Effective Neighborhood Functions for the Flexible Job Shop Problem*

Monaldo Mastrolilli · Luca Maria Gambardella
IDSIA - Istituto Dalle Molle di Studi sull'Intelligenza Artificiale
C.so Elvezia 36, 6900 Lugano, Switzerland
{monaldo,luca}@idsia.ch, http://www.idsia.ch

January 31, 2000

Abstract

The Flexible Job Shop Problem is an extension of the classical job shop scheduling problem which allows an operation to be performed by one machine out of a set of machines. The problem is to assign each operation to a machine (routing problem) and to order the operations on the machines (sequencing problem), such that the maximal completion time (*makespan*) of all operations is minimized. To solve the Flexible Job Shop problem approximately, we use local search techniques and present two neighborhood functions ($Nopt1$,$Nopt2$). $Nopt2$ is proved to be optimum connected. $Nopt1$ does not distinguish between routing or sequencing an operation. In both cases, a neighbor of a solution is obtained by moving an operation which affects the makespan. Our main contribution is a reduction of the set of possible neighbors to a subset for which can be proved that it always contains the neighbor with the lowest makespan. An efficient approach to compute such a subset of feasible neighbors is presented. A tabu search procedure is proposed and an extensive computational study is provided. We show that our procedure outperforms previous approaches.

Keywords: flexible job shop, tabu search.

*Supported by Swiss National Science Foundation project 21-55778.98, "Resource Allocation and Scheduling in Flexible Manufacturing Systems".

1 Introduction

In the job shop scheduling problem (JSP), there is a set $\mathcal{J} = \{J_1, \ldots, J_n\}$ of n jobs that must be processed on a group $M = \{1, \ldots, m\}$ of m machines. Each job J_j consists of a sequence of n_j operations $O_{1j}, O_{2j}, \ldots, O_{n_j j}$, where O_{ij} must be processed without interruption on machine $m_{ij} \in \{1, \ldots, m\}$ during p_{ij} time units. The operations $O_{1j}, O_{2j}, \ldots, O_{n_j j}$ must be processed one after another in the given order and each machine can process at most one operation at a time.

In this paper we study a generalization of JSP called the flexible job shop problem (FJSP), which provides a closer approximation to a wide range of problems encountered in real manufacturing systems.

FJSP extends JSP by allowing an operation O_{ij} to be executed by one machine out of a set M_{ij} of given machines, where $M_{ij} \subseteq M$. The processing time for operation O_{ij} on machine $k \in M_{ij}$ is $p_{ijk} > 0$. The goal is to choose for each operation O_{ij} a machine $\mu(O_{ij}) \in M_{ij}$ and a starting time s_{ij} when it must be performed so that makespan is minimized.

FJSP is therefore made more complex than JSP by the need to determine a routing policy (i.e., the assignment of operations to machines) other than the traditional sequencing decisions. The FJSP is NP-hard since it is an extension of the job shop scheduling problem [17]. The NP-hardness of an optimization problem suggests that it is not always possible to find an optimal solution quickly. Therefore, instead of searching for an optimal solution with enormous computational effort, we may instead use a local search method or an approximation algorithm to generate approximate solutions that are close to the optimum with considerably less investment of computational resources.

Our approach to solve the FJSP approximately is based on a local search method [1]. Local search employs the idea that a given solution may be improved by making small changes. Consider the minimization problem $\min\{f(S)|S \in \Sigma\}$, where f is the objective function and Σ is the search space, i.e. the set of feasible solutions of the problem. A neighborhood function is a mapping $\mathcal{N} : \Sigma \to 2^{\Sigma}$, which defines for each solution $S \in \Sigma$ a subset $\mathcal{N}(S)$ of Σ, called a neighborhood. Each solution in $\mathcal{N}(S)$ is a neighbor of S. A local search algorithm starts off with an initial solution and then continually tries to find better solutions by searching neighborhoods.

In this paper, a neighbor of a given solution S is obtained by *moving* an operation. More precisely, first, an operation v is chosen and is deleted from its machine sequence; next, assign v to an eligible machine (not necessary different from the previous one); finally, *insert* v in the chosen machine sequence such that the resulting schedule is feasible. An optimal insertion of v is the one with the lowest makespan (see section 4.2). We reduce the set of possible neighbors to a subset that always contains an optimal insertion. In particular, for a selected operation v and a machine k, an efficient method is proposed for computing a restricted set of solutions F_{vk} that always contains the solution obtained by inserting v in k optimally.

This paper is organized as follows. In Sections 2 and 3 a solution graph

representation and an overview of previous research are provided. Section 4 defines and proves the above mentioned properties of F_{vk}. Next, neighborhood functions $Nopt1$ and $Nopt2$ are presented in Section 6. In Section 7 a tabu search procedure is proposed. In Section 8 we present an extensive computational study on 221 benchmark problems where our approach is compared with previous approaches. Some final remarks are given in Section 9.

2 The Solution Graph

Let $\mathcal{O}$ denote the set of all operations, i.e., $\mathcal{O} = \{O_{ij} | j = 1, ..., n \text{ and } 1 \leq i \leq n_j\}$. A solution (μ, s) of the problem defines for each operation $v \in \mathcal{O}$ a unique machine $\mu(v)$ on which v is processed without preemption and a starting time s_v.

We will represent a solution (μ, s) by means of a *solution graph*. Each node represents an operation. Two dummy nodes, 0 and $*$, are introduced, representing the start and the end of the planning period. Each node has a weight which is equal to the processing time $p_{v,\mu(v)}$ of the corresponding operation v, when v is processed on machine $\mu(v)$; furthermore $p_0 = p_* = 0$.

Precedence relations are incorporated in the solution graph by means of *precedence arcs*; for each couple (O_{ij}, O_{i+1j}), where $1 \leq i \leq n_j - 1$ and $j = 1, ..., n$, an arc (O_{ij}, O_{i+1j}) is introduced. For each operation $v \in \mathcal{O}$ we introduce *dummy arcs* $(0, v)$ and $(v, *)$. If it has been decided that an operation u is performed before operation v on a machine, a *machine arc* (u, v) is introduced.

We will denote a solution graph by $G(\mathcal{V}, \mathcal{A}')$, where $\mathcal{V} = \mathcal{O} \cup \{0, *\}$, and $\mathcal{A}'$ consists of all machine arcs, precedence arcs, and dummy arcs. Let the starting time of the dummy operation 0 be fixed at 0. Each arc $(u, v) \in \mathcal{A}'$ can then be considered as representing a constraint of the form $s_v \geq s_u + p_{u,\mu(u)}$.

Many of the machine arcs and dummy arcs in a solution graph are redundant in the sense that they are implied by other machine and dummy arcs. It is easy to compute a reduced solution graph $G(\mathcal{V}, \mathcal{A})$, so that all information represented in the solution graph $G(\mathcal{V}, \mathcal{A}')$ is also present in the reduced solution graph $G(\mathcal{V}, \mathcal{A})$ and the number of arcs is bounded by $O(N)$, where N is the total number of operations. Precisely, graph $G(\mathcal{V}, \mathcal{A})$ is the transitive reduction of $G(\mathcal{V}, \mathcal{A}')$.

Here, the length of a path $(w_1, w_2, ..., w_q)$ is defined as the sum of the processing times of the operations w_2 up to and including w_{q-1}. Denoting the value of some longest path from node i to node j by $\ell(i, j)$, the starting time s_w of operation w in a left justified schedule is equal to $\ell(0, w)$ in the corresponding solution graph. The makespan of a solution is equal to the length of some longest path from 0 to $*$, i.e., $\ell(0, *)$. This path is often referred to as the critical path, and the arcs and nodes on this path are called critical. Sometimes there are several longest paths. A solution is infeasible if and only if the corresponding solution graph contains a cycle.

