ELSEVIER

Discrete Applied Mathematics 65 (1996) 97–122

DISCRETE APPLIED MATHEMATICS

Improving local search heuristics for some scheduling problems—I

Peter Brucker[a,*], Johann Hurink[a,1], Frank Werner[b,2]

[a] *F.B. Mathematik/Informodik, Universität Osnabrück, 49069 Osnabrück, Germany*

[b] *Institut für Math. Optimierung, Otto-von-Guericke-Universität, 39016 Magdeburg, Germany*

Received 22 December 1992; revised 17 December 1993

Abstract

Local search techniques like simulated annealing and tabu search are based on a neighborhood structure defined on a set of feasible solutions of a discrete optimization problem. For the scheduling problems $P2 \| C_{max}$, $1|prec|\sum C_i$ and $1 \| \sum T_i$ we replace a simple neighborhood by a neighborhood on the set of all locally optimal solutions. This allows local search on the set of solutions that are locally optimal.

1. Introduction

In this paper we consider certain NP-hard scheduling problems. The problems can be formulated as *discrete optimization* problems, which can be described as follows. For a given finite set $\mathscr{S}$ and a given function $c: \mathscr{S} \to \mathbb{R}$ one has to find a solution $s^* \in \mathscr{S}$ with

$$c(s^*) \leqslant c(s) \quad \text{for all } s \in \mathscr{S}.$$

In general, the set $\mathscr{S}$ is specified in some implicit form. It is called a *feasible set*.

Problems of this type can be solved either by exact methods like branch and bound or dynamic programming or by heuristic methods. Popular heuristics are local search methods like simulated annealing (see [11]) and tabu search (see [1, 4] or [3]). These methods depend on an underlying neighborhood structure. Usually, the quality of the neighborhood structure has some important influence on the methods.

* Corresponding author.

[1] Supported by Deutsche Forschungsgemeinschaft (Project JoPTAG).

[2] Supported by Alexander von Humboldt Stiftung.

SSDI 0166-218X(95)00030-5

In general, local search is an iterative procedure that moves from one solution $s \in \mathcal{S}$ to another solution repeating this step as long as it seems to be necessary. The possible moves from s to the next solutions are restricted by a set OP of possible operators op. For each $op \in OP$ the set $\mathcal{S}^{op}$ denotes that subset $\mathcal{S}$ for which op is defined. So each operator $op \in OP$ is a function $op: \mathcal{S}^{op} \to \mathcal{S}$. Thus,

$$\mathcal{N}(s) := \{op(s) \mid op \in OP,\ s \in \mathcal{S}^{op}\} \tag{1.1}$$

is the set of all possible *neighbors* of s. A neighborhood on the set $\mathcal{S}$ is now defined by the sets

$$\mathcal{N}(s), \quad s \in \mathcal{S}.$$

The simplest local search is the method of *iterative improvement* that chooses the best solution in $\mathcal{N}(s)$ as the solution to move to from s. It stops if no solution in $\mathcal{N}(s)$ improves the solution s. In this case s is a *local optimum*. Unfortunately, the value $c(s)$ of a local optimum may be far away from the optimal value. To avoid this problem, simulated annealing and tabu search allow moves to nonimproving solutions. Still a disadvantage of these methods is an oscillation around local optima, which results in a slow convergence.

Our approach to overcome these difficulties is to replace the original feasible set $\mathcal{S}_1$ by the subset $\mathcal{S}_2$ of all $s \in \mathcal{S}_1$ that are locally optimal with respect to a neighborhood $\mathcal{N}_1(s)$, $s \in \mathcal{S}_1$ on the set $\mathcal{S}_1$. Furthermore, we construct a new operator set OP_2, that defines a new neighborhood $\mathcal{N}_2(s)$, $s \in \mathcal{S}_2$ on the set $\mathcal{S}_2$. These operators are based on polynomially time algorithms for constructing certain locally optimal solutions.

The advantages of such an approach are:

- the search space is reduced considerably,
- local search methods can be still applied (at a higher level),
- oscillations appear only at a higher level,
- structural properties are taken into consideration.

The construction of the operator set OP is problem specific. Thus, the method is not a general purpose method. Different problems have to be treated differently.

Independently from our investigations, Martin et al. [7] considered iterated local search for the traveling salesman problem, i.e. they apply a sampling method to the locally optimal solutions. Sometimes large steps, which they call a kick, are performed and then usual local search is applied. Thus, this algorithm operates also only with locally optimal solutions. A similar idea was used by Ulder [10] to improve a genetic local search algorithm for the job shop problem. However, contrary to our approach the determination of a locally optimal solution in these approaches is not necessarily polynomially bounded. Also connectivity properties have not been investigated.

A neighborhood is *strongly connected* if for any two feasible solutions s_1 and s_2 solution s_2 is reachable from solution s_1 by a sequence of moves. Connectivity is an important property from a theoretical point of view, since convergence proofs for

simulated annealing depend on such a property (see [11]). Also experiments have shown that in general a good neighborhood should be connected (see [9]).

In the next three sections we apply our approach to the scheduling problems:

$$P2 \,\|\, C_{\max}, 1 \,|\, prec \,|\, \sum C_i, 1 \,\|\, \sum T_i.$$

In each case we will define a strongly connected neighborhood $\mathcal{N}_1(s), s \in \mathcal{S}_1$ on the set $\mathcal{S}_1$ of all feasible solutions and construct a corresponding secondary neighborhood on the set $\mathcal{S}_2$ of all local optima with respect to $\mathcal{N}_1$. Furthermore, we will show that the defined secondary neighborhoods are strongly connected.

2. The problem P2 $\|$ $C_{\max}$

$P2 \,\|\, C_{\max}$ denotes the problem of scheduling n jobs $i = 1, \ldots, n$ with processing times p_i $(i = 1, \ldots, n)$ on two identical parallel machines such that the makespan is minimized. A feasible solution of this scheduling problem is given by a partitioning of the job set $I = \{1, \ldots, n\}$ into two disjoint sets I_1 and I_2. We denote such a partitioning by (I_1, I_2). I_v is the set of jobs to be processed on machine M_v $(v = 1, 2)$. For $v = 1, 2$, let $s_v := \sum_{i \in I_v} p_i$ the total processing time on machine M_v. Then $\max\{s_1, s_2\}$ is the makespan of the schedule defined by (I_1, I_2). We have to find a partitioning (I_1, I_2) such that

$$\max\{s_1, s_2\} = \max\left\{ \sum_{i \in I_1} p_i, \sum_{i \in I_2} p_i \right\}$$

is minimized. The problem is shown to be NP-hard by a simple reduction from the partitioning problem.

For this problem a neighborhood $\mathcal{N}_1$ is defined on the set $\mathcal{S}_1$ of all feasible solutions (I_1, I_2) by the operators $move(i)$ $(i = 1, \ldots, n)$. $move(i)$ moves job i from the machine on which i is scheduled to the other machine, i.e.

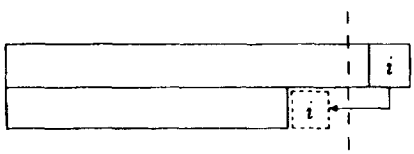

Fig. 1. $p_i \leq \Delta/2$.

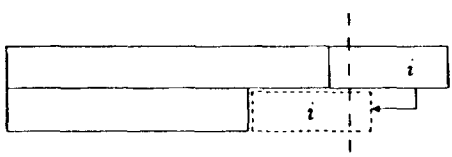

Fig. 2. $p_i > \Delta/2$.

$$move(i)(I_1, I_2) = \begin{cases} (I_1 \setminus \{i\}, I_2 \cup \{i\}) & \text{if } i \in I_1, \\ (I_1 \cup \{i\}, I_2 \setminus \{i\}) & \text{if } i \in I_2. \end{cases}$$

Each operator may be applied to each feasible solution, i.e. we have

$$\mathscr{S}_1^{\text{move}(i)} = \mathscr{S}_1.$$

Let (I_1, I_2) be a feasible solution with $s_1 \leqslant s_2$. Then (I_1, I_2) is locally optimal with respect to $\mathscr{N}_1$ if and only if for all $i \in I_2$ we have $p_i \geqslant \Delta := s_2 - s_1$.

Given a feasible solution (I_1, I_2) and a sequence π of all jobs, a locally optimal solution can be calculated by a procedure *localopt*(π) with the following basic step.

If $s_1 < s_2$ $(s_2 < s_1)$ and there exists an $i \in I_2$ $(i \in I_1)$ with $p_i < \Delta$ then in π we choose the first $i \in I_2$ $(i \in I_1)$ with $p_i < \Delta$ and replace (I_1, I_2) by $move(i)(I_1, I_2)$. After that move we update s_1, s_2 and Δ. This is done by considering two possible cases.

Case 1: $p_i \leqslant \Delta/2$. In this case we have (see Fig. 1)

$$\Delta_{\text{new}} := \Delta - 2p_i \geqslant 0. \tag{2.1}$$

Case 2: $p_i > \Delta/2$. We have (see Fig. 2)

$$\Delta_{\text{new}} := 2p_i - \Delta > 0. \tag{2.2}$$

In both cases we have

$$\Delta_{\text{new}} < \Delta. \tag{2.3}$$

In the first case this follows immediately from (2.1). In the second case we have

$$\Delta_{\text{new}} := p_i - (\Delta - p_i) < p_i < \Delta. \tag{2.4}$$

Notice, that such a step improves the objective value. We repeat these steps as long as an improvement is possible.

