ELSEVIER

Discrete Applied Mathematics 119 (2002) 181–203

DISCRETE APPLIED MATHEMATICS

A tabu search algorithm for scheduling a single robot in a job-shop environment

Johann Hurink[a,*], Sigrid Knust[b,1]

[a] *Faculty of Mathematical Sciences, University of Twente, P.O. Box 217, NL-7500 AE Enschede, Netherlands*

[b] *Fachbereich Mathematik/Informatik, Universität Osnabrück, D-49069 Osnabrück, Germany*

Received 26 August 1999; received in revised form 8 June 2000; accepted 29 October 2000

Abstract

We consider a single-machine scheduling problem which arises as a subproblem in a job-shop environment where the jobs have to be transported between the machines by a single transport robot. The robot scheduling problem may be regarded as a generalization of the traveling salesman problem with time windows, where additionally generalized precedence constraints have to be respected. The objective is to determine a sequence of all nodes and corresponding starting times in the given time windows in such a way that all generalized precedence relations are respected and the sum of all traveling and waiting times is minimized. We present a local search algorithm for this problem where an appropriate neighborhood structure is defined using problem-specific properties. In order to make the search process more efficient, we apply some techniques which accelerate the evaluation of the solutions in the proposed neighborhood considerably. Computational results are presented for test data arising from job-shop instances with a single transport robot.

Keywords: Scheduling; Traveling salesman problem with time windows; Time-lags; Tabu search

1. Introduction

In this paper we consider a single-machine scheduling problem which arises as a subproblem in a job-shop environment where the jobs additionally have to be transported between the machines by a single transport robot. A *job-shop problem with transportation times and a single robot* is a generalization of the classical job-shop problem and may be formulated as follows: We are given m machines and n jobs. Each job consists of a chain of operations which have to be processed in this order.

* Corresponding author. Tel.: +31-534893447; fax: +31-534894858.
E-mail address: hurink@math.utwente.nl (J. Hurink).

[1] Supported by the Deutsche Forschungsgemeinschaft, Project 'Komplexe Maschinen-Schedulingprobleme'.

0166-218X/02/$ - see front matter
PII: S0166-218X(01)00273-6

With each operation a dedicated machine is associated on which the operation has to be processed without preemption for a given duration. Each machine can process at most one operation at a time. Additionally, *transportation times* are considered. They occur if a job changes from one machine to another and depend on the jobs and the machines between which the transport takes place. We assume that all these transport operations have to be done by a single transport robot which can handle at most one job at a time. The objective is to determine a feasible schedule which minimizes the makespan, i.e. the completion time of the operation processed last.

If for the robot only the given transportation times are important, we may consider the robot as an additional "machine" which has to "process" all transport operations. Therefore, in this case the problem is equivalent to a classical job-shop problem with $m+1$ machines. Since the robot has to process many more operations than the other machines (each second operation of a job), it is also called a bottleneck machine. However, in practice in addition to the transportation times also *empty moving times* arise when the robot moves empty between two machines without carrying a job. These empty moving times may be regarded as sequence-dependent setup times on the robot and, thus, the empty moving times imply that the robot cannot be treated in the same way as the other machines. Consequently, the job-shop problem with transportation times and a single robot consists of scheduling a set of "classical" machines and a special machine on which additionally sequence-dependent setup times have to be taken into account.

Since for scheduling the machines in a classical job-shop many techniques are known from the literature, a possible solution approach to integrate a transportation stage into existing procedures is to apply a two-level approach, where on the first level machine orders for the job-shop machines are fixed and on the second level a corresponding robot order is constructed. The resulting robot scheduling problem on the second level corresponds to the single-machine problem considered in this paper. It consists of determining a schedule for the transport robot when all orders of the operations on the machines are fixed. The robot order has to be chosen such that it is compatible with the given orders for the machines and leads to a schedule with minimal makespan among all robot orders which are compatible with the given machine orders.

The robot has to perform all transportations which have durations equal to the corresponding transportation times. Due to the given orders of operations belonging to the same job, precedence constraints (in form of chains) are induced between the transport operations belonging to the same job. Since each job has to be processed on a machine between two consecutive transportations, time-lags (equal to the corresponding processing time) are associated with these precedences. In the same way precedences (and associated time-lags) between transport operations belonging to different jobs are induced by the fixed orders on the machines. Besides these precedences also the empty moving times of the robot have to be considered. When the robot has finished the transportation of a job, it may have to move empty to another machine where it takes the next job. The corresponding empty moving times in between may be regarded as