In the remainder $p_{w,\mu(w)}$ is written as p_w, when it is clear which machine processes operation w. In addition, the tail time t_w is defined. It corresponds to

the length of some longest path from operation w to node $*$, i.e., $t_w = \ell(w, *)$. It follows that an operations w is critical if and only if $s_w + p_w + t_w = C_{\max}$, where $C_{\max}$ denotes the makespan of the corresponding solution.

Bellman's labeling algorithm [10] can be used in order to compute the makespan and all s_w and t_w values in time $O(N)$, for any given solution.

3 Literature review

A review of some approaches to the FJSP is reported here. An early approach is described by Brucker and Schlie [8]. They give a polynomial time algorithm for minimizing the makespan for the case of two machines.

Heuristic approaches based on local search techniques are usually classified according to how routing and scheduling problems are solved during the search. A distinction between simultaneous and hierarchical approaches is usually made. The hierarchical approach [2, 5, 6, 15, 26] is based on the observation that when a routing is chosen, FJSP turns into the classical job shop problem. Given the machine assignment, the problem is to find the sequence of operations minimizing a given performance function. Brandimarte [6] first determines an assignment and then focuses on the job shop scheduling problem. A reassignment of one of the critical operations is done at predefined intervals and then again the focus is on the resulting job shop scheduling problem. Other authors suggest more integrated approaches [4, 7, 11, 21, 19, 27]. In [7, 4, 19, 21] the assignment of operations to resources and the sequencing of operations on the resources are two different types of transitions. In [11, 27] there is no distinction between reassigning or resequencing an operation. Vaessens [27] obtains a neighbor by deleting an operation v from the machine ordering, by identifying a set of feasible insertions containing the optimal one and by inserting v in the best possible way. Vaessens' algorithm spends a considerable amount of computing time for defining such a set of feasible insertions. Brucker and Neyer [7] also propose the best insertion of an operation in their neighborhood function, but the algorithm they suggest is too time consuming. In order to reduce the computational effort, they propose a faster algorithm that guarantees only the feasibility of an insertion.

Recently, Jansen et al. [20] developed a linear time approximation scheme for the flexible job shop problem with fixed number of machines and number of operations per job.

4 Moving an operation

In the proposed approach, a neighbor of a solution S is obtained by *moving* and *inserting* an operation in an allowed machine sequence. Suppose that a feasible solution (μ, s) is given and let G be the corresponding solution graph. A k-insertion of operation $v \in \mathcal{O}$ in machine $k \in M_v$ is performed in 2 steps.

1. Delete v from its current machine sequence by removing all its machine arcs. Set the weight of node v equal to 0.

2. Assign v to machine k and choose the position of v in the processing order of k, by adding its machine arcs and setting the weight of node v equal to p_{vk}.

Let G^- be the graph obtained from G at the end of step 1 (for the remainder, the superscript "$-$" refers to the situation after the execution of step 1, and v denotes the operation to be moved). A k-insertion of v is *feasible* if it does not create a cycle in the resulting graph. Clearly, G^- is acyclic since G is already acyclic. A k-insertion is called an optimal k-insertion, if

- it is feasible;
- the makespan of the corresponding schedule is minimal; i.e. is smaller than or equal to the makespan of all other schedules resulting from feasible k-insertions.

An insertion of v is called optimal if it leads to a schedule with minimal makespan within the set of all schedules resulting from optimal k-insertions of v, $k \in M_v$.

In the following we describe how to compute, for any given operation v and machine $k \in M_v$, a subset of neighbors F_{vk} that always contains the solution obtained by performing an optimal k-insertion of v. Obviously, the set of neighbors $F_v = \cup_{k \in M_v} F_{vk}$ contains the solution obtained by performing an optimal insertion of v.

Let s_x^- and t_x^- be the starting and the tail time of a generic operation $x \in \mathcal{O}$ in G^-. Notice that in G^-, for every operation x, we have $s_x^- \leq s_x$ and $t_x^- \leq t_x$, since the removal of an operation from a machine cannot increase the starting and tail times. Furthermore, since in G^- the machine arcs of operation v are deleted, it follows that $s_v^- = s_{PJ[v]}^- + p_{PJ[v]}$ and $t_v^- = p_{SJ[v]} + t_{SJ[v]}^-$, where $PJ[v]$ ($SJ[v]$) denotes the operation of the same job of v that directly precedes (follows) v. If v is the first (last) job operation, we set $PJ[v] = 0$ ($SJ[v] = *$). It should be clear that $s_{PJ[v]}^- = s_{PJ[v]}$ and $t_{SJ[v]}^- = t_{SJ[v]}$, since the removal of v cannot change the starting time of its previous job operations, and the tail times of its following job operations.

Let Q_k be the set of operations processed by k in G^- and sorted by increasing starting time (note that $v \notin Q_k$). Let R_k and L_k denote two subsequences of Q_k defined as follows,

$$R_k = (x \in Q_k | s_x + p_x > s_v^-) \quad (1)$$

$$L_k = (x \in Q_k | p_x + t_x > t_v^-) \quad (2)$$

The set F_{vk} will be defined as follows.

Definition 1 *Let F_{vk} be the set of solutions obtained by inserting v after all the operations of $L_k \backslash R_k$ and before all the operations of $R_k \backslash L_k$.*

Note that the computation of R_k and L_k only requires starting and tail times of operations in G, therefore F_{vk} is calculated by using only information from the current graph solution.

In subsection 4.1 it is shown that F_{vk} is a set of feasible neighbors, while in subsection 4.2 it is proved that other k-insertions than the ones used to define F_{vk} cannot deliver a solution with a better makespan.

4.1 Feasible insertion

Let x be any operation of the set Q_k of operations processed by k. If a path from x to v exists in G^-, then x must be scheduled before v in order to obtain a feasible solution, otherwise a cycle is produced. Similarly, if a path from v to x exists in G^-, then x must be scheduled after v. Finally, if no path between x and v exists in G^-, then any insertion of v just after or before x delivers a feasible solution.

Properties of R_k By the definition of R_k there is no path from any operation $x \in R_k$ to v in G^-. If x is an operation of $Q_k \backslash R_k$ then $s_x + p_x \leq s_v^-$. Since $s_v \geq s_v^-$ and $p_v > 0$, it follows that $s_x < s_v + p_v$. Hence no path from v to any operation of $Q_k \backslash R_k$ in G (and hence in G^-) exists. The above situation is illustrated in Figure 1.

Properties of L_k Again, by the definition of L_k there is no path from v to any operation $x \in L_k$ in G^-. If x is an operation of $Q_k \backslash L_k$ then $p_x + t_x \leq t_v^-$. Since $t_v \geq t_v^-$ and $p_v > 0$, it follows that $t_x < p_v + t_v$, thus there is no path from any operation of $Q_k \backslash L_k$ to v in G (and hence in G^-). This situation is depicted in Figure 2.

Relationship between R_k and L_k In the following it is shown that the set F_{vk} of solutions obtained by inserting v after every operation of $L_k \backslash R_k$ and before every operation of $R_k \backslash L_k$, is a set of feasible solutions.

By the definition of R_k and L_k, any operation of $L_k \backslash R_k$ is completed before any operation of $R_k \backslash L_k$. We distinguish between two cases, i.e., $L_k \cap R_k \neq \emptyset$ and $L_k \cap R_k = \emptyset$.