Theorem 2.1. *The procedure localopt*(π) *calculates a local optimum in at most* n^2 *iterations.*

Proof. W.l.o.g. assume that $s_1 < s_2$. If there exists no job $i \in I_2$ with $p_i < \Delta$, then a movement of any job onto the other machine does not decrease the Δ-value. Thus, we have a local optimum if the procedure stops.

If $p_i > \Delta/2$ then $\Delta_{\text{new}} < p_i$ holds (see (2.4)). Furthermore, due to (2.3) the Δ-values are decreasing. Thus, we have $\Delta < p_i$ for all further iterations which means that job i is never moved again.

On the other hand, if a Case 1 movement of job i is done a second time then this is only possible if there is at least one Case 2 movement of some job in between. Therefore, the total number of movements is at most n^2. □

Next we define a secondary neighborhood on the set $\mathscr{S}_2$ of all feasible solutions that are locally optimal. The corresponding set OP_2 of operators is given by

$$OP_2 = \{localopt(\pi^*) \circ move(i) \mid i = 1, \ldots, n\}, \tag{2.5}$$

where "$\circ$" denotes the composition of operators and π^* is a special permutation of the jobs. The operators (2.5) can be applied to all solutions from $\mathscr{S}_2$. They move a job i and apply the procedure $localopt(\pi^*)$ to transform the new solution again into a locally optimal solution. π^* is the unique sequence with

$$\pi^*(1) \prec \pi^*(2) \prec \cdots \prec \pi^*(n),$$

where

$$i \prec j \quad \text{if and only if } (p_i, i) \text{ is lexicographically smaller than } (p_j, j). \tag{2.6}$$

With such a choice of π^*, connectivity of the secondary neighborhood can be estblished easily.

Theorem 2.2. *The secondary neighborhood is strongly connected.*

Proof. We show that for arbitrary locally optimal solutions (I_1, I_2) and (J_1, J_2) there exists a sequence of operators (2.5) that transforms (I_1, I_2) into (J_1, J_2).

Such a sequence is constructed by moving step by step the largest job i with respect to (2.6) that is not on the right machine onto the opposite machine and applying $localopt(\pi^*)$ to this new partition.

We show that this procedure terminates by proving that during the procedure $localopt(\pi^*)$ no job j with $i \prec j$ (i.e. no job j that is greater or equal to i with respect to (2.6)) is moved.

Assume that j is the first job with $i \prec j$ that is moved from a current set, say I_1, to I_2 when applying $localopt(\pi^*)$. Due to the definition of π^*, the current set I_1 cannot contain a job $l \prec j$. Furthermore, i is the largest job in $(I_1 \setminus J_1) \cup (I_2 \setminus J_2)$. Thus, all jobs $l \in I_1$ with $i \prec l$ belong to J_1. Thus, $I_1 \subseteq J_1$. Because $j \in I_1$ is moved, we must have

$$\sum_{i \in J_1} p_i \geqslant \sum_{i \in I_1} p_i > \sum_{i \in I_2} p_i \geqslant \sum_{i \in J_2} p_i.$$

This contradicts the fact that (J_1, J_2) is locally optimal because moving j would also improve (J_1, J_2). □

3. The $1 \mid prec \mid \sum C_i$ problem

$1 \mid prec \mid \sum C_i$ denotes the problem of scheduling n jobs $1, \ldots, n$ with processing times p_i $(i = 1, \ldots, n)$ on one machine such that the mean flow time is minimized. Between the jobs precedence constraints $\rightarrow$ are given (a precedence constraint $i \rightarrow j$

expresses that job i has to be processed before job j). Lawler has shown that this problem is NP-hard [6]. A feasible schedule for this problem is given by a sequence (permutation) $\pi = (\pi_1, \ldots, \pi_n)$ of the jobs that is compatible with the precedence constraints. The corresponding completion times C_i for the jobs π_i are given by $\sum_{j=1}^{i} p_{\pi_j}$ $(i = 1, \ldots, n)$. We have to find a feasible sequence π such that

$$\sum_{i=1}^{n} C_i = \sum_{i=1}^{n} \sum_{j=1}^{i} p_{\pi_j}$$

is minimized.

For this problem a neighborhood is defined by the set $\mathscr{S}_1$ of all feasible sequences π and by the "adjacent pairwise interchange" operators $api(i)$ $(i = 1, \ldots, n-1)$. The operator $api(i)$ will interchange the jobs that are scheduled in position i and $i+1$, i.e. the sequence $\pi' = api(i)(\pi)$ is defined by

$$\pi'_k = \begin{cases} \pi_{i+1} & \text{if } k = i, \\ \pi_i & \text{if } k = i+1, \\ \pi_k & \text{otherwise.} \end{cases}$$

The operator $api(i)$ maps a feasible schedule $\pi \in \mathscr{S}_1$ into a feasible schedule if and only if there does not exist a precedence constraint $\pi_i \to \pi_{i+1}$, i.e. we may apply $api(i)$ only to sequences from the set

$$\mathscr{S}_1^{api(i)} = \{\pi \in \mathscr{S}_1 \mid \pi_i \to \pi_{i+1} \text{ is not a precedence constraint}\},\ i = 1, \ldots, n-1.$$

We define the neighborhoood $\mathscr{N}_1$ by

$$\mathscr{N}_1(\pi) = \{api(i)(\pi) \mid \pi \in \mathscr{S}_1^{api(i)}, i = 1, \ldots, n-1\},\ \pi \in \mathscr{S}_1.$$

Let $\pi \in \mathscr{S}_1$ be a feasible solution. Due to Smith's rule an interchange of jobs π_i and π_{i+1} will reduce the mean flow time if and only if $p_{\pi_i} > p_{\pi_{i+1}}$. Therefore, π is locally optimal with respect to the neighborhood $\mathscr{N}_1$, if and only if either $p_{\pi_i} \leqslant p_{\pi_{i+1}}$ or $\pi_i \to \pi_{i+1}$ holds for $i = 1, \ldots, n-1$. Fig. 3 shows the structure of a locally optimal solution π.

Given a solution $\pi \in \mathscr{S}_1$, a corresponding locally optimal solution can be calculated by a procedure *localopt*(π) which is similar to the "bubble sort" algorithm. In each step some job will be "bubbled" onto a position where it fulfills the condition of a locally optimal solution. More precisely, in iteration i the job π_i will be shifted to the left until π_i and its predecessor fulfills the condition of a locally optimal solution for the first time. All interchanges of jobs in this procedure will lead to a decrease of the mean flow time.

Theorem 3.1. *The procedure localopt*(π) *calculates a locally optimal schedule in* $O(n^2)$ *time.*

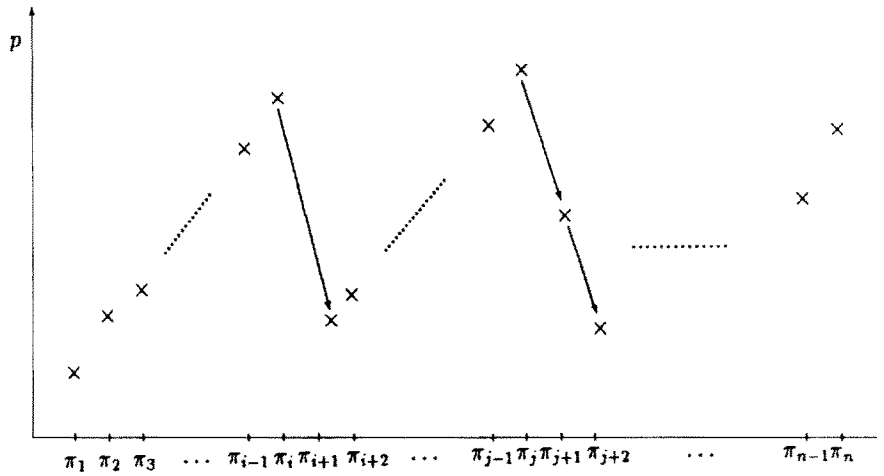

Fig. 3. Locally optimal solution.

Proof. By complete induction one can prove that after iteration i the jobs $\pi_1, \ldots, \pi_i$ fulfill the condition of a locally optimal schedule. □

Next we define a secondary neighborhood $\mathcal{N}_2$ on the set $\mathcal{S}_2$ of all locally optimal solutions. The operators that define $\mathcal{N}_2$ consist of two parts. First a given solution is perturbed by shifting a job to the left or to the right. Afterwards the procdedure *localopt*(π) is used to calculate a locally optimal solution corresponding to the perturbed solution.

There are two types of shift operators that perturb a solution. The operators *left*(i) will shift the job from position i to a position $j \leqslant i$ ($i = 2, \ldots, n$) and the operators *right*(i) will shift the job from position i to a position $j \geqslant i$ ($i = 1, \ldots, n-1$). In both cases it may happen that not only one job in position i but also some of its precedence predecessors or precedence successors are shifted. We will try to define these perturbation operators in such a way that *localopt* will not reverse the interchanges of *left*(i) and *right*(i). Since the operators *left*(i) and *right*(i) will be compositions of interchange operators, *localopt* will not reverse these operators if one of the underlying interchange operators leads to a decrease of the mean flow time.

For a given solution $\pi \in \mathcal{S}_2$, the operator *left*(i) will iteratively consider the jobs $\pi_{i-1}, \pi_{i-2}, \ldots$. In step k it will shift job π_{i-k} immediately after the job π_i if this is possible, i.e. if job π_{i-k} is not a (not necessarily immediate) precedence predecessor of job π_i. Otherwise the current sequence will not change. In this case the job π_{i-k} will be shifted together with job π_i to the left in the next iterations. Since each precedence predecessor of π_{i-k} is also a precedence predecessor of π_i, each shift in one of the next iterations will stay feasible.

