1} and π^0, $i = 2, \ldots, maxE$. The best known makespan C^* is updated by all search processes.

Just after ES has been filled with $maxE$ solutions, the algorithm passes to the proper work phase. In this phase, it repeatedly modifies the set ES, preserving its cardinality $maxE$ fixed. Because elite solutions from ES are derived from BV, one can use them to approximate the location of global minima. BV centre is approximated by the solution from ES with the minimal makespan, denoted next by π^k, whereas the remaining solutions roughly approximate borderlines of the promising search area. Our aim is to lessen, step-by-step, this search area, replacing solutions from ES with new ones, located towards the centre. To this order, we identify, at first, the most distant solution π^l from ES with respect to the centre π^k. Then, we generate the new initial solution $\phi = NIS(\pi^k, \pi^l, C^*)$ located between π^k and π^l. In the end, starting from this ϕ, we initiate local exploration $(\pi^B, C^B) = TSAB(\phi)$, which provides new elite solution π^B to replace unconditionally old π^l. This process is repeated until the distance from π^k to each of the remaining elite solutions becomes less than a specified, small threshold value $maxD$.

Although one can find some common elements in i-TSAB and *path relinking* (PR) philosophy, differences are evident. PR is based on the belief that while going towards trajectory between two local minima we can find a better local minimum (or even a global one), *immediately on* or

close to this trajectory. In our approach, the trajectory is an auxiliary tool that provides dispersed starting solutions for successive local explorations, very often conducted far from the trajectory. We found, in experiments, that a small number of restarts of LS algorithm with longer but precise exploration is much better than numerous restarts of LS, equipped with a quick but cursory exploration mechanism.

5. IMPLEMENTATION AND TUNING PARAMETERS OF i-TSAB

Algorithm i-TSAB has been coded in Delphi and run on a PC with Pentium III 900 MHz processor. Initial solutions have been found by algorithm INSA equipped with accelerator presented in Nowicki and Smutnicki (2001), which reduces running time from $O(o^2 \max_{k\in M} m_k)$ to $O(o^2)$. Made implementation applies Theorem 1 in NIS and TSAB (called at the lower level) for neighbourhoods $\mathcal{N}(\pi)$ and $\mathcal{N}^*(\pi)$, respectively.

5.1. Topological order

Below, we show how to find a new list F_α for the processing order $\alpha = \pi_{(w)}$, obtained by making a selected move $w = (x, y)$ from the proper neighbourhood in a quick way. Formally, it can be found by standard procedure, in the time $O(o)$, on the basis of $G(\alpha)$. The method proposed hereafter theoretically owns the same complexity as a standard procedure; however it is faster in practice. Let $j' = f_\pi(x)$ and $j'' = f_\pi(y)$ denote positions of swapped operations x, y on the list F_π; $j' < j''$. We split subsequence $Y = (x = F_\pi(j'), F_\pi(j'+1), \ldots, F_\pi(j'') = y)$ from the list F_π into two disjoined subsequences $W = (w_1, \ldots, w_r = y)$ and $Z = (x = z_1, \ldots, z_s)$, so that W contains all nodes from Y (excluding x) from which there exists a path to node y in graph $G(\pi)$; $r + s = j'' - j' + 1$. It can be verified that the list $L = (F_\pi(1), \ldots, F_\pi(j'-1), W, Z, F_\pi(j''+1), \ldots, F_\pi(o))$ is the topological order for graph $G(\alpha)$, which means that we can set $F_\alpha := L$. Operations y and x are located on positions $f_\alpha(y) := j' - 1 + r$ and $f_\alpha(x) := j' + r$, respectively. This procedure needs $O(j'' - j' + 1)$ of time. Value $j'' - j' + 1$ is significantly less than o, for o of order 1000 varies from 2 to 20.