1. Case $L_k \cap R_k \neq \emptyset$. For each operation x of $L_k \cap R_k$, there is no path from v to x and from x to v. Therefore, every k-insertion of v in $L_k \cap R_k$ (after every operation of $L_k \backslash R_k$ and before every operation of $R_k \backslash L_k$) delivers a feasible solution. This case is represented in Figure 3.

2. Case $L_k \cap R_k = \emptyset$. For each operation x of $Q_k \backslash (L_k \cup R_k)$, there is no path from v to x and from x to v in G^-. Hence, any k-insertion of v after every operation of L_k and before every operation of R_k delivers a feasible solution. This case is represented in Figure 4.

4.2 Optimal insertion

In this subsection we prove that other k-insertions than the ones used to define F_{vk} cannot deliver a solution with a better makespan.

Let (u, w) denote a k-insertion of v obtained by scheduling v just after $u \in Q_k$, and before $w \in Q_k$, where w is the operation scheduled just after u in Q_k. The corresponding solution graph $G^{(u,w)}$ is obtained from G^- by adding machine arcs (u, v) and (v, w) and setting the weight of node v equal to p_{vk}. For any $x \in \mathcal{V}$, let the starting and tail times in $G^{(u,w)}$ be denoted as $s_x^{(u,w)}$ and $t_x^{(u,w)}$, respectively. The length of a longest path in $G^{(u,w)}$ containing v is equal to $s_v^{(u,w)} + p_{vk} + t_v^{(u,w)}$.

Since $G^{(u,w)}$ is derived from G^- by only adding arcs incident to v, the makespan $C_{\max}^{(u,w)}$ of the solution graph $G^{(u,w)}$ is the maximum between the length $\ell^-(0, *)$ of a longest path in the graph G^-, and $s_v^{(u,w)} + p_{vk} + t_v^{(u,w)}$, i.e.,

$$C_{\max}^{(u,w)} = \max\left\{\ell^-(0, *), s_v^{(u,w)} + p_{vk} + t_v^{(u,w)}\right\}.$$

A k-insertion (u, w) of v is an optimal k-insertion if $C_{\max}^{(u,w)}$ is minimal. However, since the constant $\ell^-(0, *)$ occurs for each insertion, it suffices that $s_v^{(u,w)} + p_{vk} + t_v^{(u,w)}$ is minimal. Since $s_v^{(u,w)} = \max\{s_u^- + p_u; s_v^-\}$ and $t_v^{(u,w)} = \max\{p_w + t_w^-; t_v^-\}$, the value of $s_v^{(u,w)} + p_{vk} + t_v^{(u,w)}$ is equal to $s_v^- + p_{vk} + t_v^- + \Delta(u, w)$, where

$$\Delta(u, w) = \max\left\{s_u^- + p_u - s_v^-; 0\right\} + \max\left\{p_w + t_w^- - t_v^-; 0\right\}. \tag{3}$$

Thus, a k-insertion (u, w) of v for which $\Delta(u, w)$ is minimal is an optimal k-insertion.

Since F_{vk} is defined as the set of solutions obtained by inserting v after all the operations of $L_k \backslash R_k$ and before all the operations of $R_k \backslash L_k$, the following theorem shows that other k-insertions than the ones used to define F_{vk} cannot deliver a solution with a better makespan.

Theorem 1 *There is an optimal k-insertion of v obtained by inserting v after every operation of $L_k \backslash R_k$ and before every operation of $R_k \backslash L_k$. Furthermore, if $L_k \cap R_k = \emptyset$ such insertions are all optimal k-insertion.*

Proof. Section 4.1 shows that any insertion of v after every operation of $L_k \backslash R_k$ and before every operation of $R_k \backslash L_k$ produces a feasible set F_{vk} of solutions. F_{vk} is shown to contain an optimal insertion in two steps.

(Case 1) The insertion of v as a direct successor of the last operation of $L_k \backslash R_k$ is not worse than all those obtained by inserting v before any operation of $L_k \backslash R_k$;

(Case 2) The insertion of v as a direct predecessor of the first operation of $R_k \backslash L_k$ is not worse than all those obtained by inserting v after any operation of $R_k \backslash L_k$.

1. For each operation u of $L_k \backslash R_k$ we have $\max\{s_u + p_u - s_v^-; 0\} = 0$. Since $s_u \geq s_u^-$, it follows that $\max\{s_u^- + p_u - s_v^-; 0\} = 0$. Therefore the value of expression (3) when v is inserted just after u is equal to $\max\{p_w + t_w^- - t_v^-; 0\}$, where w is the operation that follows u in Q_k. Since for each operation x that follows w in Q_k we know that $\max\{p_w + t_w^- - t_v^-; 0\} \geq \max\{p_x + t_x^- - t_v^-; 0\}$, the set of solutions obtained by inserting v before the last operation of $L_k \backslash R_k$ cannot deliver a better solution than the one obtained by inserting v as a direct successor of the last operation of $L_k \backslash R_k$.

2. (Case 2 is symmetric to Case 1).

Now consider the case when $L_k \cap R_k = \emptyset$. Then $L_k \backslash R_k = L_k$ and $R_k \backslash L_k = R_k$. For each operation x of $Q_k \backslash (L_k \cup R_k)$, $s_x + p_x \leq s_v^-$ and $p_x + t_x \leq t_v^-$. Again, since $s_x \geq s_x^-$ and $t_x \geq t_x^-$, it follows that $\max\{s_x^- + p_x - s_v^-; 0\} = \max\{p_x + t_x^- - t_v^-; 0\} = 0$. For each operation x of L_k, $\max\{s_x^- + p_x - s_v^-; 0\} = 0$, and for each operation x of R_k, $\max\{p_x + t_x^- - t_v^-; 0\} = 0$. Therefore all the insertions of v after all the operations of L_k and before all the operations of R_k are optimal. ■

The computation of a reduced set of insertions that always contains an optimal k-insertion of v is a task that takes $O(\log N)$ time. Indeed, $|Q_k| < N$ and the time required to determine L_k and R_k is $O(\log |Q_k|)$ by using binary search.

5 Moves Evaluation

In order to assess the effectiveness of a given k-insertion of v we use the value of the new longest path which contains operation v. The length of this path is a valid lower bound on the value of the new solution. The exact value of the longest path which contains operation v can be calculated in $O(N)$ time; however, doing this for every candidate v and every machine $k \in M_v$ at every step becomes expensive. In the following we introduce an attempt to avoid the calculation of longest paths. Indeed, we compute upper bounds instead of the exact values. Over 52 problems with different features, we experimentally found that the proposed upper bound is very close to the exact length. Indeed, it was on average only 0.001% bigger than the exact value.