sequence-dependent setup times. Furthermore, the given job and machine orders for the operations of the jobs result in earliest possible starting times (release dates) for the transport operations, which have to be respected in all feasible schedules of the robot. Finally, the completion time of the last transport operation in a schedule of the robot (makespan of the robot) does not coincide with the makespan of the complete job-shop schedule since after each transport operation of a job this job has to be scheduled at least on one further machine. Thus, for each transport operation j we have besides a processing time p_j a so-called tail q_j which corresponds to the minimal time period after the completion time of this operation before the complete job-shop schedule is finished. Thus, in the well-known $\alpha|\beta|\gamma$-notation (cf. [9]) the single-machine problem for the robot can be denoted by $1|prec(l_{ij}), r_j, s_{ij}|\max\{C_j + q_j\}$, where $prec(l_{ij})$ indicates arbitrary non-negative finish-start time-lags $l_{ij} \geqslant 0$, s_{ij} stands for sequence-dependent setup times, r_j for release dates (heads), and $C_j + q_j$ denotes for each job the sum of its completion time and its tail. Since this problem generalizes several $\mathcal{NP}$-hard problems, it itself is strongly $\mathcal{NP}$-hard.

The decision version of the problem (i.e. given a threshold value T, does a feasible schedule exist with $\max\{C_j + q_j\} \leqslant T$?) may also be regarded as a generalization of an asymmetric traveling salesman problem with time windows (ATSP-TW) where the traveling times between nodes i and j are given by $p_i + s_{ij}$ and additionally generalized precedence constraints (minimal time-lags) have to be respected. Since q_j is a lower bound for the time we need after finishing operation j, in each schedule with an objective less than or equal to T operation j has to be started in its time window $[r_j, d_j]$ with the deadline $d_j := T - p_j - q_j$. Several algorithms have been developed for the standard ATSP-TW with different objective functions (cf. [4,5]). Mingozzi et al. [12] proposed a dynamic programming approach for a generalization of the ATSP-TW in which additionally "ordinary" precedence relations (i.e. without associated time-lags) have to be respected. Ascheuer [1] investigated an application of the ATSP-TW in an automated storage system in which the time for unloaded moves of a stacker crane has to be minimized. While in this application the objective of total travel time is considered (i.e. the sum of traveling times), in our problem the total scheduling time (i.e. the sum of traveling and waiting times) has to be minimized.

In this paper we present a tabu search algorithm which calculates heuristic solutions for the considered problem. The tabu search method (cf. [6,7]) belongs to the class of local search methods which start with an initial solution and iteratively search through the solution space to find better solutions. In each step the current solution is replaced by a solution in some neighborhood of the solution. The choice of the neighborhood structure has an important influence on the efficiency of local search methods since it determines how we navigate through the solution space.

The remainder of the paper is organized as follows. In Section 2 we give a formal definition of the considered robot scheduling problem, state some additional assumptions, and describe how solutions are represented. Suitable neighborhood structures which are based on problem-specific properties are presented in Section 3. In Section 4 we propose some techniques which accelerate the evaluation of neighbored solutions

in the search process considerably. A tabu search procedure based on all these concepts is described in Section 5 and some computational results can be found in Section 6.

2. Problem formulation

In this section we give a formal definition of the robot scheduling problem in a job-shop environment for fixed machine orders and state some additional assumptions. As mentioned in the introduction, this scheduling problem is also an interesting single-machine problem on its own since it generalizes the ATSP-TW.

We are given a set $V := \{1,\dots,n\} \cup \{0, n+1\}$ of jobs where 0 and $n+1$ are dummy jobs with $p_0 = p_{n+1} = 0$. All jobs $j \in V$ have to be processed without preemptions for p_j time units on a single machine. Job j cannot be started before its release date (head) r_j and stays in the system for q_j time units (tail) after processing. Furthermore, a set C of precedence relations $i \to j$ with associated minimal finish-start time-lags $l_{ij} \geqslant 0$ is given. If the relation $i \to j$ (also called a conjunction) is in C, job j cannot be started earlier than l_{ij} time units after the completion of job i. We assume that all transitive arcs $i \to j$ are contained in the set C and that the time-lags are transitively adjusted, i.e. $l_{ih} + p_h + l_{hj} \leqslant l_{ij}$ holds. Furthermore, we regard the heads and tails as special time-lags by setting $l_{0j} := r_j$ and $l_{j,n+1} := q_j$ for all jobs $j = 1,\dots,n$.

For all pairs of jobs i, j sequence-dependent setup times s_{ij} and s_{ji} are given, which occur if these jobs are processed directly after each other. If, additionally, a time-lag l_{ij} exists, the setup time s_{ij} has no influence, i.e. we may assume $l_{ij} \geqslant s_{ij}$ for all i, j with $i \to j \in C$. In order to have a unique notation for the delay of job j after the completion of job i if j is planned directly after i, we define

$$t_{ij} := \begin{cases} l_{ij} & \text{if } i \to j \in C, \\ s_{ij} & \text{if } i \to j \notin C \text{ and } j \to i \notin C \end{cases} \tag{2.1}$$

for all jobs $i, j \in V$ where i may be processed before j, i.e. for which the relation $j \to i$ is not contained in C.