The above iterative procedure will stop when π_{i-k} is shifted immediately after π_i and its processing time is at least the processing time of π_i, or, otherwise, when the

last π_{i-k} considered was π_1. In the first case, *localopt* will not reverse the last change of *left*(i), since the interchange of job π_i and its actual successor would not lead to a decrease of the mean flow time. In the second case, *localopt* will reverse all interchanges made by *left*(i), if and only if all underlying interchange operators of all shifts executed by *left*(i) have led to an increase of the mean flow time.

The operator *right*(i) is defined in a symmetric way. It will iteratively consider the jobs $\pi_{i+1}, \pi_{i+2}, \ldots$, and shift them immediately before job π_i if this is possible. It stops when π_{i+k} is shifted immediately before π_i and its processing time is at most the processing time of π_i, or, otherwise, when the last π_{i+k} considered was π_n. Again, in the first case, *localopt* will not reverse the interchanges made by *right*(i). In the second case the situation is a bit different from the situation for the operator *left*(i), since the *localopt* builds up a locally optimal solution from left to right. *Localopt* first will try to interchange the job π_{i-1} with its new successor (the successor after applying *right*(i)) since the jobs π_{k-1} and π_k, $k = 2, \ldots, i-1$, fulfill the local optimality condition. If this interchange is possible (i.e. no precedence constraint exists between the jobs) and leads to a decrease of the mean flow time, *localopt* will interchange these two jobs and therefore not reverse *right*(i). Otherwise, *localopt* will reverse all interchanges made by *right*(i), if and only if all underlying interchange operators of all shifts executed by *right*(i) have led to an increase of the mean flow time.

Now the set of operators OP_2 for the secondary neighborhood $\mathcal{N}_2$ is given by

$$OP_2 = \{localopt \circ left(i) \mid i = 2, \ldots, n\} \cup \{localopt \circ right(i) \mid i = 1, \ldots, n-1\}.$$

For each operator from OP_2 we define

$$\mathcal{S}_2^{localopt \cdot left(i)} = \{\pi \in \mathcal{S}_2 \mid localopt \circ left(i)(\pi) \neq \pi\}$$

and

$$\mathcal{S}_2^{localopt \cdot right(i)} = \{\pi \in \mathcal{S}_2 \mid localopt \circ right(i)(\pi) \neq \pi\}.$$

As stated above it is easy to check whether or not an operator *left*(i) or *right*(i) will be reversed by *localopt*.

The following lemma follows immediately from the definition of the operators *localopt* and *right*(i).

Lemma 3.1. *For a solution π, let $p_{\pi_i} \leqslant p_{\pi_j}$ for some $i < j$.*

1. *In localopt(π) the jobs π_i and π_j are in the same order as in π.*
2. *If in right(k) (π) the jobs π_i and π_j are in an opposite order as in π we must have: π_k is a (not necessarily immediate) precedence predecessor of π_i or equal to π_i.*

Theorem 3.2. *The neighborhood $\mathcal{N}_2$ on the set $\mathcal{S}_2$ is strongly connected.*

Proof. We show that for arbitrary locally optimal solutions π and π' there exists a sequence of operators that transforms π into π'.

Assume that π'_i is sequenced before π'_{i+1} in the solution π, $i = 1, \ldots, k-1$. We will show that by a sequence of operators we can achieve a solution where this property is true for $i = 1, \ldots, k$.

If π'_{k+1} is sequenced after π'_k in π we are done. Otherwise, let l be the position of π'_{k+1} in π. Note, that in this case no precedence relation between π'_k and π'_{k+1} exists. We apply the operator *right*(l) to π. Since π'_k must have a processing time not greater than π'_{k+1} and since π'_k is sequenced after π'_{k+1} in π this operator interchanges π'_{k+1} with a job which has a smaller or equal processing time and which is sequenced after π'_{k+1}. Afterward we apply the operator *localopt*. This operator will not reverse the interchange of π'_{k+1} with the job with smaller or equal processing time.

We repeat this step until π'_{k+1} is sequenced after π'_k. This situation will be achieved after a finite number of steps since only a finite number of jobs with processing time not greater than π'_{k+1} are sequenced after π'_{k-1}.

During the above procedure we only apply the operator *localopt* and the operator *right* with the job π'_{k+1}. Due to Lemma 3.1 these operators will not change the order of the jobs π'_i and π'_{i+1}, $i = 1, \ldots, k-1$, since π'_{k+1} is not a (not necessarily immediate) precedence predecessor of a job from $\{\pi'_1, \ldots, \pi'_k\}$. Therefore, we achieve a sequence where π'_i is sequenced before π'_{i+1} for $i = 1, \ldots, k$. □

Corollary 3.1. *The secondary neighborhood $\mathcal{N}'_2$ on the set $\mathcal{S}_2$ defined by the operator set $OP'_2 = \{localopt \circ right(i) \mid i = 1, \ldots, n-1\}$ is strongly connected.*

Proof. The proof of Theorem 3.2 uses only operators *localopt* *right*(i). Therefore, it can be applied to prove the corollary. □

Contrary to the first problem, each sequence of $\mathcal{S}_2$ has only $O(n)$ neighbors in $\mathcal{N}'_2$.

We finally note that the same ideas may be applied to problem $1 \mid prec \mid \sum w_i C_i$. In this case we only have to replace the p_i-values by p_i/w_i.

4. The problem $1 \parallel \sum T_i$

In this section we consider the $1 \parallel \sum T_i$ scheduling problem. n jobs have to be processed on a single machine. For each job i, a processing time p_i and a due date d_i are given. The objective is to minimize the total tardiness $\sum T_i$ where

$$T_i = \max\{0, C_i - d_i\}$$

is the tardiness of job i and again C_i denotes the completion time of job i. In this section we assume that the jobs are numbered in such a way that $d_1 \leqslant d_2 \leqslant \cdots \leqslant d_n$.

The complexity of this problem was open for a long time. Recently, Du and Leung have proven that the problem is NP-hard [2]. However, a pseudopolynomial algorithm has been given by Lawler [5].

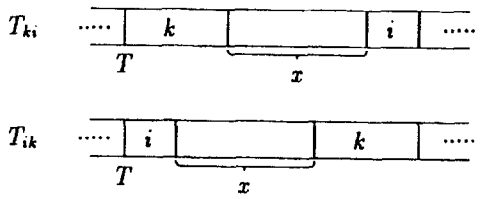

Fig. 4. T_{ki} and T_{ik}.

4.1. Basic properties

In this section we give some basic properties of our problem that are useful for the further considerations. First we investigate the question how special interchanges of jobs affect the objective function value. We assume that $i < k$, which implies $d_i \leqslant d_k$, and define T_{ik} as the contribution of two jobs i and k to the objective function for a sequence where job i starts at time T and between jobs i and k other jobs are processed during a period of x time units, i.e.

$$T_{ik} := \max\{0, T + p_i - d_i\} + \max\{0, T + p_i + x + p_k - d_k\}.$$

We investigate the question under which conditions on T we have $T_{ik} \leqslant T_{ki}$ (see Fig. 4). This inequality means that interchanging jobs i and k in the corresponding sequence does not improve the objective value.

We consider the following cases:

(a) $T + p_i > d_i$, $T + p_k > d_k$,

(b) $T + p_i > d_i$, $T + p_k \leqslant d_k < T + p_k + p_i + x$,

(c) $T + p_i > d_i$, $T + p_k + p_i + x \leqslant d_k$,

(d) $T + p_i \leqslant d_i$.

It is easy to see that $T_{ik} \leqslant T_{ki}$ holds for all T considered in cases (c) and (d). In case (a), $T_{ik} \leqslant T_{ki}$ holds for all considered T if $p_i \leqslant p_k$ and there does not exist any T with this property if $p_i > p_k$. Moreover, in case (b) the condition $T_{ik} \leqslant T_{ki}$ is equivalent to $T \leqslant d_k - p_i$.

The set of feasible solutions of the problem $1 \| \sum T_i$ is given by the set of all $n!$ permutations of the jobs. However, by means of the above considerations we can exclude some sequences from the search for an optimal solution.

Lemma 4.1. *Let i, k be two jobs with $p_i \leqslant p_k$ and $i < k$. Then there exists an optimal solution where job i is processed before job k.*

Proof. Let π be an optimal sequence where the job k is processed before job i and π' is the sequence obtained from π by interchanging jobs i and k. Let T be the starting time of job i in π' and x be the sum of the processing times of the jobs scheduled between i and k in π'. The completion times of the jobs between i and k in π' are not greater than in π since $p_i \leqslant p_k$, and the completion times of the jobs before i and after k in

π are equal in π and π'. Therefore, if in addition the condition $T_{ik} \leqslant T_{ki}$ is satisfied, then π' is not worse than π. For cases (a), (c) and (d) $T_{ik} \leqslant T_{ki}$ trivially holds. For case (b) we have $T \leqslant d_k - p_k \leqslant d_k - p_i$, hence $T_{ik} \leqslant T_{ki}$. □

Note that Lemma 4.1 describes a special case of a dominance criteria derived by Rinnooy Kan et al. for the more general weighted tardiness problem [8]. Hence, we can establish a precedence constraint $i \to k$ between jobs i and k if the condition of Lemma 4.1 is satisfied. Therefore, we need only to consider such sequences that are compatible with all precedence constraints established by Lemma 4.1 because we have an optimal solution of our problem within this set.

In the following, we only consider the case $x = 0$, i.e. $T_{ik} \leqslant T_{ki}$ for some T means that interchanging adjacent jobs i and k in a sequence where the first job i starts at time T does not improve the objective value.