5.2. Heads and tails

Having known the list F_α, theoretically we can find $r_\alpha(i), q_\alpha(i), i \in O, \alpha = \pi_{(w)}, w = (x, y)$, in the time $O(o)$, using obvious recursive formula

$$r_\alpha(F_\alpha(j)) = \max\{r_\alpha(i) : i \in B_\alpha(F_\alpha(j))\} + p_{F_\alpha(j)}, \quad j = 1, \ldots, o, \tag{8}$$

$$q_\alpha(F_\alpha(j)) = \max\{q_\alpha(i) : i \in A_\alpha(F_\alpha(j))\} + p_{F_\alpha(j)}, \quad j = o, o-1, \ldots, 1; \tag{9}$$

for completeness, we set $r_\alpha(0) = 0 = q_\alpha(0)$. In practice, they can be calculated in a faster way. First, observe that only several r_j's have to be recalculated, namely $r_\alpha(i) \neq r_\pi(i)$ only for nodes $i \in MR = \{j \in O : \exists \text{ path in } G(\pi) \text{ from } x \text{ to } j\}$; $x \in MR$. The set MR and new (modified) respective values $r_\alpha(i), i \in MR$ can be found by bow-tie algorithm from the paper by Ten Eikelder et al. (1999); however, we can do this in a more efficient way. All nodes from MR are in the subsequence $(F_\alpha(j' - 1 + r), \ldots, F_\alpha(o))$ of the list F_α. Thus, it is enough to apply formula (8) successively for $j = j' - 1 + r, j' + r, \ldots, o$, putting $r_\alpha(F_\alpha(k)) = r_\pi(F_\alpha(k))$, $1 \leq k \leq j' - 2 + r$. By symmetry, we calculate new values of $q_\alpha(i), i \in O$, using formula (9), successively for $j = j' + r, j' + r - 1, \ldots, 1$,

putting $q_\alpha(F_\alpha(k)) = q_\pi(F_\alpha(k))$, $j' + r + 1 \leq k \leq o$. This procedure theoretically needs a time $O(o)$; similarly the bow-tie algorithm, however, is several times faster in practice.

5.3. ULC detector

In order to detect and prevent situations when the same elite solutions are periodically introduced to ES (this phenomenon we call *upper level cycle*, ULC), a special ULC detector has been designed and appended to *i*-TSAB. ULCs have been observed sporadically in our computer tests, chiefly for smaller instances and only if the distance $\max_{1 \leq i \leq maxE} D(\pi^k, \pi^i)$ becomes sufficiently small, less than 50–80. The detector breaks an identified ULC by introducing to *ES* the processing order π^k instead of π^l (see Section 4 for the meaning of indices k and l).

5.4. Tuning parameters

Algorithm *i*-TSAB has two groups of parameters, which have to be chosen experimentally: (1) parameters introduced in Section 4: *max E*, *max V*, *max D*; and (2) parameters associated with TSAB acting at the lower level: *max iter* (maximal number of iterations without improving the makespan), *max t* (tabu list length), *max l* (maximal number of promising solutions stored), *max c*, *max* δ (parameters of the lower level cycle detector). All these parameters strongly interact, hence tuning of the algorithm may cause a certain problem. To make the tuning process simple we set parameters from the group (2) in accordance to the recommendation from the source paper (Nowicki and Smutnicki, 1996), namely $max\ t = 8$, $max\ l = 5$, $max\ c = 2$, $max\ \delta = 100$. Parameter *max iter* has been set, as the result of some sophisticated tests on Taillard's instances, as follows: *max iter* = 40,000/7,000, excluding small cases (with $o \leq 225$) for which we set *max iter* = 20,000/4,000; symbol a/b means *max iter* $= a$ if an improvement has occurred, and *max iter* $= b$ if Back Jump Tracking has been performed. After some other tests, we decided to set $max\ E = 8$ (as the best, from the accuracy and running time points of view), $max\ V = 0.5$ (since we do not observe dominance of any peculiar value from the interval [0, 1]), and $max\ D = 5$ (as the arbitrary stop criteria).