We assume that the setup times satisfy the weak triangle inequality $s_{ih} + p_h + s_{hj} \geqslant s_{ij}$. Furthermore, minimal time-lags and setups are supposed to satisfy the triangle inequalities

$$l_{ih} + p_h + s_{hj} \geqslant s_{ij} \quad \text{and} \quad s_{ih} + p_h + l_{hj} \geqslant s_{ij}. \tag{2.2}$$

Since the time-lags are transitively adjusted, the inequality $l_{ih} + p_h + l_{hj} \geqslant s_{ij}$ is not supposed (because then also $i \to j \in C$ holds). Note that all these triangle inequalities hold in instances of the robot scheduling problem if the transportation and empty moving times satisfy some additional restrictions. We require that for the transportation and empty moving times triangle inequalities hold and that an empty move between two machines does not take longer than carrying a job between the same machines (see [10]). These conditions are satisfied in most practical situations.

The problem is to determine feasible starting times S_j (and, thus, completion times $C_j = S_j + p_j$) for all jobs $j \in V$ which respect the release dates, the minimal time-lags and the setup times, and minimize the objective $\max_{j=1,\ldots,n}\{C_j + q_j\} = C_{n+1} = C_{\max}$.

In the following, we will show that the solution space may be represented by the set of all permutations of the jobs $j \in V$ which are compatible with the precedence relations (i.e. if $i \to j \in C$ and $\pi_k = i$, $\pi_l = j$ in a permutation π, we have $k < l$). Obviously, each feasible schedule S for the single-machine problem uniquely defines a permutation $\pi^S = (0, \pi_1^S, \ldots, \pi_n^S, n+1)$ in which the jobs are ordered according to increasing starting times. On the other hand, with each permutation $\pi = (0, \pi_1, \ldots, \pi_n, n+1)$ which is compatible with the precedence constraints we may associate the set of all feasible schedules in which the jobs are processed according to the order defined by π.

Among these schedules represented by π a schedule S^π with minimal makespan may be determined as follows. We set $S_0^\pi := 0$ and calculate for $k = 1, \ldots, n+1$ the starting time S_j^π of job $j := \pi_k$ by

$$S_j^\pi := \max\left[\max_{\{i \mid i \to j \in C\}}\{S_i^\pi + p_i + l_{ij}\}, S_{\pi_{k-1}}^\pi + p_{\pi_{k-1}} + s_{\pi_{k-1}j}\right]. \tag{2.3}$$

Obviously, job j cannot start before the time given by the first term

$$L_j^\pi := \max_{\{i \mid i \to j \in C\}} i\{S_i^\pi + p_i + l_{ij}\}, \tag{2.4}$$

which ensures that all minimal time-lags l_{ij} are respected. The second term in (2.3) takes into account that j cannot start before its immediate predecessor π_{k-1} on the machine has finished and the setup $s_{\pi_{k-1}j}$ has been done. The complexity for calculating the schedule S^π is $O(n + |C|) = O(n^2)$.

The calculation of the starting times S_j^π implies that the resulting schedule S^π is an optimal schedule among all schedules represented by π since each job starts as early as possible. The makespan of S^π is given by $C_{\max}(S^\pi) = S_{\pi_{n+1}}^\pi$. This value corresponds to the length of a longest path in the acyclic graph $G^\pi = (V, A^\pi)$ with arcs $A^\pi := C \cup \{(\pi_k, \pi_{k+1}) \mid k = 1, \ldots, n-1\}$ and arc lengths

$$d_{ij}^\pi := \begin{cases} p_i + l_{ij} & \text{if } i \to j \in C \\ p_i + s_{ij} & \text{if } i \to j \notin C \end{cases} \quad \text{for } (i,j) \in A^\pi.$$

For further considerations we are also interested in an optimal schedule $\bar{S}^\pi$ represented by π in which each job starts as late as possible. After calculating $C_{\max}(\pi) := C_{\max}(S^\pi)$, schedule $\bar{S}^\pi$ may be constructed by a backward calculation. We set $\bar{S}_{n+1}^\pi := C_{\max}(\pi)$ and calculate for $k = n, \ldots, 0$ the starting time $\bar{S}_i^\pi$ of job $i := \pi_k$ by

$$\bar{S}_i^\pi := \min\left[\min_{\{j \mid i \to j \in C\}}\{\bar{S}_j^\pi - p_i - l_{ij}\}, \bar{S}_{\pi_{k+1}}^\pi - p_{\pi_k} - s_{i\pi_{k+1}}\right]. \tag{2.5}$$

Note that S_i^π is the earliest starting time and $\bar{S}_i^\pi$ is the latest starting time for job i among all feasible schedules which are represented by π with makespan $C_{\max}(\pi)$.