The upper bound of a longest path from 0 to $*$ containing v in the solution graph induced by a k-insertion of v, is determined as follows. First, compute sequences L_k and R_k. If $L_k \cap R_k = \emptyset$ (see Theorem 1), every k-insertion of v after the operations of L_k and before the operations of R_k is an optimal k-insertion, and the exact length of the longest path containing v is equal to $s_v^- + p_{vk} + t_v^-$. If $L_k \cap R_k \neq \emptyset$, consider $L_k \cap R_k$ sorted in increasing order of starting times. Let l be the cardinality of $L_k \cap R_k$ and let the function $\pi : \{1, ..., l\} \rightarrow L_k \cap R_k$ denote this order; hence $s_{\pi(1)} < s_{\pi(2)} < ... < s_{\pi(l)}$ but also $t_{\pi(1)} > t_{\pi(2)} > ... > t_{\pi(l)}$. The exact length $L_P(v, k, i)$ of the longest path passing through v when it is

inserted in k in the i^{th} position of $L_k \cap R_k$ ($i = 0$ means that v is inserted just before $\pi(1)$, $1 \leq i \leq l$ means that v is inserted just after $\pi(i)$) is,

$$L_p(v,k,i) = p_{vk} + \begin{cases} s_v^- + max\{p_{\pi(1)} + t^-_{\pi(1)}, t_v^-\} & \text{if } i = 0 \\ max\{s^-_{\pi(i)} + p_{\pi(i)}, s_v^-\} + max\{p_{\pi(i+1)} + t^-_{\pi(i+1)}, t_v^-\} & \text{if } 1 \leq i < l \\ max\{s^-_{\pi(i)} + p_{\pi(i)}, s_v^-\} + t_v^- & \text{if } i = l \end{cases} \tag{4}$$

and v is k-optimally inserted when equation (4) is minimized. If $L_k \cap R_k \neq \emptyset$ it is necessary to compute $s^-_{\pi(i)}$ and $t^-_{\pi(i)}$ for each $i \in \{1, ..., l\}$, which takes $O(N)$ time. In that situation, we use instead a less expensive approximation procedure to estimate $s^-_{\pi(i)}$ and $t^-_{\pi(i)}$. More precisely, to define the estimated value $\tilde{L}_P(v,k,i)$ of $L_P(v,k,i)$, we distinguish two cases.

(a) Machine k is different from the machine processing v in the current solution, i.e., $k \neq \mu(v)$.

(b) $k = \mu(v)$.

In case (a) we define $\tilde{L}_P(v,k,i)$ by simply replacing $s^-_{\pi(i)}$ and $t^-_{\pi(i)}$ with $s_{\pi(i)}$ and $t_{\pi(i)}$, and from (4) we get

$$\tilde{L}_P(v,k,i) = p_{vk} + \begin{cases} s_v^- + p_{\pi(1)} + t_{\pi(1)} & \text{if } i = 0 \\ s_{\pi(i)} + p_{\pi(i)} + p_{\pi(i+1)} + t_{\pi(i+1)} & \text{if } 1 \leq i < l \\ s_{\pi(i)} + p_{\pi(i)} + t_v^- & \text{if } i = l \end{cases} \tag{5}$$

since $\pi(i) \in L_k \cap R_k$.

In case (b), we adopt a procedure used by Dell'Amico and Trubian [13] for the job shop problem which takes $O(l)$ time. Hence, computing $\tilde{L}_P(v,k,i)$ takes $O(\log |Q_k| + l)$ time. The latter is on average a substantial gain over the $O(N)$ time needed in the exact case.

We call the k-insertion of v for which the estimated longest path is minimized, *approximate* optimal k-insertion. Over 17×10^6 trials and 172 different problems, we experimentally found out that an approximate optimal k-insertion is also an optimal k-insertion of v in 99% of the cases with a maximum absolute error of 1%.

In the following we want to study how "far" the upper bound $\tilde{L}_P(v,k,i)$ is from the correct value $L_P(v,k,i)$ in the worst case. Only case (a) ($k \neq \mu(v)$) is considered. Nevertheless, since the approximation procedure adopted in case (b) is more accurate than the one used for (a), the same bound holds also for case (b).

Theorem 2 $\tilde{L}_P(v,k,i) \leq 2L_P(v,k,i) + (p_{v\mu(v)} - p_{vk})$.

Proof. Let c and d denote the operations which are executed just before and after v, respectively, in the schedule of machine $\mu(v)$ in G. Let u and w

denote the operations which are executed on machine k just before and after v, respectively, when v is inserted in the i^{th} position of $L_k \cap R_k$. Let $\mathcal{P}(u)$ ($\mathcal{S}(w)$) be the set of operations belonging to any path between 0 and u (between w and $*$) in G. First note that it is not possible that $v \in \mathcal{P}(u)$ and $v \in \mathcal{S}(w)$ simultaneously, otherwise there would be a cycle in G. Therefore, in G three different situations are possible.

1. $v \in \mathcal{P}(u)$ and $v \notin \mathcal{S}(w)$. In this case $t_w^- = t_w$. If $i = 0$ then $L_P(v,k,0) = \tilde{L}_P(v,k,0)$. Otherwise, if d is not on the longest path between 0 and u in G, then $s_u^- = s_u$ and eq. (5) gives the exact value. Otherwise, $s_u - s_d$ is equal to the length of the longest path between d and u. Since after the k-insertion (u,w) of v the value of the longest path between d and u cannot change, we have $s_u^- \geqslant s_d^- + s_u - s_d$. Furthermore, $s_d - s_d^- = \max\left\{s_v + p_{v,\mu(v)}, s_{PJ[d]} + p_{PJ[d]}\right\} - \max\left\{s_c + p_c, s_{PJ[d]} + p_{PJ[d]}\right\}$, and as $\max\left\{s_v + p_{v,\mu(v)}, s_{PJ[d]} + p_{PJ[d]}\right\} \leqslant s_v^- + p_{v,\mu(v)} + \max\left\{s_c + p_c, s_{PJ[d]} + p_{PJ[d]}\right\}$, then $s_d - s_d^- \leqslant s_v^- + p_{v,\mu(v)}$. Thus,

$$\begin{aligned}\tilde{L}_P(v,i,k) &\leqslant s_d - s_d^- + \max\left\{s_u^- + p_u, s_v^-\right\} + p_{v,k} + \max\left\{p_w + t_w^-, t_v^-\right\} \\ &\leqslant s_v^- + p_{v,\mu(v)} + L_P(v,i,k) \\ &\leq 2L_P(v,k,i) + (p_{v\mu(v)} - p_{vk}).\end{aligned}$$

   since $s_v^- + p_{vk} \leq L_P(v,i,k)$.

2. $v \notin \mathcal{P}(u)$ and $v \in \mathcal{S}(w)$. Analogous to case 1, it is possible to prove that $\tilde{L}_P(v,i,k) \leqslant t_v^- + p_{v,\mu(v)} + L_P(v,i,k)$. Since $t_v^- + p_{vk} \leq L_P(v,i,k)$, the claim follows.

3. $v \notin \mathcal{P}(u)$ and $v \notin \mathcal{S}(w)$. In this case, $\tilde{L}_P(v,i,k) = L_P(v,i,k)$ since $s_u^- = s_u$ and $t_w^- = t_w$.

■

6 Neighborhood function

In this section we present two neighborhood functions ($Nopt1$, $Nopt2$) which can be used in local search methods for the FJSP.

In order to minimize the makespan, it may be profitable to consider only neighbors which are obtained by reinserting operations v that belong to a critical path in the solution graph of the current schedule. Indeed, a neighbor that is obtained by reinserting an operation v that does not belong to a longest path in the solution graph of the current schedule, does have a longest path in its solution graph which is at least as long as the longest path in the solution graph of the current schedule. The reason for this is that the longest path of the current schedule remains present in the solution graph of such a neighbor [22].

In order to reduce the neighborhood size we consider only a single critical path P of G. In P the preference is given to job arcs and it is obtained as

follows: the first operation of P is the first operation of any arbitrarily selected critical path of G; for each operation $v \in P$, if the immediate job successor $SJ[v]$ of v is a critical operation, then $SJ[v]$ belongs to P, otherwise the immediate machine successor $SM[v]$ of v is a critical operation that belongs to P (if $SJ[v]$ and $SM[v]$ are defined). By using this particular critical path P it is possible to prove some properties (see theorems 3 and 4) of $Nopt1$ and $Nopt2$.

Let $\tilde{S}_{vk}$ denote the solution of F_{vk} obtained from solution S by performing an approximate optimal k-insertion of v (see Section 5) obtained by considering only insertions on different positions than the current, i.e. $S \neq \tilde{S}_{vk}$.