6. COMPARISON OF *i*-TSAB WITH OTHER ALGORITHMS

Algorithm *i*-TSAB has been tested on the following benchmarks: FT6, 10, 20 (Fisher and Thompson, 1963), LA01-40 (Lawrence, 1984), ABZ5-9 (Adams, Balas, and Zawack, 1988), ORB01-10 (Applegate and Cook, 1991), YN1-4 (Yamada and Nakano, 1992), SWV01-20 (Storer, Wu, and Vaccari, 1992), TA01-80 (Taillard, 1993) and DMU01-80 (Demirkol, Mehta, and Uzsloy, 1998). For the sake of limited length of the paper, we show here results for three hard benchmarks YN1-4, SWV01-20, TA01-80. Full results from our studies can be found in the report; Nowicki and Smutnicki (2002). Here, we only mention that four oldest benchmarks have been solved optimally (excluding ABZ8, 9), either by B&B or by approximate methods; modern approximate approaches (including TSAB) solve them with accuracy below 0.1%.

Among a rich number of approximate algorithms for the job shop problem, only a few (besides TSAB) own high accuracy, confirmed by representative tests on the broad benchmark set. These are two variants of shifted bottleneck approach, namely SB-RGSL10 (called here BAL) from Balas and Vazacopolous (1998) and TSSB (called here PEZ) from Pezzella and Merelli (2000). These algorithms (together with TSAB) dominate the remaining ones, thus only they will be used for

evaluations. To complete comparison, we also refer to the best solutions extracted from the report of Balas and Vazacopolous (1995) (called here BAL*), i.e. solutions from SB-GLS1, SB-GLS2 and double series of procedures SB-RGSLk for $k = 1, 5, 10, 15, 18$. In order to avoid discussion about computers' speed used in tests, we enclosed, for each algorithm, the original name of a machine, on which it has been tested, as well as the original running time. It is difficult to get the real computer-independent CPU time; Dongarra test (Balas and Vazacopoulos, 1998) refers to floating point operations and is therefore not suitable for algorithms PEZ and i-TSAB, which use exclusively integer operations.

For comparison to other methods, we have used two measures, namely the *accuracy* being opposed to the *running time*. Accuracy of an approximate algorithm A, which generates the solution π^A for some instance, has been evaluated by the relative percentage error RE of makespan $C_{\max}(\pi^A)$ to the lower bound LB of the optimal makespan and the relative improvement RI of $C_{\max}(\pi^A)$ to the fixed reference makespan C^{Ref} (more precisely to a certain good upper bound, commonly available in literature)

$$RE = 100\% \cdot \frac{C_{\max}(\pi^A) - LB}{LB}, \quad RI = 100\% \cdot \frac{C^{\text{Ref}} - C_{\max}(\pi^A)}{C^{\text{Ref}}}. \tag{10}$$

The results of running times will be presented in a compact form. We have found that the running time of i-TSAB (on the long-time horizon) almost linearly depends on the total number of iterations performed—the latter value is traced in our algorithm by counter *Tot Iter*. (Single iteration means the exploration of single neighbourhood in TSAB or NIS.) This fact allows us to calculate CPU time immediately from *Tot Iter* by simple proportion, assuming known CPU time for a fixed number of iterations; we use CPU time for 1 million iterations as the reference value. The average values of these measures, calculated over the selected group of instances, will be denoted by ARE, ARI and ACPU, respectively. Besides ACPU, we also refer to ACPU-B, which means the average CPU time to the best solution found.

Let us go to the detailed discussion of computational results. TA class contains 80 instances ($225 \le o \le 2{,}000$) organized into eight groups TA01-10, ..., TA71-80 with sizes $n \times m : 15 \times 15$, 20×15, 20×20, 30×15, 30×20, 50×15, 50×20 and 100×20, respectively. Instances TA51-80 are commonly considered easy. For all of them (excluding TA62, 67) the optimal solution has been found already in the initial phase of algorithm i-TSAB, as a rule, after a single call of TSAB. Therefore, we focus our attention on instances TA01-50, for which only 16 instances have known optimal solutions, see http://www.eivd.ch/ina/collaborateurs/etd/default.htm. Groups TA21–30, 41–50 still remain the hardest—not a single optimal solution has been found.