Jobs $i \in V$ for which $S_i^\pi = \bar{S}_i^\pi$ holds, are called *critical*, since they cannot be moved in any optimal schedule represented by π. A sequence $(0 = i_0, i_1, \ldots, i_k, i_{k+1} = n+1)$ of

critical jobs is called a *critical path* if for all pairs $(i,j):=(i_\lambda, i_{\lambda+1})$ with $\lambda \in \{0,\ldots,k\}$ the starting time of job j is determined by job i, i.e. $S_j^\pi = S_i^\pi + d_{ij}^\pi$ holds. Such a path always exists and it is easy to see that it corresponds to a longest path in the graph G^π introduced above.

Based on this representation of feasible schedules by permutations, in the following we describe a local search procedure for calculating heuristic solutions.

3. Neighborhood structures

In this section we describe how suitable neighborhood structures can be defined based on structural properties of the single-machine problem. Given a permutation π with the associated schedules S^π and $\bar{S}^\pi$, we will define neighborhoods of π which take into account that the basic goal of local search is to improve the solution.

To incorporate problem-specific properties into the definition of neighborhood structures we use a so-called block approach. Such an approach was first proposed for the single-machine problem $1 \,|\, r_j \,|\, L_{\max}$ [8]. Later it was successfully adapted to some other scheduling problems (like the job-shop or flow-shop problem, cf. [2,14,15]). With the definition of blocks it can be stated that only certain changes of a solution π may have a chance to improve the current makespan $C_{\max}(\pi)$ and in the search process only such solutions will be considered as candidates for neighbored solutions.

Let $P^\pi = (0 = i_0, i_1, \ldots, i_k, i_{k+1} = n+1)$ be a critical path associated with π. A subsequence $(i_b,\ldots,i_f)$ of at least two successive jobs (i.e. $f > b$) on P^π is called a *block* if

- the jobs of the subsequence are processed consecutively on the single machine without idle times (idle time is time which is not used for processing or setups),
- no conjunction exists between two consecutive jobs of the subsequence, and
- enlarging the subsequence by one job leads to a subsequence which does not fulfill both of the above properties.

The definition of a block implies that for all pairs $(i_\lambda, i_{\lambda+1})$ in a block $(i_b,\ldots,i_f)$ the equality $S_{i_{\lambda+1}}^\pi = S_{i_\lambda}^\pi + p_{i_\lambda} + s_{i_\lambda i_{\lambda+1}}$ holds.

The following theorem is the basis for defining suitable neighborhoods on the set of all permutations.

Theorem 1. *Let π be an arbitrary permutation with makespan $C_{\max}(\pi)$ and let P^π be a critical path associated with π. If another permutation π' with $C_{\max}(\pi') < C_{\max}(\pi)$ exists, then in π' at least two jobs of a block on P^π are processed in the opposite order as in π.*

Proof. Let $P^\pi = (0, u_1^1, u_2^1, \ldots, u_{m_1}^1, \ldots, u_1^k, u_2^k, \ldots, u_{m_k}^k, n+1)$, where $u_1^j,\ldots,u_{m_j}^j$ $(j=1,\ldots,k)$ denotes a maximal number of jobs which are processed consecutively on the single machine and no conjunction exists between two consecutive jobs (i.e. a block if $m_j > 1$).

Assume that a permutation π' with $C_{\max}(\pi') < C_{\max}(\pi)$ exists and all jobs of the blocks on P^π are processed in the same order as in π. Then the graph $G^{\pi'}$ contains a path which consists of the jobs of P^π in the same order as in π and which possibly contains some additional jobs in between. This means that we again have the arcs $0 \to u_1^1$, $u_{m_j}^j \to u_1^{j+1}$ for $j = 1, \ldots, k-1$ and $u_{m_k}^k \to n+1$ with the same arc lengths as in G^π. It remains to consider two arbitrary jobs $u_\lambda^j, u_{\lambda+1}^j$ which may be separated by some additional jobs $v_\lambda^j, \ldots, v_\mu^j$. By iteratively applying the triangle inequalities (2.2) $s_{uv} + p_v + s_{vw} \geqslant s_{uw}$, $l_{uv} + p_v + s_{vw} \geqslant s_{uw}$, $s_{uv} + p_v + l_{vw} \geqslant s_{uw}$ we obtain that the length of the path between u_λ^j and $u_{\lambda+1}^j$ in $G^{\pi'}$ is not smaller than the arc length $d^\pi_{u_\lambda^j u_{\lambda+1}^j} = p_{u_\lambda^j} + s_{u_\lambda^j u_{\lambda+1}^j}$ in G^π. Thus, we have $C_{\max}(\pi') \geqslant C_{\max}(\pi)$, which is a contradiction. □

Contrary to the job-shop problem the interchange of internal jobs in a block may improve the current objective value due to the setup times. Based on the theorem which gives necessary conditions for a permutation π' to be better than π, we introduce the following neighborhood $\mathcal{N}_1$.