We have alredy seen that $T_{ik} \leqslant T_{ki}$ holds for each T if $i < k$ and $p_i \leqslant p_k$. Now we consider the case $p_i > p_k$.

Lemma 4.2. *Assume that $p_i > p_k$ with $i < k$. Then $T_{ik} \leqslant T_{ki}$ if $T \leqslant d_k - p_i$ and $T_{ki} < T_{ik}$ otherwise.*

Proof. Let $T \leqslant d_k - p_i$. Then case (a) cannot occur, because this would yield $d_k < T + p_k < T + p_i \leqslant d_k$. For cases (b)–(d) $T_{ik} \leqslant T_{ki}$ trivially holds.

Let $T > d_k - p_i$. Then we have

$$0 \leqslant d_k - d_i < T + p_i - d_i \leqslant T + p_k + p_i - d_i,$$

$$0 < T + p_i - d_k \leqslant T + p_i + p_k - d_k,$$

and

$$\max\{0, T + p_k - d_k\} < T + p_i - d_k.$$

Therefore,

$$T_{ki} = \max\{0, T + p_k - d_k\} + T + p_k + p_i - d_i$$

$$< T + p_i - d_i + T + p_i + p_k - d_k = T_{ik}. \quad \square$$

Next, for each T we define a linear order relation $(i, j)_T$ that is compatible with $i \to j$. Furthermore, if in a sequence job i starting at time T is an immediate predecessor of job j, then $(i,j)_T$ holds if $T_{ij} < T_{ji}$. Moreover, this linear order relation may be used to break ties in the case $T_{ij} = T_{ji}$.

Definition 4.1. Let i, k be two jobs with $i < k$. Then we define

$$(i,k)_T \quad \text{if and only if} \quad p_i \leqslant p_k \text{ or } T \leqslant d_k - p_i$$

and

$(k,i)_T$ if and only if $p_i > p_k$ and $T > d_k - p_i$.

Lemma 4.3. *For each T the relation $(i,j)_T$ is transitive, i.e. $(i,j)_T$ and $(j,k)_T$ implies $(i,k)_T$.*

Proof. Let $i < j < k$. Then we have to show

(a) $(i,j)_T$ and $(j,k)_T$ implies $(i,k)_T$ and

(b) $(k,j)_T$ and $(j,i)_T$ implies $(k,i)_T$.

Case (a):

$(i,j)_T$ and $(j,k)_T$

$\Leftrightarrow$ $(p_i \leqslant p_j$ or $T \leqslant d_j - p_i)$ and $(p_i \leqslant p_k$ or $T \leqslant d_k - p_j)$

$\Rightarrow$ $(p_i \leqslant p_j$ and $p_j \leqslant p_k)$ or $(p_i \leqslant p_j$ and $T \leqslant d_k - p_j)$ or $(T \leqslant d_j - p_i)$

$\Rightarrow$ $(p_i \leqslant p_j$ and $p_j \leqslant p_k)$ or $(p_i \leqslant p_j$ and $T \leqslant d_k - p_j)$ or $(T \leqslant d_j - p_i)$

$\Rightarrow$ $(p_i \leqslant p_k)$ or $(T \leqslant d_k - p_j)$ or $(T \leqslant d_k - p_i)$

$\Leftrightarrow$ $(i,k)_T$.

Case (b):

$(k,j)_T$ and $(j,i)_T$

$\Leftrightarrow$ $(p_j > p_k)$ and $(T > d_k - p_j)$ and $(p_i > p_j)$ and $(T > d_j - p_i)$

$\Rightarrow$ $(p_j > p_k)$ and $(T > d_k - p_j)$ and $(p_i > p_j)$

$\Rightarrow$ $(p_i > p_k)$ and $(T > d_k - p_i)$

$\Leftrightarrow$ $(k,i)_T$. □

Corollary 4.1. *For each T there exists a unique sequence $\varrho^T = (\varrho_1, \ldots, \varrho_n)$ such that $i < k$ implies $(\varrho_i, \varrho_k)_T$.*

Proof. Consider the directed graph G with vertices $1, \ldots, n$ and arcs (i,j) if and only if $(i,j)_T$ holds. For arbitrary $i \neq j$ we either have $(i,j)_T$ or $(j,i)_T$. Furthermore, by transitivity of the relation $(i,j)_T$, this graph has no cycles. Thus, there exists a unique topological numeration ϱ of the vertices of G. □

Let us observe that if we define $T_{\min} := \min\{d_k - p_i \mid i < k, p_i > p_k\}$ and $T_{\max} := \max\{d_k - p_i \mid i < k, p_i > p_k\}$, then for $T \leqslant T_{\min}$ the sequence ϱ of Corollary 4.1 corresponds to the earliest due date sequence $\varrho^{\mathrm{EDD}} = (1, \ldots, n)$ and for $T > T_{\max}$ the sequence ϱ corresponds to the shortest processing time sequence ϱ^{SPT} if all p_i are different.

4.2. The neighborhoods for the $1 \| \sum T_i$ problem

For the problem $1 \| \sum T_i$ a neighborhood is defined by the set $\mathscr{S}_1$ of all sequences $\pi = (\pi_1, \ldots, \pi_n)$ that are compatible with the precedence constraints according to Lemma 4.1 and by the operators $api(i)$ $(i = 1, \ldots, n-1)$. The operator $api(i)$ interchanges the adjacent jobs that are scheduled in position i and $i+1$, i.e. the sequence $\pi' = api(i)(\pi)$ is defined by

$$\pi'_k = \begin{cases} \pi_{i+1} & \text{if } k = i, \\ \pi_i & \text{if } k = i+1, \\ \pi_k & \text{otherwise,} \end{cases}$$

again. This operator maps a feasible sequence $\pi \in \mathscr{S}_1$ into a feasible sequence if and only if there does not exist a precedence constraint $\pi_i \to \pi_{i+1}$. Therefore, the sets $\mathscr{S}_1^{api(i)}$ and also the primary neighborhood $\mathscr{N}_1$ are defined as in Section 3.

In the following we assume that all jobs have different processing times and due dates (if this is not the case, it can be obtained by a simple perturbation of the data of the problem, that does not change the optimal solution).

To define the secondary neighborhood, we define $\mathscr{S}_2$ as the set of all feasible sequences $\pi \in \mathscr{S}_1$ for which $(\pi_i, \pi_{i+1})_{S(i)}$ holds for $i = 1, \ldots, n-1$, where $S(i)$ is the starting time of job π_i on position i in π. Clearly, all sequences in $\mathscr{S}_2$ are locally optimal with respect to $\mathscr{N}_1$. But not all locally optimal solutions belong to $\mathscr{S}_2$. However, each locally optimal sequence is represented by at least one sequence in $\mathscr{S}_2$ with the same objective value. Thus, we may restrict to $\mathscr{S}_2$.

The definition of the operator set OP_2 on $\mathscr{S}_2$ is more complicated than for the two problems considered before.

We call a subsequence $\pi^F = (\pi_1^F, \ldots, \pi_k^F)$, $k \leqslant n$, a *final sequence* if the following three conditions are satisfied when the jobs of π^F are processed at the end of a complete sequence, i.e. the job π_1^F starts at time $\sum_{j=1}^{n} p_j - \sum_{j=1}^{k} p_{\pi_j^F}$:

- among the jobs contained in π^F no precedence constraint is violated;
- π^F contains no job that is a predecessor of a job not contained in π^F;
- the jobs π_i^F and π_{i+1}^F fulfill the condition of local optimality when the job π_i^F starts at time $\sum_{j=1}^{n} p_j - \sum_{j=i}^{k} p_{\pi_j^F}$, for $i = 1, \ldots, k-1$.

Starting with a sequence $\pi = (\pi_1, \ldots, \pi_n) \in \mathscr{S}_2$, for each $j = 1, \ldots, n$ we define

$$cut(j)(\pi) = (\pi_{j+1}, \ldots, \pi_n),$$

i.e. $cut(j)$ cuts π after position j and leaves the final sequence $(\pi_{j+1}, \ldots, \pi_n)$.

Next, to the final sequence $\pi^F = (\pi_{j+1}, \ldots, \pi_n)$ we apply

$$add(k)(\pi^F) = (k, \pi_{j+1}, \ldots, \pi_n),$$

where k does not belong to $\{\pi_j, \ldots, \pi_n\}$. $add(k)$ adds job k at the beginning of π^F. The resulting sequence is a final one if and only if $(k, \pi_{j+1})_S$ holds where $S = \sum_{i=1}^{n} p_i - \sum_{v=j+1}^{n} p_{\pi_v} - p_k$ and k is not a predecessor of any job in $\{\pi_1, \ldots, \pi_j\} \setminus \{k\}$.

If this is the case, we try to extend $(k, \pi_{j+1}, \ldots, \pi_n)$ to a complete sequence that belongs to $\mathscr{S}_2$ by adding the remaining jobs to the left of $(k, \pi_{j+1}, \ldots, \pi_n)$. This is done by an operator *localopt*. *Localopt* is not defined if such an extension does not exist.

Finally, we define the operator set OP_2 by

$$OP_2 = \{op(i,j) = localopt \circ add(i) \circ cut(j) \mid i,j = 1, \ldots, n\}$$

and

$$\mathscr{S}_2^{op(i,j)} = \{\pi \in \mathscr{S}_2 \mid i \in \{\pi_1, \ldots, \pi_{j-1}\}, (i, \pi_{j+1}, \ldots, \pi_n) \text{ is a final sequence,}$$
$$localopt \text{ is defined on } (i, \pi_{j+1}, \ldots, \pi_n)\}.$$

It remains to describe *localopt* in more detail.