The neighborhood set $Nopt1(S)$ is defined as the set of every $\tilde{S}_{vk}$, for each operation $v \in P$ and each machine $k \in M_v$. The proposed neighborhood $Nopt1(S)$ allows us to formulate a sufficient condition of optimality.

Theorem 3 *Let S be a feasible solution of a given FJSP. If $Nopt1(S) = \emptyset$ then S is an optimal solution.*

Proof. If $Nopt1(S) = \emptyset$, any operation $v \in P$ can be processed by only one machine, i.e., $M_v = \{k\}$; the set F_{vk} contains only the current solution, i.e., $F_{vk} = \{S\}$. Therefore, for each operation $v \in P$ it must be the case that $L_k \cap R_k = \emptyset$, otherwise at least two insertion points would be available for v, guaranteeing therefore that $Nopt1(S) \neq \emptyset$. Since v is a critical operation in G so that $L_k \cap R_k = \emptyset$, we will show that $PJ[v]$ and $SJ[v]$ are also critical operations in G. Indeed, at least one operation between $SJ[v]$ and $SM[v]$ must be critical. If $SM[v]$ is critical, since $L_k \cap R_k = \emptyset$ and $SM[v] \in R_k$, it follows that $SM[v] \notin L_k$. Therefore $p_{SM[v]} + t_{SM[v]} \leq t_v^- = p_{SJ[v]} + t_{SJ[v]}$. As $SM[v]$ is a critical operation it follows that $p_{SM[v]} + t_{SM[v]} = p_{SJ[v]} + t_{SJ[v]}$ and therefore $SJ[v]$ is also critical (analogous considerations hold to prove that $PJ[v]$ is a critical operation). Then, every operation of P belongs to the same job J_i, and the length of the critical path P is equal to the sum of the processing times of all the operations of J_i. Obviously, this value also defines a lower bound for the makespan of an optimal schedule and thus S is optimal. ■

The neighborhood set $Nopt2(S)$ is defined as an extension of $Nopt1(S)$ by adding for the current machine k of v all schedules of the set $F_{vk} \setminus \{S\}$. Since $Nopt1(S) \subseteq Nopt2(S)$ it follows from Theorem 3 that S is an optimal solution if $Nopt2(S) = \emptyset$.

Because of theoretical and practical reasons a fundamental question is whether a given neighborhood is *optimum connected* or not. This property is defined as follows.

Definition 2 *A neighborhood function $\mathcal{N}$ is called optimum connected if, from each solution S, there exists a finite sequence of solutions $(S_1, S_2, ..., S_k)$ with $S_0 = S$, S_k is optimal, and $S_{i+1} \in \mathcal{N}(S_i)$ $\forall i = 1, ..., k-1$.*

The optimum connectivity of a neighborhood function has consequences for local search algorithms. For instance, if a given neighborhood function is not optimum connected, then there are solutions for which no sequence of transitions

leads to an optimal solution. In this case any local search algorithm that starts with such an initial solution and makes only transitions to neighboring solutions is unable to find an optimal solution. In Theorem 4 it is shown that $Nopt2$ is optimum connected. The concept of *block* is used in the proof. A *block* is the maximal sequence, with cardinality greater than one, of adjacent operations of different jobs processed by the same machine and belonging to a critical path of a current solution.

Theorem 4 *The neighborhood function* $Nopt2$ *is optimum connected.*

Proof. The proof of this theorem is similar to the proof of a corresponding theorem in Hurink et al. [19]. In the following we provide only the differences. Consider an optimal solution S^*. In the neighborhood structure $Nopt2$, there is always a transition which allows an operation $v \in P$ to be reassigned to any machine in the set M_v. Thus, at most N transitions are necessary to get a solution for which all operations of all blocks on the critical path P are assigned to the same machines as in S^*; assume this situation in the following. Let $P = \{B_1, B_2, ..., B_l\}$ be the current critical path (previously defined) of l blocks. As in [19], it is sufficient to prove that any solution obtained by reversing two adjacent operations (b_x, b_{x+1}) $(1 \leqslant x \leqslant h-1)$ belonging to $B_j = (b_1, ..., b_x, ..., b_h)$ (for $j = 1, ..., l$) is one of $Nopt2(S)$. By the definition of $Nopt2(S)$, b_x can be inserted just after b_{x+1} if $t_{b_{x+1}} + p_{b_{x+1}} > t^-_{b_x}$ and $s_{SM[b_{x+1}]} + p_{SM[b_{x+1}]} > s^-_{b_x}$.

The first condition is verified since b_x and b_{x+1} are two adjacent and critical operations, hence $t_{b_{x+1}} + p_{b_{x+1}} = t_{b_x}$; furthermore $t_{b_x} \geqslant t^-_{b_x}$ and $t_{b_x} \neq t^-_{b_x}$ by the definition of P. Indeed, suppose, on the contrary, that $t_{b_x} = t^-_{b_x}$, then $t_{SJ[b_x]} + p_{SJ[b_x]} = t_{b_x}$ and it follows that $SJ[b_x]$ is a critical operation and, by the definition of P, operation x is the last operation of its block, i.e., $x = h$, which is a contradiction since $1 \leqslant x \leqslant h-1$.

The second condition is verified since $s_{SM[b_{x+1}]} + p_{SM[b_{x+1}]} \geqslant s_{b_{x+1}} = s_{b_x} + p_{b_x} > s^-_{b_x}$. ■

7 Search Procedure

A local search algorithm starts with some initial solution and moves from neighbor to neighbor in order to find better solutions. The main problem with this strategy is to escape from local minima where the search cannot find any further neighborhood solution that decreases the objective function value.

Different strategies have been proposed to solve this problem. One of the most efficient strategies for job shop problems is tabu search [18]. Tabu search allows the search to explore solutions that do not decrease the objective function value only in those cases where these solutions are not forbidden. This is usually obtained by keeping track of the last solutions in term of the action used to transform one solution to the next. When an action is performed it is considered *tabu* for the next T iterations, where T is the tabu status length. A solution is forbidden if it is obtained by applying a tabu action to the current solution.

Two tabu search algorithms were implemented by employing neighborhoods $Nopt1$ and $Nopt2$, respectively. We compared the $Nopt1$-version and $Nopt2$-version, and we experimentally found that the $Nopt1$-version is as good as, but faster than the $Nopt2$-version since a smaller neighborhood size is used. Therefore, in our experience and for the considered problem, the connectivity of the neighborhood is only of theoretical but not of practical interest. The following refers to the $Nopt1$-version only.

When $Nopt1$ is used, an action is composed of a couple (v, k), where v is the operation being moved and k is the machine operation to which v is assigned before the move. In order to keep track of the actions performed, we use a $N \times m$ matrix TM. Each time, action (v, k) is performed, $TM(v, k)$ is set to $iter + T$, where $iter$ is the current iteration number and T is defined in the following. An action (v, k) is tabu if $TM(v, k) > iter$. The tabu status length T is crucial to the success of the tabu search procedure, and we propose a self-tuning procedure based on empirical evidence. T is dynamically defined for each operation v and each solution. It is equal to the number of operations of the current critical path P plus the number of alternative machines available for operation v, i.e., $T := |P| + |M_v|$. We choose this empirical formula since it summarizes, to some extent, the features of the given problem instance and those of the current solution. For instance, there is a certain relationship between $|P|$ and the instance size, between $|P|$ and the quality of the current solution. In order to diversify the search it may be unprofitable to repeat the same action often if the number of candidate actions is "big" or the solution quality is low, in some sense, when $|P|$ is "big". Furthermore, the tabu status length of v is augmented by $|M_v|$ in order to diversify the machine assignment of v.