The behaviour of i-TSAB on instances TA01–50 has been shown in Table 1. Values of ARE, ARI and ACPU-B were scanned after a given number of iterations *Tot Iter*. Value of ACPU-B for fixed *Tot Iter*, represents the average CPU time to the best solution found, under restriction that i-TSAB stops after *Tot Iter* iterations. Algorithm i-TSAB provides solutions of quality better than algorithms PER, BAL and BAL* in a short time. For example, for TA21–30, algorithm i-TSAB offers ARE = 5.68% found in ACPU-B equals 328 seconds. For comparison, algorithm i-TSAB achieves ARE = 6.57% (that of BAL) before 0.5 million of iterations, which corresponds to ACPU-B less than 6 seconds on our PC (BAL needs, to this aim, approximately 4,380 seconds). Even taking into account a big difference in computers speed, i-TSAB works faster. Value ARE = 6.10%, characteristic for BAL*, is reached by algorithm i-TSAB within 2–5 million of iterations (25 and 47 seconds of ACPU-B). The same note refers to algorithm PEZ with ARE = 6.52%. Similar observation can also be made for the next hardest group TA41-50. Generally, i-TSAB on

Table 1. Test for TA, SWY and YN instances

	i-TSAB scanned at *Tot Iter* (in millions)									
Instances	0.5	1	2	5	10	20	50	PEZ	BAL	BAL*
	ARE [%] (to LB from http://www.eivd.ch/ina/collaborateurs/etd/default.htm; Jain and Meeran, 1999, and Brinkkötter and Brucker, 2001)									
TA01–10	0.64	0.48	0.27	0.11				0.45	0.25	0.16
TA11–20	4.04	3.47	3.11	2.98	2.91	2.81		3.47	3.34	2.81
TA21–30	6.55	6.20	6.12	6.01	5.88	5.69	5.68	6.52	6.57	6.10
TA31–40	1.73	1.63	1.48	1.18	1.03	0.85	0.78	1.92	1.13	0.80
TA41–50	6.81	6.25	5.98	5.69	5.33	4.97	4.70	6.04	5.71	5.20
TA01–50	3.95	3.61	3.39	3.19	3.05	2.89	2.82	3.68	3.40	3.01
SWV01–05	2.99	2.60	2.13	1.88	1.56	1.21	1.01		2.02	1.26
SWV06–10	10.03	9.41	8.90	8.71	8.42	8.07	7.49		9.64	8.06
SWV11–15	1.59	1.35	1.21	1.05	0.84	0.57	0.51		2.12	1.41
YN01–04	6.36	6.36	5.98	5.76	5.41	5.27	5.18		5.96	5.59
	ARI [%] (to reference makespans T, SWV and YN from Balas and Vazacopoulos, 1995)									
TA01–10	−0.04	0.11	0.33	0.49				0.15	0.34	0.44
TA11–20	−0.17	0.38	0.72	0.84	0.91	1.01		0.37	0.50	1.01
TA21–30	0.12	0.44	0.52	0.62	0.48	0.92	0.93	0.15	0.10	0.54
TA31–40	0.68	0.77	0.93	1.22	1.36	1.53	1.60	0.50	1.11	1.59
TA41–50	−0.46	0.07	0.33	0.60	0.93	1.27	1.53	0.26	0.56	1.05
SWV01–05	7.49	7.84	8.26	8.49	8.77	9.09	9.27		8.37	9.04
SWV06–10	9.15	9.67	10.09	10.25	10.49	10.77	11.24		9.48	10.78
SWV11–15	7.53	7.75	7.88	8.03	8.22	8.47	8.52		7.06	7.70
YN01–04	5.53	5.53	5.86	6.05	6.37	6.49	6.57		5.88	6.21
	ACPU-B [sec]									
TA01–10	4	7	17	26				2 175	1 182	
TA11–20	7	12	24	41	52	108		2 526	3 383	
TA21–30	6	15	25	47	140	278	328	34 910	4 377	
TA31–40	6	9	24	66	129	275	341	14 133	5 069	
TA41–50	11	27	38	99	260	536	975	11 512	10 726	
SWV01–05	5	9	26	62	131	269	462		1 290	
SWV06–10	5	13	22	33	67	273	514		2 917	
SWV11–15	7	16	27	50	143	228	360		9 173	
YN01–04	7	7	20	54	154	216	510		5 938	