The operator *localopt* is defined by a procedure using two operators. The first operator *localopt*1(R) constructs for a given set R of jobs a locally optimal schedule π^R, where the first job of π^R starts at time 0.

If we apply this operator for a given final sequence π^F to the set R of unscheduled jobs, i.e. to the set of jobs not contained in π^F, the concatenation of π^R with the final sequence π^F does not necessarily lead to a locally optimal sequence $\pi' = (\pi^R, \pi^F)$ because the last job of π^R and the first job of π^F may violate the condition of local optimality. In this case we try to extend π^F to a final sequence (π^M, π^F) such that the set R of still unscheduled jobs (jobs not contained in π^M or π^F) can be scheduled with *localopt*1(R) and the concatenation of π^R and (π^M, π^F) leads to a locally optimal sequence. The corresponding operator will be denoted by *localopt*2. If it is not possible to extend π^F to a locally optimal sequence, *localopt*2 will stop with this information. Summarizing, we first apply *localopt*2 to a final sequence π^F and extend π^F to a final sequence (π^M, π^F) (if this is possible). Afterwards we apply *localopt*1 to the set R of still unscheduled jobs and we get a locally optimal sequence $\pi' = (\pi^R, \pi^M, \pi^F)$.

First we give an algorithm for the operator *localopt*1 that generates for a given set R of jobs a sequence $\pi^R = (\pi_1^R, \ldots, \pi_{|R|}^R)$ such that $(\pi_i^R, \pi_{i+1}^R)_{S(i)}$ holds for $i = 1, \ldots, |R| - 1$ where $S(i)$ denotes the starting time of the job π_i^R.

The operator *localopt*1 constructs a schedule for the jobs of the set R from left to right. In each step the completion time T of the current partial schedule is computed and the first job of the corresponding sequence ϱ^T of unscheduled jobs is scheduled next.

It is immediately clear from Corollary 4.1 that the generated sequence $\pi^R = (\pi_1^R, \ldots, \pi_{|R|}^R)$ satisfies the condition $(\pi_i^R, \pi_{i+1}^R)_{S(i)}$ for $i = 1, \ldots, |R| - 1$. Furthermore, if we apply *localopt*1 to all n jobs, we get a locally optimal sequence $\pi \in \mathscr{S}_2$.

Next we described the operator *localopt*2. Let $\pi^F = (\pi_1^F, \ldots, \pi_k^F)$ be a final sequence, R be the set of jobs not contained in π^F and $S = \sum_{i \in R} p_i$ be the starting time of the final sequence. First we determine the job $r \in R$ with maximal due date, i.e. $d_r > d_i$ for all $i \in R \setminus \{r\}$. We distinguish three cases.

Case 1: The job r cannot be processed immediately before π_1^F in order to fulfill local optimality, i.e. $(\pi_1^F, r)_{S-p_r}$ holds.

In this case it is not possible to extend π^F to a locally optimal sequence (for the proof see Lemma 4.8). *localopt*2 will stop with this information.

Case 2: $(r, \pi_1^F)_{S-p_r}$ and $S - p_r \leqslant d_r$.

In this case one can prove that each job $i \in R \setminus \{r\}$ can be concatenated with (r, π^F) to a final sequence (i, r, π^F) (apply Lemma 4.4 with $S^* = S - p_r$). Therefore, in this case *localopt*2 determines $\pi^M = (r)$ and we may apply *localopt*1 to the set $R' = R \setminus \{r\}$ to get a locally optimal sequence $\pi = (\pi^{R'}, \pi^M, \pi^F)$.

Case 3: $(r, \pi_1^F)_{S-p_r}$ and $S - p_r > d_r$.

In this case the operator *localopt*2 constructs a sequence $\pi^M = (\pi_1^M, \ldots, \pi_l^M)$ such that (π^M, π^F) is a final sequence. Furthermore, the job π_1^M starts not later than its due date $d_{\pi_1^M}$ and π_1^M has a larger due date than the jobs that are not contained in π^M and π^F. Since in this case the job π_1^M fulfills the same conditions as the job r in Case 2, the set R' of jobs not contained in π^M and π^F may be scheduled by *localopt*1 in order to get a locally optimal sequence $\pi = (\pi^{R'}, \pi^M, \pi^F)$.

To reach this goal, we first determine a candidate for the last job in π^M. This job must fulfill together with π_1^F the local optimality condition. Furthermore, this job is not allowed to have successors in R with respect to the precedence constraints $\rightarrow$, since we only consider sequences from $\mathscr{S}_1$. Among all jobs with the above conditions, let t be the job with maximal processing time, i.e.

$$p_t = \max\{p_i \mid (i, \pi_1^F)_{S-p_i}, i \in R, p_i \geqslant p_j \text{ or } d_i > d_j \text{ for all } j \in R\}.$$

Such a job t exists since at least the job r fulfills the above conditions. Furthermore, we define by A the set of all jobs in R with a larger processing time than p_t, i.e.

$$A = \{i \in R \mid p_i > p_t\}.$$

Note, that r does not belong to A since r was also a candidate for t. By the definition of A, a job $i \in A$ either dominates a job $j \in A$ (i.e. there exists a precedence constraint $i \rightarrow j$) or it cannot be processed immediately before the final sequence π^F in order to get a locally optimal sequence. The operator *localopt*2 will not schedule the jobs from A.

Since $\mathscr{S}_1$ contains only sequences that are compatible with the precedence constraints $\rightarrow$ also the predecessors of jobs from A will not be scheduled by *localopt*2. Let a be the job with maximal due date in A and let B be the subset of jobs in $R \setminus A$ with a larger due date than a, i.e.

$$B = \{i \in R \setminus A \mid d_i > d_a\}.$$

Since job a belongs to A it has a larger processing time than each job $i \in R \setminus A$. Therefore, for all jobs $i \in R \setminus (A \cup B)$, we now have $d_i < d_a$ and $p_i < p_a$, which implies a precedence constraint $i \rightarrow a$. Since no job from A will be scheduled by the operator *localopt*2 this implies that only jobs from B will be scheduled in π^M by *localopt*2. By the definition of A, we have $p_i > p_t$ for all $i \in A$. Therefore, from the definition of t we must have $d_i < d_t$ for all $i \in A$, which yields $t \in B$.

Let $(\varrho_1, \ldots, \varrho_g) = \varrho_B^{\mathrm{SPT}}$ be the shortest processing time sequence of the jobs in B, i.e. $p_{\varrho_j} < p_{\varrho_{j+1}}$ for $j = 1, \ldots, g-1$ and let S_j be the starting time of job ϱ_j if the jobs

$(\varrho_1, \ldots, \varrho_g)$ are scheduled immediately before the final sequence π^F, i.e. $S_j = S - \sum_{i=j}^{g} p_{\varrho_i}$. Note that $\varrho_g = t$ since t belongs to B.

As mentioned above, the first job of the sequence π^M has to start not later than its due date. Therefore, let ϱ_g be the last job in $(\varrho_1, \ldots, \varrho_g)$ for which $S_q \leqslant d_{\varrho_q}$ holds, i.e. $S_i > d_{\varrho_i}$, for $i = q+1, \ldots, g$. For the sequence $(\varrho_{q+1}, \ldots, \varrho_g)$ with starting times $S_{q+1}, \ldots, S_g$, one can prove that the local optimality conditions are fulfilled (see Lemma 4.6).

If no job ϱ_q with $S_q \leqslant d_{\varrho_q}$ exists, there are two possible cases. First, if $A = \emptyset$ then the sequence $(\varrho_1, \ldots, \varrho_g, \pi^F)$ is a complete sequence of all jobs. Due to Lemma 4.6, the sequence $(\varrho_1, \ldots, \varrho_g)$ fulfills the local optimality conditions. Furthermore, since $\varrho_g = t$ the jobs ϱ_g and π_1^F fulfill the local optimality condition. Therefore, in this case $(\varrho_1, \ldots, \varrho_g, \pi^F)$ is locally optimal. If $A \neq \emptyset$ we will prove that it is not possible to complete the final sequence π^F to a locally optimal sequence (see Lemma 4.9). *localopt2* will stop with this information.

The second condition for the first job in the sequence π^M is, that this job must have a greater due date than the jobs not contained in π^F and π^M. Therefore, let ϱ_z be the job with maximal due date in the set $\{\varrho_1, \ldots, \varrho_q\}$. We calculate the first position f, $q \leqslant f \leqslant g$ (for the case $z = q$: $q+1 \leqslant f \leqslant g$) such that

$$S_f - p_{\varrho_z} \leqslant d_{\varrho_z} \quad \text{and} \quad S_{f+1} - p_{\varrho_z} > d_{\varrho_z}, \tag{4.1}$$

where S_{g+1} is defined to be the starting time S of the final sequence π^F.

Job r has the largest due date in R and it does not belong to the set A. Therefore, the job r belongs to the set B and we either have $r = \varrho_z$ or $r \in \{\varrho_{q+1}, \ldots, \varrho_g\}$, which implies $p_{\varrho_z} \leqslant p_r$. Because we assume $S - p_r > d_r$ in Case 3, we have

$$S_{g+1} - p_{\varrho_z} = S - p_{\varrho_z} \geqslant S - p_r > d_r \geqslant d_{\varrho_z}.$$

Furthermore, we have $S_q - p_{\varrho_z} < S_q \leqslant d_{\varrho_q} \leqslant d_{\varrho_z}$ for $z \neq q$ and $S_{q+1} - p_{\varrho_z} = S_q \leqslant d_{\varrho_q} \leqslant d_{\varrho_z}$ for $z = q$, respectively. Therefore, a position f that fulfills (4.1) always exists.