We denote as *best move* the one with the smallest estimated length of the new longest path containing the moved operation. If several non-tabu moves exist, the next one is randomly chosen between the best two non-tabu moves. This method is useful to decrease the probability of generating cycles. In order to explore the search space in a more efficient way, tabu search is usually augmented with some aspiration criteria. The latter are used to accept a move even if it has been marked tabu. In the present case a move of operation v is always accepted if the estimated length of the new longest path containing v decreases the best makespan obtained so far (when several moves satisfy the previous condition, the best one is chosen). Finally, when only tabu moves are available, the chosen solution is the one (v, k) with the lowest value of $TM(v, k)$.

Before presenting the computational results we describe how the first feasible solution is computed. We put the first operation which has not yet been scheduled of a randomly chosen job at the end of a permissible and randomly chosen machine.

8 Computational results

The $Nopt1$-version of the search procedure described in Section 7 was implemented in C++ on a 266 MHz Pentium and tested on a large number of problem

instances from the literature. The results obtained with our procedure (denoted $TSopt1$) were then compared with results obtained with other procedures.

First we discuss the flexible job shop scheduling instances we use in our computational comparisons. Several sets of problem instances were considered.

- The first data set (BRdata) comes from Brandimarte [6]. The data were randomly generated using a uniform distribution between given limits. They consist of ten problems where the number of jobs ranges from 10 to 20, the number of machines ranges from 6 to 15, operations for each job range from 5 to 15 and the maximum number of equivalent machines per operation ranges from 3 to 6.

- The second test sample (DPdata) comes from Dauzère-Pérès and Paulli [11]. The data consist of 18 test problems where the number of jobs ranges from 10 to 20, machines range from 5 to 10, operations for each job range from 5 to 25. The set of machines capable of performing an operation was constructed by letting a machine be in that set with a probability that ranges from 0.1 to 0.5.

- The third data set (BCdata) comes from Barnes and Chambers [4]. The data were constructed from three of the most challenging classical job shop problems [16, 23] (mt10, la24, la40) by replicating machines selected according to two simple criteria: the total processing time required by a machine and the cardinality of critical operations on a machine. The set consists of 21 test problems and the processing times for operations on replicated machines are assumed to be identical to the original.

- The fourth test sample (HUdata) comes from Hurink et al. [19]. The problems are generated by modifying benchmark problems for the classical job shop problem. More specifically, they were obtained from three problems by Fisher and Thompson [16] (mt06, mt10, mt20) and 40 problems from Lawrence [23] (la01–la40). Each set M_{ij} is equal to the machine to which operation O_{ij} is assigned in the original problem, plus any of the other machines with a given probability. Depending on this probability, Hurink et al. generated three sets of test problems: *edata, rdata* and *vdata*. The first set contains the problems with the least amount of flexibility, whereas the average size of M_{ij} is equal to 2 in rdata and $m/2$ in vdata.

The non-deterministic nature of our algorithm makes it necessary to carry out multiple runs on the same problem instance in order to obtain meaningful results. We ran our algorithm five times from different starting solutions. The results obtained with our procedure were then compared with results obtained with all procedures for which we could find results (in terms of makespan and CPU time) in the literature. We use the following notation for those procedures: BR stands for the best tabu search procedure of Brandimarte [6]. DP stands for a tabu search procedure of Dauzère-Pérès and Paulli [11]. BC stands for a tabu search procedure of Barnes and Chambers [4]. HJT stands for a tabu

search procedure of Hurink et al. [19]. BN stands for a tabu search procedure of Brucker and Neyer [7]. For HUdata the best results of the combined effort of HJT, BN and DP are denoted as HBD.

The number of iterations was chosen experimentally in order to ensure a compromise between the running time and solution quality. We limited the number of iterations to 10^5 for BRdata and HUdata and to 4×10^5 for DPdata and BCdata.

The acronym CI-CPU stands for computer-independent CPU times. These values were computed using the normalization coefficients of Dongarra [14], as interpreted by Vaessens et al. [28].

For the HUdata problems we consider the lower bounds computed by Jurisch [21]. For the remaining data sets only lower bounds computed in a straightforward way are available, therefore they are generally not very close to the optimum. We compute the relative error for each algorithm and each instance; i.e. the percentage by which the best solution value (UB) obtained is above the best known lower bound (LB), that is 100(UB/LB). MRE denotes the mean relative error for the set of problems reported. For our procedure MRE: best out of 5 runs, MRE_{av}: average out of five runs.

Table 1 shows our computational results. The name and size of data sets are given in column 1 and 2. Our procedure outperforms the combined effort of the other procedures. Comparing the best value with average value, we can see that algorithm $TSopt1$ is quite robust (i.e. the differences between the solution values of the runs are small) and the quality of the solutions is weakly dependent upon the random choices and the poor starting solutions.

More information on data set HUdata is shown in table 3. The test problems in edata seem to be more difficult than the test problems in rdata and vdata. For some of the test problems in edata, the deviation from the lower bound is quite high, e.g. la36-la40. Admittedly, this might be caused by poor lower bound quality for these problems, but it might also come from the fact that the test problems get easier as the flexibility increases.

A comparative overview of the $TSopt1$ average makespan and CI-CPU out of 5 runs is given in Table 2. Column 4 (CI-CPU) gives the sum of the CI-CPU times used to compute all the solutions except for our procedure where it is the sum of the average CI-CPU out of five runs. Column 5 (B:E:W) represents the number of examples for which $TSopt1$ average makespan is better (B), equal (E) or worse (W) than the best makespan found by the procedure of column 2. The major message of table 2, is that $TSopt1$ significantly outperforms all the other procedures both in terms of average solution quality and computing time. With reference to HUdata, our average computational effort is about 15, 30 and 25 times smaller than the corresponding CI-CPU of HJT, BN and DP, respectively, and comparing our average solution values with the best known upper bounds, in 129 test problems we obtain 74 better results, while in only 5 cases our average solution values are worse. Furthermore, $TSopt1$ is about 5 and 20 times faster than DP and BC on DPdata and BCdata, respectively. Computing times for BRdata are not reported in [6] and therefore a comparison of the computational effort for these runs is not possible. Additional details on

the computational experience reported here can be found in [24].

Considering our best results we note that $TSopt1$ found 120 new better upper bounds and 77 optimal solutions over 178 benchmark problems and it was outperformed in only one problem instance. Furthermore, it should be noted that by increasing the number of iterations, it is possible to eliminate that case. The computing times are generally small (no run required more that 3 minutes for DPdata and 30 seconds for the remaining problems).

Since the classical job-shop scheduling problem is a special case of the flexible job shop problem, our tabu search procedure can also be used to solve these problems. We tested our procedure on the original 43 test problems [16, 23] (sdata) used to generate test sample BCdata and HUdata. For this set of problems we ran $TSopt1$ 10 times from different starting solutions and limited the number of iterations to 10^6. Our tabu-search algorithm finds an optimal solution for 38 of 43 problems. With regard to the 5 remaining problems, the distance from the best lower bound (or optimum, if known) is smaller than 0.1%. It is worth noting that our procedure is able to find the optimal solution to the notorious 10×10 problem (mt10) by Fisher and Thompson in just 0.01 seconds starting from a solution with makespan equal to 1686. In order to make a more detailed comparison on problems for which it is meaningful, we selected the 13 most difficult instances, according to Balas and Vazacopoulos [3], among the 43 problems. Table 4 contains comparisons related to the best makespans found by $TSopt1$, against those from DT (the tabu search procedure of Dell'Amico and Trubian [13]), NS (the tabu search procedure of Nowicki and Smutnicki [25]) and BV (the procedure SB-GLS2 of Balas and Vazacopoulos [3]). Our results are close to the ones obtained by the best search procedures to solve the job shop problems, although the computational time used by $TSopt1$ is considerably greater. Indeed, on average $TSopt1$ is 6 times slower than BV. However, our tabu search procedure is quite simple and it is an obvious topic for further computational experiments to analyze how more sophisticated search strategies could improve the computational time.