Neighborhood $\mathcal{N}_1$: Let π be an arbitrary permutation and let P^π be a critical path associated with π. Neighborhood $\mathcal{N}_1(\pi)$ contains all feasible permutations π' (compatible with the precedence relations) which can be constructed by interchanging two adjacent jobs in a block on the critical path P^π.

Simple examples show that neighborhood $\mathcal{N}_1$ is not connected, i.e. it is not possible to transform two arbitrary permutations into each other by iteratively applying only changes according to $\mathcal{N}_1$. But the neighborhood $\mathcal{N}_1$ is opt-connected, which means that from each permutation an optimal solution can be reached by a finite sequence of moves in the neighborhood. This can be shown by an adaption of the proof in [11] who considered $\mathcal{N}_1$ in connection with the job-shop problem. Furthermore, since due to the definition of a block between two adjacent jobs no precedence relation exists, an interchange of two adjacent jobs in a block always results in a feasible permutation π'.

A disadvantage of neighborhood $\mathcal{N}_1$ is that permutations are changed only slightly by the interchange of two adjacent jobs in a block. Thus, many interchanges may be necessary to change the structure of a given solution significantly. To decrease this number of moves, we extend $\mathcal{N}_1$ in the following way.

Neighborhood $\mathcal{N}_2$: Let π be an arbitrary permutation and let P^π be a critical path associated with π. Neighborhood $\mathcal{N}_2(\pi)$ contains all feasible permutations π' which can be constructed as follows:

- two adjacent jobs of a block on P^π are interchanged,
- in π' the first job of a block B on P^π is shifted to the right to another position in B, or
- in π' one job of a block B on P^π different from the last one is shifted to the end of B.

Since neighborhood $\mathcal{N}_2$ contains $\mathcal{N}_1$ as a subneighborhood (i.e. $\mathcal{N}_1(\pi) \subseteq \mathcal{N}_2(\pi)$ for each permutation π), $\mathcal{N}_2$ is also opt-connected. The size of both neighborhoods

is polynomially bounded by $O(n)$ because the number of jobs in blocks on a critical path is bounded by n.

Formally, the two neighborhoods can be defined by the following operators which may be applied to every block B on a critical path P^π:

- api_k for $k = 1, \ldots, |B| - 2$ interchanges the kth job in block B with its immediate successor,
- $rshift_k$ for $k = 2, \ldots, |B| - 1$ shifts the first job of block B to the kth position in the same block, and
- $endshift_k$ for $k = 1, \ldots, |B| - 1$ shifts the kth job in block B to the last position in B.

Note that the operator api_1 results in the same permutation as $rshift_2$, the operator $rshift_{|B|}$ is the same as $endshift_1$ and $api_{|B|-1}$ equals $endshift_{|B|-1}$. Since within the following considerations the operators $api_{|B|-1}$ and $rshift_{|B|}$ cannot be treated in the same way as the other operators of the same type, we included them into the set of the endshift operators where they can be dealt with in a unique context.

Local search methods usually evaluate all solutions in the neighborhood of the current solution and choose the best one as a new solution. For a straightforward calculation of the best neighbor of a permutation we have to calculate the objective value for $O(n)$ neighbored permutations. Although a neighbored permutation differs only slightly from the current permutation, the starting times for all jobs which appear after the shifted job have to be recalculated to calculate the makespan of a neighbored solution. These starting times may be changed by different amounts due to the minimal time-lags (according to (2.3)). Thus, the straightforward calculation of a best neighbor will need $O(n(n + |C|))$ time since calculating the objective value of a permutation needs $O(n + |C|) = O(n^2)$ time. In order to make the search process more efficient we do not calculate the correct objective values for all neighbored solutions, but use some approximate values. Such a procedure will be described in the next section.

4. Efficient evaluation of neighbors

In this section we will present some techniques which accelerate the evaluation of solutions in the proposed neighborhoods. We will describe an approach which calculates a hopefully good (but not necessarily the best) neighbor in a shorter amount of time. The whole process of evaluating all neighbors for the operators api_k and $rshift_k$ will be done in $O(n+|C|)$ time using estimations for the real objective values. Then the $O(n+|C|)$-procedure for calculating the correct objective value has to be applied only once for the determined neighbor. Thus, we are able to evaluate *all api*- and *rshift*-neighbors with the same time complexity which is needed for calculating the objective value of a *single* neighbor. This means that the computational time for evaluating all *api*- and *rshift*-neighbors of a solution is reduced by the factor n. Unfortunately, for the operator $endshift_k$ we are not able to evaluate all *endshift*-neighbors with worst-case complexity $O(n + |C|)$. By using similar estimations as for the other two operators,

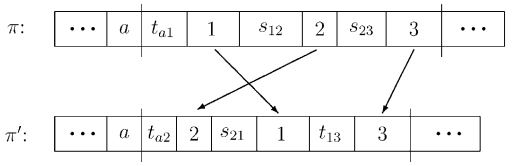

Fig. 1. Permutations π and $\pi' = a\,pi_k(\pi)$.

we propose an approach which evaluates all *endshift*-neighbors of a solution π with worst-case complexity $O(|C|+\sum_{B\in P^\pi}|B|^2) = O(|C|+n^2)$, which is, nevertheless, better than $O(n|C|+n^2)$.