It is possible to prove that $(\varrho_z, \varrho_f, \ldots, \varrho_g, \pi^F)$ is a final sequence (see Lemma 4.7). Furthermore, for $j \in q+1, \ldots, f-1$ we have $S_j > d_{\varrho_j}$ which implies

$$d_{\varrho_j} < S_j + p_{\varrho_j} - p_{\varrho_z} = S_{j+1} - p_{\varrho_z} \leqslant d_{\varrho_z},$$

where the last inequality follows from the definition of f. Furthermore, by definition of ϱ_z, $d_{\varrho_z} > d_{\varrho_j}$ for $j \in \{1, \ldots, q\} \setminus \{z\}$, and $d_{\varrho_z} > d_a \geqslant d_i$ for $i \in R \setminus B$, since $\varrho_z \in B$. Therefore, ϱ_z has a greater due date than all jobs not contained in $(\varrho_z, \varrho_f, \ldots, \varrho_g, \pi^F)$. If we now schedule the jobs $(\varrho_z, \varrho_f, \ldots, \varrho_g)$ immediately before the final sequence π^F, the job ϱ_z starts before its due date. Therefore, *localopt2* determines $\pi^M = (\varrho_z, \varrho_f, \ldots, \varrho_g)$. As stated above, *localopt1* will sequence the set R' of still unscheduled jobs in such a way that $\pi = (\pi^{R'}, \pi^M, \pi^F)$ is a locally optimal sequence.

The above considerations can be summarized in the following procedure.

1. $S := \sum_{j \in R} p_j$;
2. Let r be the job with maximal due date in R;

3. **if** $(\pi_1^F\ r)_{S-p_r}$ **then stop** {no solution exists}
4. **else if** $S - p_r \leqslant d_r$ **then**
5. $\pi^M := (r)$

else begin

6. Let t be the job with
$p_t = \max\{p_i | (i, \pi_1^F)_{S-p_t},\ i \in R,\ p_i \geqslant p_j$ or $d_i > d_j$ for all $j \in R\}$;
7. $A := \{i \in R | p_i > p_t\}$;
8. Let a be the job in A with maximal due date;
9. **if** $A \neq \emptyset$ **then** $d := d_a$ **else** $d := -\infty$;
10. $B := \{i \in R \backslash A | d_i > d\}$;
11. Let $(\varrho_1, \ldots, \varrho_g)$ be the shortest processing time sequence of the jobs in B;
12. $q := g$;
13. **while** $S - p_{\varrho_q} > d_{\varrho_q}$ and $q \geqslant 1$ **do**
begin
14. $S := S - p_{\varrho_q}$;
15. $q := q - 1$
end;
16. **if** $q = 0$ **then**
17. **if** $A = \emptyset$ **then**
18. $\pi^M = (\varrho_1, \ldots, \varrho_g)$
19. **else stop** {no solution exists}

else begin

20. Let ϱ_z be the job with maximal due date in $\{\varrho_1, \ldots, \varrho_q\}$;
21. **if** $z = q$ **then**
begin
22. $S := S + p_{\varrho_q}$;
23. $q := q + 1$
end;
24. $f := q$;
25. **while** $S - p_{\varrho_z} \leqslant d_{\varrho_z}$ **do**
begin
26. $S := S + p_{\varrho_f}$;
27. $f := f + 1$;
end
28. $\pi^M := (\varrho_z, \varrho_f, \ldots, \varrho_g)$
end

end

To illustrate the above algorithm, we consider the following example:

i	1	2	3	4	5	6
p_i	30	25	8	20	11	6
d_i	44	48	52	56	60	72

We get the precedence constraints $3 \to 4$ and $3 \to 5$. Moreover, we obtain the following sequences in S_2: $\pi^1 = (1,3,4,5,6,2)$, $\pi^2 = (1,3,2,6,5,4)$, $\pi^3 = (2,3,4,5,6,1)$, $\pi^4 = (2,3,1,6,5,4)$.

Furthermore, we have

$$\mathcal{N}_2(\pi^1) = \{\pi^2, \pi^3\}, \mathcal{N}_2(\pi^2) = \{\pi^1, \pi^3, \pi^4\},$$

$$\mathcal{N}_2(\pi^3) = \{\pi^1, \pi^2\}, \text{ and } \mathcal{N}_2(\pi^4) = \{\pi^1, \pi^2, \pi^3\}.$$

Note that we have $|\mathcal{S}_1| = 240$ but only $|\mathcal{S}_2| = 4$ and the corresponding graph of the secondary neighborhood has the diameter 2.

To illustrate the three cases mentioned above, we first consider

$$add(5) \circ cut(5)(\pi^1) = (5,2),$$

with $R = \{1,3,4,6\}$. Because of $d_6 = \max\{d_i \mid i \in R\} = 72$ and $S = 64$ we obtain $(5,6)_{58}$. Hence we have Case 1, i.e. π^F cannot be completed to a locally optimal sequence.

Next we consider

$$add(6) \circ cut(5)(\pi^3) = (6,1).$$

with $R = \{2,3,4,5\}$. Because of $d_5 = \max\{d_i \mid i \in R\} = 60$ and $S = 64$, we obtain $S - p_5 = 53 \leqslant 60$ and $(5,6)_{53}$, i.e. we have Case 2 and $\pi^M = (5)$. Applying *localopt*1 yields π^3.

Finally, we consider

$$add(4) \circ cut(6)(\pi^1) = (4),$$

with $R = \{1,2,3,5,6\}$. Because of $d_6 = \max\{d_i \mid i \in R\} = 72$ and $S = 80$, we obtain $S - p_6 = 74 > 72$ and $(6,4)_{74}$, i.e. we have Case 3. In *localopt*2 we determine $\pi^M = (6,5)$, which yields $(\pi^M, \pi^F) = (6,5,4)$. Since job 6 now starts not later than its due date we can apply *localopt*1 , which yields π^2.

To illustrate that also in Case 3 we may establish that it is not possible to extend the given final sequence, consider

$$add(5) \circ cut(6)(\pi^1) = (5),$$

with $R = \{1,2,3,4,6\}$. Because of $d_6 = \max\{d_i \mid i \in R\} = 72$ and $S = 89$, we obtain $S - p_6 = 83 > 72$ and $(6,5)_{83}$, i.e. we have Case 3 again. However, we cannot include further jobs into π^M by *localopt*2 (note that the precedence constraint $3 \to 4$ holds), i.e. the above final sequence cannot be extended to a complete sequence in $\mathcal{S}_2$.

We still have to prove that our algorithm works correct. First we will show that if a job starts before its due date then each job with a smaller due date may be scheduled immediately before this job without violating the conditions of local optimality.

Lemma 4.4. *Let R be a set of jobs, r be a job with $d_r > d_i$ for all $i \in R \setminus \{r\}$ and $S^* \leqslant d_r$ be the starting time for job r. Then*

$$(i,r)_{S^*-p_i} \quad \text{for all } i \in R \setminus \{r\}.$$

Proof. Let $i \in R$ be given. According to Definition 4.1 we have $(i,r)_T$ for all T if $p_r > p_i$ and for all $T \leqslant d_r - p_i$ if $p_r < p_i$. Since $S^* \leqslant d_r$ we have $S^* - p_i \leqslant d_r - p_i$, which proves the lemma. □

Since the job r in Step 5 and the job ϱ_z in Step 28 of the procedure *localopt2* together with the set R of still unscheduled jobs fulfill the conditions of Lemma 4.4, *localopt1* may be used to complete the final sequence to a locally optimal sequence.

Furthermore, we have to prove that the sequence $\pi^M = (\varrho_1, \ldots, \varrho_g)$ defined in Step 18 and the sequence $\pi^M = (\varrho_z, \varrho_i, \ldots, \varrho_g)$ defined in Step 28 of the procedure *localopt2* together with π^F defines a final sequence (π^M, π^F). We first prove:

Lemma 4.5. *Let $(\varrho_1, \ldots, \varrho_g)$ be the shortest processing time sequence of the jobs in B as defined in Step 11 of the procedure localopt2. For $1 \leqslant i < j \leqslant g$ we have*

$$(\varrho_i, \varrho_j)_T \Leftrightarrow [d_{\varrho_i} > d_{\varrho_j} \Rightarrow T > d_{\varrho_i} - p_{\varrho_j}].$$

Proof.

$$\begin{aligned}
&(\varrho_i, \varrho_j)_T \\
&\quad \Leftrightarrow [d_{\varrho_i} < d_{\varrho_j} \text{ and } (p_{\varrho_i} \leqslant p_{\varrho_j} \text{ or } T \leqslant d_{\varrho_j} - p_{\varrho_i})] \\
&\qquad \text{or } [d_{\varrho_i} > d_{\varrho_j} \text{ and } p_{\varrho_i} \leqslant p_{\varrho_j} \text{ and } T > d_{\varrho_i} - p_{\varrho_i}] \\
&\quad \Leftrightarrow [d_{\varrho_i} < d_{\varrho_j}] \text{ or } [d_{\varrho_i} > d_{\varrho_j} \text{ and } T > d_{\varrho_i} - p_{\varrho_i}] \\
&\quad \Leftrightarrow [d_{\varrho_i} > d_{\varrho_j} \Rightarrow T > d_{\varrho_i} - p_{\varrho_j}].
\end{aligned}$$

The first equivalence holds by definition, while the second holds since $p_{\varrho_i} < p_{\varrho_j}$ by definition of ϱ. □

Lemma 4.6. *Let $(\varrho_1, \ldots, \varrho_g)$ be the shortest processing time sequence of the jobs in B as defined in Step 11 of the procedure localopt2. Let $1 \leqslant i < j \leqslant g$. If $S > d_{\varrho_i}$; then $(\varrho_i, \varrho_j)_S$ holds.*

Proof. Since $S > d_{\varrho_i} \geqslant d_{\varrho_i} - p_{\varrho_j}$ we have $(\varrho_i, \varrho_j)_S$ by Lemma 4.5. □

This lemma proves that $\pi^M = (\varrho_1, \ldots, \varrho_g)$ defined in Step 18 together with π^F defines a locally optimal sequence (π^M, π^F) since $\varrho_g = t$ and π_1^F fulfill the local optimality condition.