The time needed by *i*-TSAB to perform 1 milion of iterations

Instances:	TA					SWV			YN
Group:	01–10	11–20	21–30	31–40	41–50	01–05	06–10	11–15	01–04
ACPU [s]	15.8	19.5	25.3	24.5	33.4	13.2	18.7	24.2	25.0

Note. PEZ—algorithm from Pezzella and Merelli (2000), BAL—algorithm SB-RGSL10 from Balas and Vazacopoulos (1998), BAL*—found on the basis of the best solution among those provided in Balas and Vazacopoulos (1995), ACPU-B—*i*-TSAB on Pentium 900 MHz, PEZ on Pentium 133 MHz, BAL on Sun Sparc 330.

Table 2. New upper bounds for TA11–50 instances

	$k = 10$			$k = 20$			$k = 30$			$k = 40$		
TA	LB	UB	UB*	LB	UB	UB*	LB	UB	UB*	LB	UB	UB*
k + 1	1323	1364	1361	1539	1645	1644		1764°		1859	2018	
k + 2	1351	1367		1511	1601	1600	1774	1796		1867	1961	1956
k + 3	1282	1342		1472	1558	1557	1778	1796	1793	1809	1879	1859
k + 4		1345°		1602	1651	1647	1828	1832	1829	1927	1989	1984
k + 5	1304	1340		1504	1597	1595		2007°		1997	2005	2000
k + 6	1302	1360		1539	1647	1645		1819°		1940	2022	2021
k + 7	1462	1464	1462°	1616	1687	1680	1771	1784	1778	1789	1913	1903
k + 8	1369	1396		1591	1615	1614		1673°		1912	1956	1952
k + 9	1297	1341	1335	1514	1625			1795°		1915	1968	
k + 10	1318	1353	1351	1473	1585	1584	1631	1686	1674	1807	1937	1928

Note. UB—old upper bound; UB*—new upper bound.
°Optimal makespan.

instances TA01–50 provides ARE = 2.82%, whereas PEZ, BAL and BAL* provides only 3.68, 3.40 and 3.01%, respectively.

SWV class contains 20 instances ($200 \leq o \leq 500$) organized into four groups SWV01–05, ..., SWV16–20 with sizes 20×10, 20×15, 50×10, 50×10, respectively. SWV16–20 are easy (optimal solutions have been found in single call of TSAB) and we will therefore analyse only the first three groups; optimal solutions are known for six instances. YN class contains four instances YN01–04 ($o = 400$) with size 20×20; no optimal solutions have been known. SWV and YN have not been tested by algorithm PEZ; we have compared i-TSAB only with BAL and BAL*. Results are shown in Table 1. Similarly as for TA instances, algorithm i-TSAB provides solutions of a quality better than algorithms BAL, BAL* in a short time.

Up till now, all benchmarks have been attacked, by means of very time-consuming runs, for *all* new generations of algorithms. Nevertheless, i-TSAB during various tests and standard runs (with tuning parameters recommended in Section 5) found many new upper bounds. In the group TA11–50, among the unsolved 34 instances, we improved 25 upper bounds, see Table 2; for TA17, the result provides optimal solution. In groups SWV, YN, we found six new upper bounds, namely, SWV04–1474, SWV07–1600, SWV9–1661, SWV11–2983, YN01–885 and YN03–892; for SWV11 the obtained makespan is optimal. Among the unsolved 63 DMU instances, we improved 60 upper bounds, see Taillard (1994). To summarize, we have found 91 better upper bounds among 112 yet unsolved instances.