We will describe the used estimates for neighbored permutations in dependence of the operators applied to generate the neighbors. At first we consider the operator api_k with $k \in \{1,\ldots,|B|-2\}$ which interchanges two adjacent jobs in a block B on a critical path P^π. According to the definition of a block between such two jobs no precedence relation exists. Thus, all operators api_k lead to feasible permutations $\pi' = api_k(\pi)$. To apply api_k to a block B, B has to contain at least 3 jobs. Without loss of generality let $B{:=}(i_b,\ldots,1,2,3,\ldots,i_f)$ and let 1 be the k-th job of B. Then $\pi' = api_k(\pi)$ has the form $\pi' = (\ldots,i_b,\ldots,2,1,3,\ldots,i_f,\ldots)$.

To obtain an estimate of the objective value of π', we will use estimates for the starting times $S_j^{\pi'}$ of the jobs in π'. These estimates are calculated using the current starting times S_j^π and the lower bounds L_j^π for the starting times of the jobs in π based on the time-lags (cf. (2.4)). For the sake of simplicity in the following we will denote the starting times of the jobs j in π by $S_j := S_j^\pi$ and the lower bounds by $L_j{:=}L_j^\pi$. Again, t_{ij} denotes the amount of time which has to be considered between the completion time of job i and the starting time of j (cf. (2.1)).

In a first step we will consider the new starting times $S'_j{:=}S_j^{\pi'}$ for the jobs $j \in \{1,2,3\}$ (cf. Fig. 1). Let a denote the immediate predecessor of job 1 in π. Obviously, we have

$$S'_2 = \max\{S_a + p_a + t_{a2}, L_2\} \tag{4.1}$$

since the L_2-value is not changed by the operator api_k. For the same reason the new starting time of job 1 is given by

$$S'_1 = \max\{S'_2 + p_2 + s_{21}, L_1\}. \tag{4.2}$$

The value L_3 may only be changed by the operator if the conjunction $1 \to 3 \in C$ exists. In this case L'_3 is given by

$$L'_3 = \max\left[\begin{array}{c}\max\limits_{\substack{\{j|j\neq 1;\\ j\to 3\in C\}}}\{S_j + p_j + l_{j3}\}, S'_1 + p_1 + l_{13}\end{array}\right].$$

This expression equals $\max\{L_3,\ S'_1 + p_1 + l_{13}\}$ since next we will show that job 1 is never moved to the left (i.e. $S_1 \leqslant S'_1$ holds). Furthermore, due to the definition of

blocks no conjunction $2 \to 3 \in C$ exists, i.e. we have

$$S_3' = \max\{S_1' + p_1 + t_{13}, L_3\}. \tag{4.3}$$

Due to the weak triangle inequality for the setup times s_{ij} it is easy to show that the following inequalities hold:

$$\Delta_2 := S_2' - S_2 \leqslant 0, \quad \Delta_1 := S_1' - S_1 \geqslant 0, \quad \Delta_3 := S_3' - S_3 \geqslant \Delta_2.$$

Thus, job 2 is not moved to the right and job 1 is not moved to the left. Furthermore, if job 3 is moved to the left, it cannot be moved by more than $|\Delta_2|$ units.

Since job 3 is critical with respect to π, we have $S_3 = \bar{S}_3^{\pi}$ and Δ_3 has the following meaning:

- If Δ_3 is non-negative, the sequence of critical jobs after job 3 has to be shifted to the right by at least Δ_3 time units to respect all time-lags l_{3j} and the machine capacity in π'.
- If Δ_3 is negative, $|\Delta_3|$ defines an upper bound for the amount of time by which the whole sequence of jobs after job 3 can be shifted to the left (if we shift this sequence by $|\Delta_3|$ time units, all time-lags l_{3j} and the machine capacity are respected, but time-lags l_{ij} between jobs i, j may be violated where i is placed before job 3 and j is placed after job 3 in π).

To obtain an estimate of the new starting times of the jobs scheduled after job 3 we also estimate how the jobs j with $1 \to j \in C$ are influenced by the right-shift of job 1. Since such a job j cannot be started before time $S_1' + p_1 + l_{1j}$ in the schedule $S^{\pi'}$, the value $\theta_j := S_1' + p_1 + l_{1j} - \bar{S}_j^{\pi}$ has the following meaning:

- If θ_j is non-negative, some jobs after job j exist which have to be shifted to the right by at least θ_j time units to respect the time-lags l_{1j}.
- If θ_j is negative, $|\theta_j|$ defines an upper bound for the amount of time by which the whole sequence of jobs scheduled after job j can be shifted to the left respecting only the time-lags l_{1j}.