Lemma 4.7. *Let $\pi^M = (p_z, p_f, \ldots, p_g)$ be the sequence defined in Step 28 of the procedure localopt2. Then (π^M, π^F) is a final sequence.*

Proof. Let S be the starting time of job π_1^F and $S_j = S - \sum_{l=j}^{g} p_{\varrho_l}$ be the starting time of job ϱ_j, $j = f, \ldots, g$, in the final sequence (π^M, π^F).

(i) By the definition of ϱ_z, we have $p_{\varrho_z} < p_{\varrho_f}$. Since by definition of f we have $S_f - p_{\varrho_z} = S_{f+1} - p_{\varrho_z} - p_{\varrho_f} > d_{\varrho_z} - p_{\varrho_f}$, by Lemma 4.5 $(\varrho_z, \varrho_f)_{S_f - p_{\varrho_z}}$ holds.

(ii) If $S_f > d_{\varrho_f}$, then Lemma 4.6 has proved $(\varrho_f, \varrho_{f+1})_{S_f}$. Otherwise, if $S_f \leqslant d_{\varrho_f}$ (i.e. $f = q$ and $q \neq z$), we have $d_{\varrho_z} > d_{\varrho_f} = d_{\varrho_q}$ by definition of ϱ_z, which implies that

$$S_f = S_{f+1} - p_{\varrho_f} > d_{\varrho_z} + p_{\varrho_z} - p_{\varrho_f} > d_{\varrho_z} - p_{\varrho_f} > d_{\varrho_f} - p_{\varrho_{f+1}}.$$

Therefore, by Lemma 4.5 $(\varrho_f, \varrho_{f+1})_{S_f}$ holds.

(iii) Let $l \in \{f+1, \ldots, g-1\}$. According to Lemma 4.6 we have $(\varrho_l, \varrho_{l+1})_{S_l}$.

(iv) By the definition of t, A and B in Steps 6, 7 and 10 of the procedure *localopt2*, we have $t \in B$, which means $\varrho_g = t$. This yields $(\varrho_g, \pi_1^F)_{S_q}$.

Due to (i)–(iv), the sequence (π^M, π^F) is a final sequence. □

Next we will prove that if the job with maximal due date in the set of unscheduled jobs does not fulfill the local optimality condition with the first job of a given final sequence, then this final sequence cannot be extended to a locally optimal sequence.

Lemma 4.8. *Let π^F be a final sequence, R be the set of jobs not contained in π^F, r be the job with maximal due date in R and S be the starting time of the final sequence π^F. If $(\pi_1^F, r)_{S-p_r}$ holds, then π^F cannot be extended to a locally optimal sequence.*

Proof. Assume that a locally optimal sequence $\pi' \in \mathscr{S}_2$ with the final sequence π^F exists. Let job l be the predecessor of π_1^F in π'. We distinguish two cases.

Case 1: $d_r < d_{\pi_1^F}$.

Due to Definition 4.1 we have

$$p_r > p_{\pi_1^F} \text{ and } S - p_r > d_{\pi_1^F} - p_r, \text{ i.e. } S > d_{\pi_1^F}$$

since $(\pi_1^F, r)_{S-p_r}$ holds. By the definition of r, we have $d_r > d_l$. Therefore, we now have

$$d_l < d_{\pi_1^F}, S - p_l > d_{\pi_1^F} - p_l \text{ and } (l, \pi_1^F)_{S-p_l}.$$

By Definition 4.1 this is only true if $p_l < p_{\pi_1^F}$. Therefore, for the jobs r and l we have

$$d_l < d_r \text{ and } p_l < p_r,$$

which yields a precedence constraint $l \to r$. This contradicts $\pi' \in \mathscr{S}_1$ since in π' the job r is sequenced before job l.

Case 2: $d_r > d_{\pi_1^F}$.

Since r is scheduled before π_1^F in π', we have no precedence constraint $\pi_1^F \to r$. Therefore, $p_r < p_{\pi_1^F}$ holds. Furthermore, because $(\pi_1^F, r)_{S-p_r}$ holds, we have

$$S - p_r \leqslant d_r - p_{\pi_1^F}.$$

Let $w \neq \pi_1^F$ be the successor of r in π'. Therefore, for the starting time S_r of r in π' we have

$$S_r \leqslant S - p_r - p_w \leqslant d_r - p_{\pi_1^F} - p_w \leqslant d_r - p_w.$$

By the definition of r we have $d_r > d_w$. Therefore, $(w, r)_S$ holds, which contradicts the local optimality of π'. □

Finally, we have to prove that if the procedure *localopt2* stops in Step 18, then the final sequence π^F cannot extend to a local optimal sequence.

Lemma 4.9. *Let $A \neq \emptyset$ and let $(\varrho_1, \ldots, \varrho_g)$ be the shortest processing time sequence of the jobs in the set B (B defined as in the procedure localopt2). Furthermore, let $S_j = S - \sum_{i=j}^{g} p_{\varrho_i}$ denote the starting time of job ϱ_j if the sequence $(\varrho_1, \ldots, \varrho_g)$ is scheduled immediately before the final sequence π^F. If $S_i > d_{\varrho_i}$ holds for $i = 1, \ldots, g$, then the final sequence π^F cannot be extended to a locally optimal sequence.*

Proof. Assume that a locally optimal sequence π' with the final sequence π^F exists. Let v be the job from the set A that is sequenced last in π' and let w be the successor of v in π' (v exists since $A \neq \emptyset$).

For all jobs $i \in R \setminus (A \cup B)$, we have a precedence constraint $i \to a$ where a is the job with the maximal due date in A because $i \notin A$ and $a \in A$ implies $p_i < p_t < p_a$. Therefore, all jobs from $R \setminus (A \cup B)$ must be scheduled before v. Furthermore, by the definition of the set A we have $(\pi_1^F, v)_{S-p_v}$, which implies $w \neq \pi_1^F$. Therefore, w must be a job from B, i.e. we have $w = \varrho_l$ for some $l \in \{1, \ldots, g\}$.

Let k be the smallest index such that $d_{\varrho_k} \leqslant d_w$ (ϱ_k may be equal to w). Since $w = \varrho_l$ we have $k \leqslant l$, which implies $p_{\varrho_k} \leqslant p_w$. By the definition of k we have

$$d_{\varrho_i} < d_w \leqslant d_{\varrho_k} \text{ and } p_{\varrho_i} < p_{\varrho_k} \leqslant p_w$$

for all jobs ϱ_i, $i \in \{1, \ldots, k-1\}$. This implies precedence constraints $\varrho_i \to \varrho_k$ and $\varrho_i \to w$. Therefore, the jobs $\varrho_1, \ldots, \varrho_{k-1}$ have to be scheduled before the job w and, consequently, also before v in π'. For the starting time S' of v in π', we now have

$$S' \geqslant S_k - p_v.$$

By the definition of the sets A and B, we have for the jobs w and v

$$d_v < d_w \text{ and } p_v > p_w.$$

Furthermore, we have

$$S' \geqslant S_k - p_v > d_{\varrho_k} - p_v \geqslant d_w - p_v.$$

This implies $(w, v)_{S'}$, which contradicts the local optimality of the sequence π'. □

The results of the previous lemmas can be summaized in the following theorem.

Theorem 4.1. *The operator localopt(π^F) extends a given final sequence π^F in $O(n^2)$ time to a locally optimal sequence if such a sequence exists.*

Finally, we can easily prove the connectivity of the neighborhood.

Theorem 4.2. *The neighborhood $\mathcal{N}_2$ defined on $\mathcal{S}_2$ is strongly connected.*

Proof. Let $\pi^1, \pi^2 \in \mathcal{S}_2$ with $\pi^1 \neq \pi^2$. Moreover, let j denote the maximal position with $\pi_j^1 \neq \pi_j^2$. Let $k = \pi_j^2$ and $i < j$ be the position of k in π^1. Then we apply the operator $op(i,j)(\pi^1)$. Because there exists a sequence $\pi^2 \in \mathcal{S}_2$ with the final sequence $(k, \pi_{j+1}^1, \ldots, \pi_n^1)$, algorithm *localopt* determines a sequence π' with $\pi'_u = \pi_u^2$ for $u = j, j+1, \ldots, n$, i.e. $op(i,j)$ is defined. Hence, after at most $n-1$ steps π^1 is transformed into $\pi^2 \in \mathcal{S}_2$, i.e. $\mathcal{N}_2$ is strongly connected. □

5. Computational results

We have tested both the primary and secondary neighborhood in connection with simulated annealing. The algorithms have been coded in C and run on a SPARC station 10/20. We decided to use simulated annealing for our tests for the following reasons. For iterative improvement it is clear that the secondary neighborhood leads in general to better results, since for the primary neighborhood we stop at the first local optimum, whereas for the secondary neighborhood we may get to this solution in one step. Tabu search also prefers in each iteration the best non-tabu solution in the neighborhood of the actual solution. Therefore, it can be expected that the secondary neighborhood leads to better results than the primary neighborhood for tabu search. However, for simulated annealing such an a priori argument is not valid due to the randomized character of this method.