7. CONCLUSIONS

The proposed algorithm i-TSAB provides a powerful tool to solve the job shop problem with the makespan cirterion. It offers very good accuracy, in comparison to other best known approaches, obtainable in a short running time on a modern PC. These properties, confirmed through exhaustive tests on all known benchmarks, follow from the suitable use of properties of the solution space,

especially BV. The general idea of the algorithm can be applied to other scheduling problems, as an example, the flow shop and hybrid flow-shop problem.

APPENDIX A: PROOF OF PROPERTY 1

We begin completing the definition of a set of paths. Let us denote by $P_\tau(\bar{a} \wedge \bar{b})$ and by $P_\tau(\bar{a} \wedge b)$ the set of all paths 'not containing nodes a and b' and 'not containing node a but containing node b', respectively.

Taking into account the position of operation x on a machine μ_x, the following four cases occur: (i) $Bm_\pi(x) \neq 0$, $Am_\pi(y) \neq 0$; (ii) $Bm_\pi(x) \neq 0$, $Am_\pi(y) = 0$; (iii) $Bm_\pi(x) = 0$, $Am_\pi(y) \neq 0$; and (iv) $Bm_\pi(x) = 0$, $Am_\pi(y) = 0$. In the case (i) the graph $G(\sigma)$ has been obtained from $G(\pi)$ by removing three sequencing arcs $(Bm_\pi(x), x)$, (x, y), $(y, Am_\pi(y))$ and adding the next three other sequencing arcs $(Bm_\pi(x), y)$, (y, x), $(x, Am_\pi(y))$. In the case (ii), by removing two sequencing arcs $(Bm_\pi(x), x)$, (x, y) and adding $(Bm_\pi(x), y)$, (y, x); in the case (iii), by removing two sequencing arcs (x, y), $(y, Am(y))$ and adding (y, x), $(x, Am_\pi(y))$; and finally, in the case (iv), by removing one sequencing arc (x, y) and adding (y, x). In all four cases, we obtain $P_\sigma(\bar{x} \wedge \bar{y}) = P_\pi(\bar{x} \wedge \bar{y})$. Moreover, for each path from the set $P_\pi(\bar{x} \wedge y)$, there exists in the set $P_\sigma(\bar{x} \vee \bar{y})$ a path containing all its nodes, which implies the following inequality $D[P_\pi(\bar{x} \wedge y)] \leq D[P_\sigma(x \vee y)]$. Using the obvious relation between the introduced notions and employing the above dependencies, we obtain $C_{\max}(\sigma) = \max\{D[P_\sigma(x \vee y)], D[P_\sigma(\bar{x} \wedge \bar{y})]\} = \max\{D[P_\sigma(x \vee y)], D[P_\pi(\bar{x} \wedge \bar{y})]\} = \max\{D[P_\sigma(x \vee y)], D[P_\pi(\bar{x} \wedge \bar{y})], D[P_\pi(\bar{x} \wedge y)]\} = \max\{D[P_\sigma(x \vee y)], D[P_\pi(\bar{x})]\}$ which provides (1) and completes the proof.

APPENDIX B: PROOF OF PROPERTY 2

From the definition of F_π it follows that for any path $w = (w_1, w_2, \ldots, w_z)$, $w_i \in O$, $i = 1, \ldots, z \geq 1$, in graph $G(\pi)$, we have $f_\pi(w_1) < f_\pi(w_2) < \cdots < f_\pi(w_z)$. Therefore, value $D[P_\pi(\bar{x})]$ can be found through the analysis of the following five, not necessarily disjoined, classes of paths $w \in P_\pi(\bar{x})$: (RQ) there exists a node w_J, $1 \leq J \leq z-1$ so, that $f_\pi(w_J) < f_\pi(x) < f_\pi(w_{J+1})$, (R′) $f_\pi(w_z) < f_\pi(x)$ and $w_z \in LJ$, (R″) $f_\pi(w_z) < f_\pi(x)$ and $w_z \in B_\pi(x)$, (Q′) $f_\pi(x) < f_\pi(w_1)$ and $w_1 \in FJ$, (Q″) $f_\pi(x) < f_\pi(w_1)$ and $w_1 \in A_\pi(x)$. Any path $w \in P_\pi(\bar{x})$ either belongs to one of the enumerated classes or can be considered as a part of some path from these classes. Denoting lengths of maximal paths in classes (RQ), (R′), (R″), (Q′), (Q″) by RQ, R', $R''Q'$, Q'', respectively, we get directly formula (3). Thus, there remains the proof of (4). From equation (2), we have $LB^T \geq R''$, $LB^T \geq q_\pi(Aj(x))$ and $LB^T \geq p_y + \max\{q_\pi(Aj(y)), q_\pi(Am_\pi(y))\} = q_\pi(y)$. From two recent inequalities, taking account of $y = Am_\pi(x)$, we obtain $LB^T \geq Q''$, which justifies (4) and completes the proof.