Using the maximal θ_j-value of all conjunctive successors j of job 1,

$$\Delta_1^{\text{succ}} := \max_{\{j \mid 1 \to j \in C\}} \{\theta_j\} = \max_{\{j \mid 1 \to j \in C\}} \{S_1' + p_1 + l_{1j} - \bar{S}_j^{\pi}\} \tag{4.4}$$

and the value Δ_3 the following proposition can be proven:

Proposition 2. *Let* $\Delta := \max\{\Delta_3, \Delta_1^{\text{succ}}\}$. *Then the objective value of the permutation* $\pi' = api_k(\pi)$ *can be estimated as follows*:
(a) *if* $\Delta \geqslant 0$ *holds, we have* $C_{\max}(\pi') = C_{\max}(\pi) + \Delta$, *and*
(b) *if* $\Delta < 0$ *holds, we have* $C_{\max}(\pi) \geqslant C_{\max}(\pi') \geqslant C_{\max}(\pi) - |\Delta|$.

Proof. Consider the schedule S' where the jobs j scheduled before the interchanged jobs start at their old starting times S_j, the starting times for the jobs $j \in \{1, 2, 3\}$ are given by (4.1) to (4.3), and the jobs j scheduled after job 3 start at time $S_j' = \bar{S}_j^{\pi} + \Delta$. Obviously, this schedule S' is feasible with respect to the setup times. Thus, it remains to consider the time-lags.

Case (a) $\Delta \geqslant 0$: Since all jobs scheduled after job 3 are shifted by the same amount Δ to the right and $\Delta \geqslant \Delta_3$ holds, only time-lags between job 1 and the jobs scheduled after job 3 may be violated (job 1 is the only job which may have been shifted by more than Δ to the right). However, due to the definition of Δ_1^{succ} all these time-lags are respected. Thus, the schedule S' is a feasible schedule for sequence π' and has a makespan of length $C_{\max}(\pi) + \Delta$.

This schedule is also a best-possible schedule for sequence π' since in the case $\Delta = \Delta_3$ job 3 remains a critical job and in the case $\Delta = \Delta_1^{\text{succ}}$ job 1 followed by job j determining the maximum in (4.4) are part of a critical path.

Case (b) $\Delta < 0$: Since $\Delta \geqslant \Delta_3$ and all time-lags l_{1j} are respected due to the definition of Δ_1^{succ}, only time-lags between jobs scheduled before job 3 and jobs scheduled after job 3 may be violated. If no such time-lag is violated, the schedule S' is a best possible schedule for sequence π' (same argumentation as above). On the other hand, if some time-lags l_{ij} are violated, the makespan of S' is a lower bound on the best possible makespan (using again the critical job or critical path argument from above).

An upper bound for the minimal makespan of π' can be achieved by considering the schedule S'', which is the same as S', but the jobs j scheduled after job 3 start at their latest starting times (i.e. $S_j'' = \bar{S}_j^{\pi}$). The resulting schedule is feasible since all time-lags l_{ij} with $i \neq 2$ were already satisfied in $\bar{S}$ and none of the time-lags l_{2j} are violated due to $\Delta_2 \leqslant 0$.

Summarizing, we get $C_{\max}(\pi) \geqslant C_{\max}(\pi') \geqslant C_{\max}(\pi) - |\Delta|$. □

From Proposition 2 the following conclusions can be drawn:

- in case (a) we know that π' does not improve π and the exact objective value of π' can be determined directly from the estimate Δ, and
- in case (b) we know that π' is not worse than π and the estimate Δ leads to an upper bound for the improvement.

The time complexity for the described process of evaluating all neighbors api_k for all blocks B and $k = 1, \ldots, |B| - 2$ can be determined as follows. Obviously, for each of the $|B| - 2$ neighbors for a block B we can calculate the three starting times according to (4.1)–(4.3) in constant time. Since during the calculation of all Δ_1^{succ}-values each conjunction in C is considered at most once and all blocks together contain at most n jobs, the whole process of evaluating all *api*-neighbors for all blocks can be done in $O(n + |C|)$ time.

Next, we consider the operator $rshift_k$ with $k \in \{2, \ldots, |B| - 1\}$ which shifts the first job of a block B on a critical path P^{π} to the right. To apply $rshift_k$ to a block B, B has to contain at least 3 jobs. Without loss of generality let $B := (1, 2, 3, \ldots, f)$. Then $\pi^{(k)} = rshift_k(\pi)$ is feasible if no conjunction $1 \rightarrow j \in C$ with $j \in \{3, \ldots, k\}$ exists and $\pi^{(k)}$ has the form $\pi^{(k)} = (\ldots, 2, \ldots, k, 1, k+1, \ldots, f, \ldots)$. Note that if a conjunction $1 \rightarrow h \in C$ with $h \in B$ exists, all permutations $\pi^{(k)} = rshift_k(\pi)$ for $k = h, \ldots, f$ are infeasible. Thus, we can stop evaluating the *rshift*-operators for a block as soon as we detect the first conjunction $1 \rightarrow h \in C$ with $h \in B$.