9 Conclusion

In this paper, two new neighborhood functions with several properties for solving the flexible job shop scheduling problem are presented. Due to their special structure, our tabu search based algorithm works faster and more efficiently than other known algorithms. Furthermore the procedure is quite simple. Our primary objective is to show that the proposed neighborhood structure leads to an efficient heuristic for the FJSP. A large sample of computational experiments is presented and we found 120 new better upper bounds and 116 optimal solutions over 221 benchmark problems.

Future research directions include the analysis of the procedure performance when features of tabu search such as long-term memory and backtracking memory are considered.

The neighborhood structures presented here can be extended to solve the multi-resource shop scheduling with resource flexibility [12] and the multi-mode job shop problem [7]. Both problems can be seen as special cases of Resource Constrained Project Scheduling Problem where multiple modes are allowed [9].

Acknowledgment. Thanks are due to Marco Zaffalon, Andrea Rizzoli, Fred Cummins and two anonymous referees for comments that helped improve the paper.

References

[1] Aarts, E.H.L., Lenstra, J.K., (1995): Local Search in Combinatorial Optimization, Wiley, Chichester.

[2] Akella, R., Choong, Y., (1984): "Performance of a hierarchical production scheduling policy", IEEE Trans. Components, Hybrids and Manufacturing Technology, 225–248.

[3] Balas, E., Vazacopoulos, A., (1998): "Guided Local Search with Shifting Bottleneck for Job Shop Scheduling", Management Science 44, 262-275.

[4] Barnes, J. W., Chambers, J. B., (1996): "Flexible Job Shop Scheduling by Tabu Search", Graduate Program in Operations Research and Industrial Engineering, The University of Texas at Austin, Technical Report Series, ORP96-09, http://www.cs.utexas.edu/users/jbc/.

[5] Bona B., Brandimarte P., (1990): "Hybrid hierarchical scheduling and control systems in manufacturing", IEEE Trans. Robotics and Automation, 673-686.

[6] Brandimarte, P., (1993): "Routing and scheduling in a flexible job shop by tabu search", Annals of Operations Research 22, 158-183.

[7] Brucker, P., Neyer, J., (1998): "Tabu-search for the multi-mode job-shop problem" , OR Spektrum 20, 21-28.

[8] Brucker, P., Schlie R., (1990): "Job-shop scheduling with multi-purpose machines", Computing 45, 369-375.

[9] Brucker, P., Drexl, A., Möhring, R., Neumann, K., Pesch, E., (1999): "Resource-Constrained Project Scheduling: Notation, Classification, Models, and Methods", European J. Operational Research 112, 3-41.

[10] T. H. Cormen, C.E. Leiserson, R. L. Rivest, (1991): Introduction to Algorithms, MIT Press.

[11] Dauzère-Pérès S., Paulli, J., (1997): "An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search", Annals of Operations Research 70, 281-306.

[12] Dauzère-Pérès S., Roux J., Lasserre J.B., (1998): "Multi-resource shop scheduling with resource flexibility", European Journal of Operational Research 107, 289-305.

[13] Dell'Amico M., Trubian, M., (1993): "Applying tabu-search to the job shop scheduling problem", Annals of Operations Research 22, 15-24.

[14] Dongarra, J.J., (1998): "Performance of various computers using standard linear equations software", Computer Science Department, University of Tennessee, Knoxville, Tennessee.

[15] Escudero, L.F., (1989): "A mathematical formulation of a hierarchical approach for production planning in FMS", Modern Production Management Systems, 231-245.

[16] Fisher, H., Thompson, G.L., (1963): "Probabilistic learning combinations of local job shop scheduling rules", in: Industrial Scheduling, J.F. Muth and G.L. Thompson, eds., Prentice Hall, Englewood Cliffs, 225-251.

[17] Garey M. R., Johnson D.S., Sethi R., (1976): "The complexity of flowshop and jobshop scheduling", Math. Oper. Res. 1, 117-129.

[18] Glover, F., (1989): "Tabu search - Part I", ORSA Journal on Computing 1, 190-206.

[19] Hurink,E., Jurisch, B., Thole, M., (1994): "Tabu search for the job shop scheduling problem with multi-purpose machine", Operations Research Spektrum 15, 205-215.

[20] Jansen K., Mastrolilli M., Solis-Oba R., (1999) "Approximation Algorithms for Flexible Job Shop Problems", to appear in the Proceedings of Latin American Theoretical Informatics, LATIN'2000.

[21] Jurisch, B., (1992): "Scheduling Jobs in Shops with Multi-purpose Machines", Ph.D. thesis, Fachbereich Mathematik/Informatik, Universitat Osnabruck.

[22] Van Laarhoven P.J.M., Aarts E.H.L., Lenstra J.K. (1992),"Job Shop Scheduling by simulated annealing", Operations Research 40, 113-125.

[23] Lawrence, S., (1984): Supplement to "Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques", GSIA, Carnegie Mellon University, Pittsburgh, PA.

[24] Mastrolilli, M., Gambardella, L.M., (1998): "Effective Neighborhood Functions for the Flexible Job Shop Problem: appendix", http://www.idsia.ch/~monaldo/fjsp.html.

[25] Nowicki, E., Smutnicki, C., (1996): "A fast taboo search algorithm for the job shop problem", Management Science 42, 797-813.

[26] Stecke, K.S.,(1983): "Formulation and solution of nonlinear integer production planning problems for flexible manufacturing systems", Management Science 29, 273-288.

[27] Vaessens, R.J.M., (1995): Generalized Job Shop Scheduling: Complexity and Local Search, Ph.D. thesis, Eindhoven University of Technology.

[28] Vaessens, R.J.M., Aarts, E.H.L., Lenstra, J.K., (1996): "Job shop scheduling by local search", INFORMS Journal on Computing, vol.8 n.3.

10 Figures

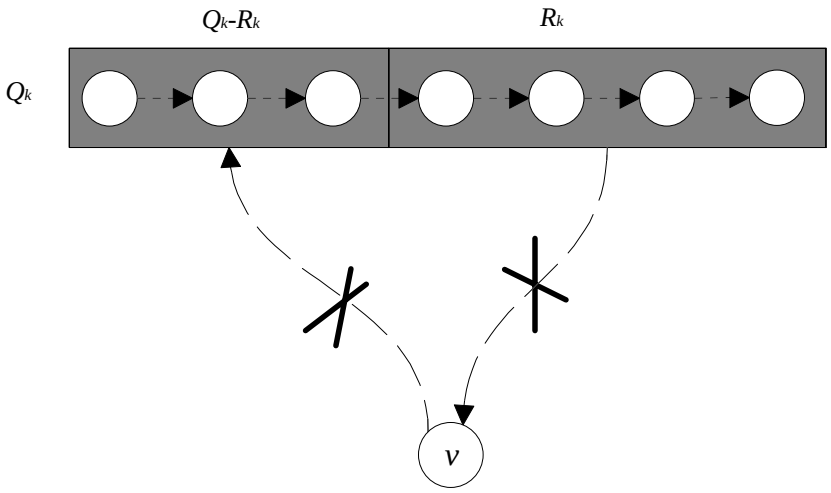

Figure 1: Properties of R_k and $Q_k \backslash R_k$ in G^-.