APPENDIX C: PROOF OF THEOREM 1

Values LB^T, R' and Q' can be found in the time $O(1)$, $O(n)$ and $O(n)$, respectively. Formally, since $|E(\pi)| = \sum_{k=1}^{m}(m_k - 1) = o - m$ and $|R| = \sum_{j=1}^{n}(o_j - 1) = o - n$, value RQ can be found in the time $O(o)$. However, in the set of arcs $\bigcup_{i=1}^{m_k-1}\{(\pi_k(i), \pi_k(i+1))\}$ associated with machine $k \in M$, there exists at most one arc (a, b) such that $f_\pi(a) < f_\pi(x) < f_\pi(b)$. One can find this arc (or an answer that it does not exist) in the time $O(\log m_k)$ by means of binary search in the increasing sequence $f_\pi(\pi_k(1)), \ldots, f_\pi(\pi_k(m_k))$. A similar case appears for a set of arcs $\bigcup_{i=1}^{o_j-1}\{(l_{j-1} + i, l_{j-1} + i + 1)\}$

associated with job j, $j \in J$. There, necessary arc (a, b) can be found in the time $O(\log o_j)$, by means of binary search in the increasing sequence $f_\pi(l_{j-1}+1), \ldots, f_\pi(l_{j-1}+o_j)$. Thus, value RQ can be found in the time $O(\max\{\sum_{j=1}^{n} \log o_j, \sum_{k=1}^{m} \log m_k\})$, which implies that $C_{\max}(\sigma)$ can be found in the same time and completes the proof.

ACKNOWLEDGMENT

The research was supported by Grant T11A 01624 of the SCSR, 2003–2006.

REFERENCES

Adams, J., E. Balas, and D. Zawack, "The shifting bottleneck procedure for job shop scheduling," *Management Science*, **34**, 391–401 (1988).

Applegate, D. and W. Cook, "A computational study of the job-shop scheduling instance," *ORSA Journal on Computing*, **3**, 149–156 (1991).

Balas, E. and A. Vazacopoulos, "Guided local search with shifting bottleneck for job-shop scheduling," Technical Report MSRR-609, GSIA, Carnegie Mellon University, Pittsburgh, 1995.

Balas, E. and A. Vazacopoulos, "Guided local search with shifting bottleneck for job-shop scheduling," *Management Science*, **44**, 262–275 (1998).

Blałżewicz, J., W. Domschke, and E. Pesch, "The job shop scheduling problem. Conventional and new solution techniques," *European Journal of Operational Research*, 1–33 (1996).

Brinkkötter, W. and P. Brucker, "Solving open benchmark probles for job shop problem," *Journal of Scheduling*, **4**, 53–64 (2001).

Demirkol, E., S. Mehta, and R. Uzsloy, "A computational study of shifting bottleneck procedure for shop scheduling problems," *Journal of Heuristic*, **3**(2), 111–137 (1997),

Demirkol, E., S. Mehta, and R. Uzsloy, Benchmarks for shop scheduling problems," *European Journal of Operational Research*, **109**, 137–141 (1998).

Dorndorf, U., E. Pesch and T. Phan-Huy, "Constraint propagation techniques for the disjunctive scheduling problem," *Artificial Intelligence*, **122**, 189–240 (2000).