The control parameters for simulated annealing have been chosen in a standard way (see [12]). In order to get a fair comparison between the primary and secondary neighborhood we have fixed the parameters for a given instance in such a way that for both neighborhoods the computational times were approximately the same. Since the calculation of a neighbor in the secondary neighborhood is more time consuming than for the primary neighborhood this results in a large number of iterations of the simulated annealing algorithm for the primary neighborhood. We first have applied simulated annealing with respect to the secondary neighborhood to an instance with a fixed number of iterations (20 and 100). Afterwards we have fixed the number of iterations for the primary neighborhood in such a way that simulated annealing uses approximately the same amount of time as for the secondary neighborhood. For all the problems considered in Sections 2–4, we have generated problems with 10, 20, 50, 100, 500 and 1000 jobs. The computational times for 100 iterations with respect to the secondary neighborhood for problems with 1000 jobs were within 1 min for the problem $P2 \| C_{\max}$ and within 3 min for the other two problems.

In order to get instances for the problem $P2 \| C_{\max}$ where the difference between the trivial lower bound and a greedy solution is not too small we have chosen the processing times of the jobs randomly from the interval [10 000, 20 000]. The resulting solutions for the two neighborhoods have almost the same quality. The differences between the numbers of iterations for the primary and the secondary neighborhood

Table 1
The $(1/n)\sum_{i=1}^{n} C_i$ values for the $1|prec|\sum C_i$ problem

		Value 1		Value 2	
n	Density (%)	M_1	M_2	M_1	M_2
10	10	774.7	510.7	548.2	510.7
	50	706.0	706.0	706.0	706.0
10	10	592.9	489.2	501.9	489.2
	50	837.5	840.1	848.8	812.4
20	10	1536.1	1221.3	1251.5	1189.1
	50	1371.6	1344.1	1325.8	1310.1
20	10	1331.1	1171.0	1192.3	1165.9
	50	1942.6	1908.9	1940.7	1907.2
50	10	5862.4	5571.4	5900.8	4827.5
	50	5557.0	5163.4	5316.0	5035.0
50	10	5846.6	4925.2	5485.1	4707.4
	50	5177.3	4971.7	5079.7	4842.1
100	10	11425.3	10404.1	11336.8	9899.0
	50	11114.0	10587.4	11015.4	10402.3
100	10	11261.6	10214.2	11140.4	9824.1
	50	11413.2	10877.5	11322.1	10567.2
500	10	60199.8	58466.8	60204.2	58535.6
	50	61817.5	60679.7	61795.1	60209.2
500	10	60345.2	58448.3	60319.5	57796.2
	50	60900.5	59760.0	60866.9	58708.1
1000	10	123875.5	122122.0	123861.5	121762.0
	50	126592.4	125454.1	126584.5	125239.0

are very large (up to a factor of 100). Thus, one step in the secondary neighborhood is very time consuming in comparison with one step in the primary neighborhood.

For the problem $1|prec|\sum C_i$ the processing times of the jobs were generated randomly from the interval [1, 500]. For a problem with given job number and given processing times we have generated two instances with different densities of the precedence constraints (we have used the densities 10% and 50% where the constraints resulting from transitivity are included). The initial solution has been generated randomly.

Table 1 summarizes the results for both neighborhoods. The columns "Value 1" contain the mean flow time values $(1/n)\sum_{i=1}^{n} C_i$ for 20 iterations of the secondary neighborhood and an equivalent number of iterations for the primary neighborhood, respectively. The columns "Value 2" contain the mean flow time values $(1/n)$ $\sum_{i=1}^{n} C_i$ for 100 iterations of the secondary neighborhood and an equivalent number of iterations for the primary neighborhood, respectively. Each entry gives the average value of two runs.

In all but one cases the secondary neighborhood leads to better results than the primary neighborhood. The differences between the results are significant. Furthermore, in all but one cases the results for the secondary neighborhood with 20

Table 2
The $(1/n)\sum_{i=1}^{n} T_i$ values for the $1||\sum T_i$ problem

	Value 1		Value 2	
n	$\mathcal{N}_1$	$\mathcal{N}_2$	$\mathcal{N}_1$	$\mathcal{N}_2$
10	271.2	262.0	267.5	262.0
10	33.8	33.8	33.8	33.8
20	248.0	245.8	245.8	245.8
20	254.6	217.6	208.5	217.6
50	193.6	169.0	171.5	169.0
50	3.5	3.5	3.5	3.5
100	55.4	46.1	47.9	46.1
100	96.8	75.9	87.2	75.5
500	31.8	25.6	31.3	25.6
500	1239.5	730.9	1190.9	729.7
1000	155.2	95.4	149.8	95.4
1000	24.3	18.5	22.8	18.5

iterations are better than the results for the primary neighborhood with a number of iterations which corresponds in time with 100 iterations for the secondary neighborhood. The differences between the number of iterations for the primary and the secondary neighborhood range between a factor of 3 and 5, i.e. the calculation of a neighbor in the secondary neighborhood takes approximately the same time as the calculation of 3 to 5 neighbors in the primary neighborhood.

For the problem $1\|\sum T_i$ the processing times of the jobs were generated randomly from the interval $[1, 1000]$. Furthermore, we have generated the due dates randomly from the interval $[0.2\sum p_i, \sum p_i]$ for the instances with job number $\leqslant 100$ and from the interval $[0.05\sum p_i, \sum p_i]$ for the instances with job number $\geqslant 500$. As initial solution we have chosen the greedy solution calculated by *localopt*1.

Table 2 summarizes the resulting objective function values $(1/n)\sum_{i=1}^{n} T_i$ for both neighborhoods. The meaning of "Value 1" and "Value 2" is the same as in Table 1. Again the entries give the average values of two runs.

As for the problem $1|prec|\sum C_i$ the secondary neighborhood leads to better results, even if the primary neighborhood uses a larger amount of computational time. The differences between the results for the primary and the secondary neighborhood are even larger than for the $1|prec|\sum C_i$ problem (the values for the secondary neighborhood are up to 45% better than the results for the primary neighborhood). The differences between the number of iterations for the primary and secondary neighborhood again range between a factor of 3 and 5.

It can be seen from Table 2 that the results with the secondary neighborhood are rather good even for a small number of iterations. Maybe this is due to the larger number of neighbors in the secondary neighborhood (i.e. we have $O(n^2)$ neighbors). An increase of the number of iterations does not improve the solution quality. We

conjecture that this is due to a small number of locally optimal solutions, i.e. already after a small number of iterations the secondary neighborhood has been inspected quite good. Hence, further tests about the number of locally optimal solutions would be of interest.

Summarizing, a main reason for the good results for the problems $1|prec|\sum C_i$ and $1|prec|\sum T_i$ could be the fact that the time which is used for one step in the secondary neighborhood is relatively small in comparison with the time used for one step in the primary neighborhood. For the problem $P2|C_{\max}$ this difference is too large. Another remarkable fact is that for the problems $1|prec|\sum C_i$ and $1||\sum T_i$ the secondary neighborhood leads to good results within a small number of iterations.

6. Concluding remarks

We have tried to improve the given neighborhoods for certain scheduling problems with respect to certain local search methods. The main idea was to construct a secondary neighborhood on the set of solutions which are locally optimal with respect to the given neighborhood. First computational tests gave promising results. Although the methods presented were problem specific, the underlying idea can be applied to other problems as well. A search in this direction is a topic of future research.

Acknowledgement

The authors are grateful to the anonymous referees for their helpful comments on earlier drafts of the paper. We also thank Elvira Kraft for providing numerical results.

References

[1] D. de Werra and A Hertz, Tabu search techniques: a tutorial and an application to neural networks, OR Spectrum 11 (1989) 131–141.
[2] J. Du and J.Y.-T. Leung, Minimizing total tardiness on one machine is NP-hard, Math. Oper. Res. 15 (1990) 483–495.
[3] F. Glover, Tabu search, Part I, ORSA J. Comput. 1 (1989) 190–206.
[4] F. Glover, Tabu search, Part II, ORSA J. Comput. 2 (1990) 4–32.
[5] E. Lawler, A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness, Ann. Discrete Math. 1 (1977) 331–342.
[6] E. Lawler, Sequencing jobs to minimize total weighted completion time subject to precedence constraints, Ann. Discrete Math. 2 (1978) 75–90.
[7] O. Martin, S.W. Otto and E.W. Felten, Large-step Markov chains for the TSP incorporating local search heuristics, Oper. Res. Lett. 11 (1992) 219–224.
[8] A.H.G. Rinnooy Kan, B.J. Lageweg and J.K. Lenstra, Minimizing total costs in one-machine scheduling, Oper. Res. 23 (1975) 908–927.

[9] M. Thole, Lösung von multi-purpose Job-Shop Problemen durch Tabu-Suche, Diplomarbeit, Fachbereich Mathematik/Informatik, Universität Osnabrück (1993).
[10] N.L.J. Ulder, Genetic local search: a population-based search algorithm, Master's Thesis, Eindhoven University of Technology (1990).
[11] P.J.M. Van Laarhoven and E.H.L. Aarts, Simulated Annealing; Theory and Applications (Reidel, Dordrecht, 1987).
[12] P.J.M. Van Laarhoven, E.H.L. Aarts and J.K. Lenstra, Job shop scheduling by simulated annealing, Oper. Res. 40 (1992) 113–125.