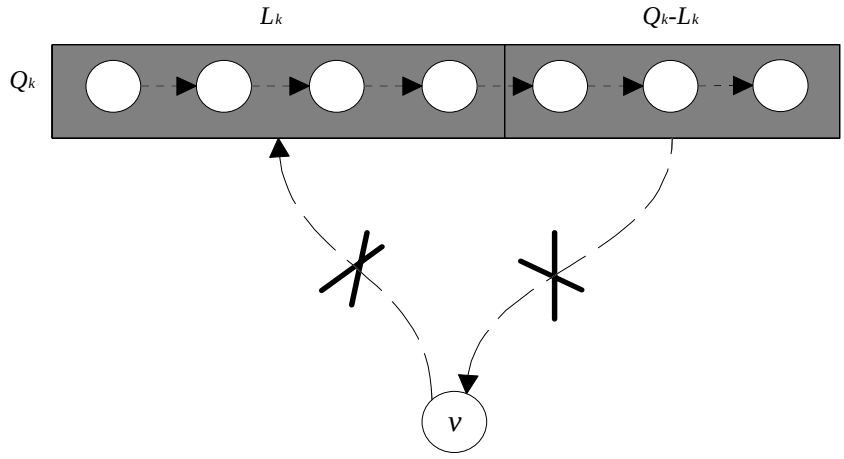

Figure 2: Properties of L_k and $Q_k \backslash L_k$ in G^-.

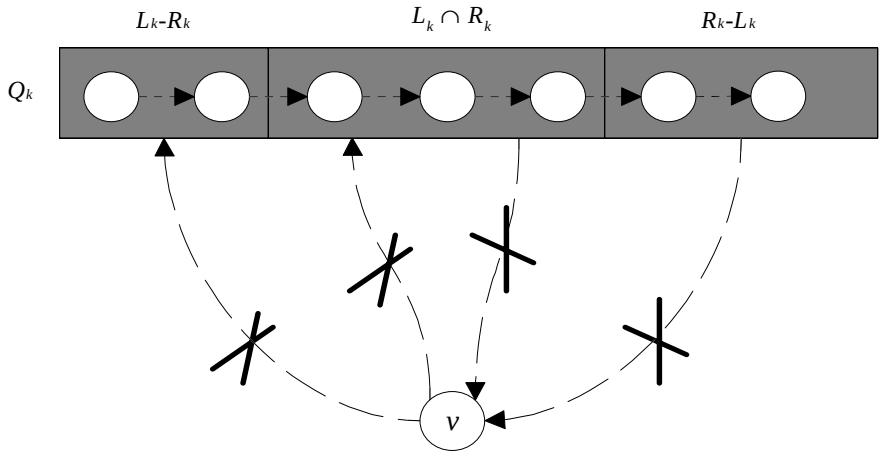

Figure 3: Properties of $L_k \cap R_k$ in G^-.

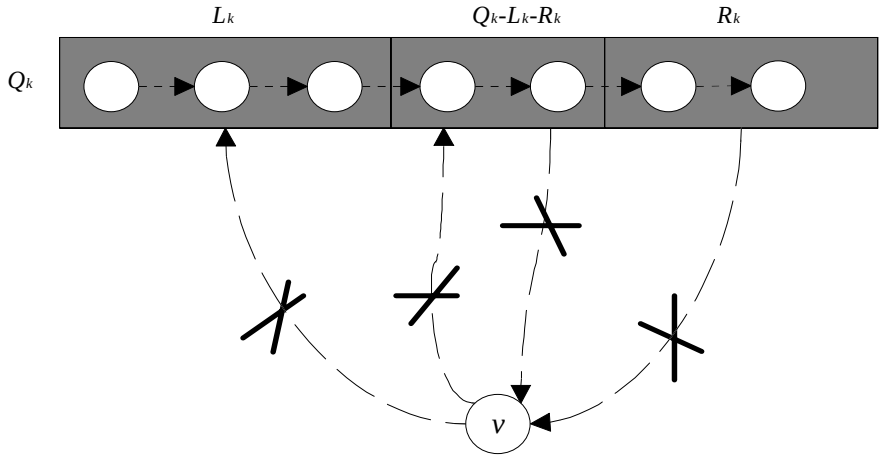

Figure 4: Properties of $Q_k \backslash (L_k \cup R_k)$ in G^-.

11 Tables

Data Set	#	Procedure	MRE (MRE_{av})
BRdata	10	BR	141.44
		$TSopt1$	115.41 (115.83)
DPdata	18	DP	103.31
		$TSopt1$	102.01 (102.24)
BCdata	21	BC	123.44
		$TSopt1$	122.53 (122.64)
edata	43	HBD	102.98
		$TSopt1$	102.17 (102.27)
rdata	43	HBD	101.70
		$TSopt1$	101.23 (101.40)
vdata	43	HBD	100.25
		$TSopt1$	100.10 (100.14)

Table 1: Mean Relative Error (MRE) over Best Known Lower Bound.

Data Set	#	Procedure	CI-CPU	B:E:W
BRdata	10	BR	–	9 : 1 : 0
		$TSopt1$	74	
DPdata	18	DP	13981	17 : 0 : 1
		$TSopt1$	2467	
BCdata	21	BC	7760	16 : 1 : 4
		$TSopt1$	356	
HUdata	129	HJT	258750	102 : 27 : 0
		BN	531056	107 : 22 : 0
		DP	428410	77 : 47 : 5
		$TSopt1$	16568	

Table 2: Summary results.

Problem Class	Jobs/Machines	edata		rdata		vdata	
		HBD	$TSopt1$ (Avg)	HBD	$TSopt1$ (Avg)	HBD	$TSopt1$ (Avg)
mt6/10/20	6/6 10/10 5/20	100.82	100.00 (100.10)	100.41	100.34 (100.36)	100.00	100.00 (100.00)
la01-05	10/5	100.15	100.00 (100.00)	100.46	100.11 (100.24)	100.33	100.00 (100.11)
la06-10	15/5	100.08	100.00 (100.00)	100.15	100.03 (100.08)	100.10	100.00 (100.03)
la11-15	20/5	100.29	100.29 (100.29)	100.04	100.02 (100.02)	100.00	100.00 (100.01)
la16-20	10/10	100.70	100.00 (100.00)	101.91	101.73 (101.77)	100.00	100.00 (100.00)
la21-25	15/10	107.04	105.62 (105.93)	105.29	103.82 (104.38)	101.32	100.70 (100.85)
la26-30	20/10	105.66	103.47 (103.76)	101.34	100.59 (100.76)	100.32	100.11 (100.18)
la31-35	30/10	100.82	100.30 (100.32)	100.32	100.09 (100.14)	100.10	100.01 (100.03)
la36-40	15/15	110.43	108.99 (109.13)	104.86	103.97 (104.47)	100.00	100.00 (100.00)

Table 3: Mean Relative Error on HUdata.

Problem	BV	$TSopt1$	NS	DT	BC
mt10	930	930	930	935	935
la02	655	655	655	655	655
la19	842	842	842	842	843
la21	1046	1046	1047	1048	1053
la24	935	935	939	941	946
la25	977	977	977	979	988
la27	1235	1235	1236	1242	1256
la29	1164	1160	1160	1182	1194
la36	1268	1269	1268	1278	1278
la37	1397	1397	1407	1409	1418
la38	1196	1196	1196	1203	1211
la39	1233	1233	1233	1242	1237
la40	1224	1228	1229	1233	1239

Table 4: Makespan for 13 hard problems.