Dorndorf, U., E. Pesch, and T. Phan-Huy, "Constraint propagation and problem decomposition: A pre-processing procedure for the job shop problem," *Annals of Operations Research*, **115**, 125–145 (2002).

Fisher, H. and G. L. Thompson, "Probabilistic learning combinations of local jobshop scheduling rules," in J. F. Muth and G. L., Thompson (eds.), *Industrial Scheduling*, Prentice Hall, Englewood Cliffs, New Jersey, 1963, pp. 225–251.

Glover, F. and M. Laguna, *Tabu Search*, Kluwer Academic Publishers. Massachusetts USA, 1997.

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

Home page of E. Taillard Internet, http://www.eivd.ch/ina/collaborateurs/etd/default.htm

Jain, A. S., "A multi-level hybrid framework for the deterministic job-shop scheduling problem," Ph.D. Thesis, University of Dudee, Dundee, Scotland, UK, 1998.

Jain, A. S. and S. Meeran, "Deterministic job-shop scheduling: Past, present and future," *European Journal of Operational Research*, **113**, 390–434 (1999).

Jain, A. S., B. Rangaswamy, and S. Meeran, "New and "stronger" job-shop neighbourhoods: A focus on the method of Nowicki and Smutnicki (1996)," *Journal of Heuristics*, **6**(4), 457–480 (2000).

Lawrence, S., "Resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques (Supplement)," Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania. 1984.

Nowicki, E. and C. Smutnicki, "A fast tabu search algorithm for the job-shop problem," *Management Science*, **42**(6), 797–813 (1996).

Nowicki, E. and C. Smutnicki, "Some new ideas in TS for job-shop scheduling," Technical Report 50/01. Institute of Engineering Cybernetics, Wroclaw University of Technology, 2001, in: C. Rego and B. Alidaee (eds.), *Metaheuristic Optimization via Memory and Evolution. Tabu Search and Scatter Search*, Kluwer Academic Publishers, 2004.

Nowicki, E. and C. Smutnicki, "Some new tools to solve the job-shop problem," Technical Report 60/02, Institute of Engineering Cybernetics, Wroclaw University of Technology, 2002. Internet http://www.zsd.ict.pwr.wroc.pl

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

Pezzella, F. and E. Merelli, "A tabu search method guided by shifting bottleneck for the job-shop scheduling problem," *European Journal of Operational Research*, **120**, 297–310 (2000).

Roy, B. and B. Sussman, "Les problemes d'ordonnancement avec contraintes disjonctives," Note DS 9 bis, SEMA, 1964, Paris (in French).

Storer, R. H., S. D. Wu, and R. Vaccari, "New search spaces for sequencing instances with application to job shop scheduling," *Management Science*, **38**, 1495–1509 (1992).

Schilham, R. M. F., "Commonalities in local search," Ph.D. Thesis, Eindhoven University of Technology, The Netherlands, 2000.

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

Taillard, E., "Parallel taboo search techniques for the job shop scheduling problem," *ORSA Journal on Computing*, **6**, 108–117 (1994).

Ten Eikelder, H. M. M., B. J. M. Aarts, M. G. A. Verhoeeven, and E. H. L. Aarts, "Sequential and paralleled local search algorithms for job shop scheduling," in S. Voss, S. Martello, I. H. Osman, and C. Roucairol (eds.), *Meta-Heuristic: advances and Trends in Local Search Paradigms for Optimization*, Kluwer Academic Publishers, Norwell, Massachusetts, pp. 1994, 359–371.

Vaessens, R., E. Aarts, and J. K. Lenstra, "Job shop scheduling by local search," *INFORMS Journal on Computing*, **8**, 303–317 (1996).

Van Laarhoven, P., E. Aarts, and J. K. Lenstra, "Job-shop scheduling by simulated annealing," *Operations Research*, **40**, 113–125 (1992).

Yamada, T. and R. Nakano, "A genetic algorithm applicable to large-scale job-shop instances," in R. Mnner and B. Manderick (eds.), *Parallel Instance Solving From Nature 2*, North-Holland, Amsterdam, 1992, pp. 281–290.