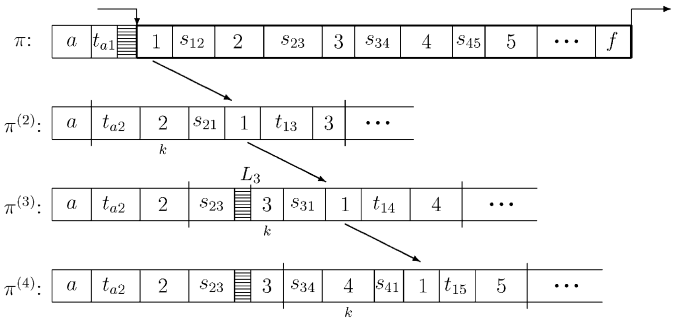

Fig. 2. Permutations π and $\pi^{(k)} = rshift_k(\pi)$.

To obtain an estimate of the makespan of $\pi^{(k)}$, we will again calculate the correct new starting times for the jobs involved in the shift and use these values to estimate the starting times of the remaining jobs. The new starting times $S_j^{(k)} := S_j^{\pi^{(k)}}$ for the jobs $j \in \{1,\ldots,k+1\}$ (see Fig. 2) can be calculated as follows. Since the operator $rshift_2$ equals api_1, for $k=2$ the starting times $S_j^{(k)}$ with $j \in \{1,2,3\}$ are given by (4.1) –(4.3), i.e.

$$S_2^{(2)} = \max\{S_a + p_a + t_{a2}, L_2\}, \tag{4.5}$$

$$S_1^{(2)} = \max\{S_2^{(2)} + p_2 + s_{21}, L_1\}, \tag{4.6}$$

$$S_3^{(2)} = \max\{S_1^{(2)} + p_1 + t_{13}, L_3^{(2)}\}, \tag{4.7}$$

where a denotes the direct predecessor of job 1 in π and the new value $L_3^{(2)}$ may again be replaced by the old value L_3.

For $k=3,\ldots,|B|-1$ we proceed as follows. By comparing $\pi^{(k-1)}$ and $\pi^{(k)}$ we see that the starting times for the jobs $j=2,\ldots,k-1$ placed before job k are the same (i.e. $S_j^{(k)} = S_j^{(k-1)}$). Thus, only the three new starting times $S_k^{(k)}, S_1^{(k)}, S_{k+1}^{(k)}$ have to be calculated for $\pi^{(k)}$. They are given by

$$S_k^{(k)} = \max\{S_{k-1}^{(k)} + p_{k-1} + s_{k-1,k}, L_k^{(k-1)}\}, \tag{4.8}$$

$$S_1^{(k)} = \max\{S_k^{(k)} + p_k + s_{k1}, L_1\}, \tag{4.9}$$

$$S_{k+1}^{(k)} = \max\{S_1^{(k)} + p_1 + t_{1,k+1}, L_{k+1}^{(k)}\}, \tag{4.10}$$

where the values $L_{v+1}^{(v)}$ in (4.8) and (4.10) denote the lower bounds for the starting time of job $v+1$ in $\pi^{(v)}$ with respect to the time-lags $l_{j,v+1}$ with $j \neq 1$, i.e.

$$L_{v+1}^{(v)} = \max_{\{j \neq 1 | j \to v+1 \in C\}} \{S_j^{(v)} + p_j + l_{j,v+1}\}.$$

Note that contrary to the *api*-operator these lower bound values may not be replaced by the old values.

Due to the weak triangle inequality the following inequalities hold:

$$\Delta_k^{(k)} := S_k^{(k)} - S_k \leqslant 0, \quad \Delta_1^{(k)} := S_1^{(k)} - S_1 \geqslant 0, \quad \Delta_{k+1}^{(k)} := S_{k+1}^{(k)} - S_{k+1} \geqslant \Delta_k^{(k)}.$$

Thus, jobs $2,\ldots,k$ are not moved to the right and job 1 is not moved to the left. Furthermore, if job $k+1$ is moved to the left, it cannot be moved by more than $|\Delta_k^{(k)}|$ time units.

The lower bound values $L_k^{(k-1)}$, $L_{k+1}^{(k)}$ in (4.8) and (4.10) do not always have to be calculated from scratch for the new starting times $S_j^{(k)}$ in iteration k. The value $L_k^{(k-1)}$ is known from the previous iteration $k-1$ and the calculation of $L_{k+1}^{(k)}$ in (4.10) can be done as follows. If no job $2,\ldots,k-1$ defines the value L_{k+1} in π, (i.e. $S_j + p_j + l_{j,k+1} < L_{k+1}$ for $j = 2,\ldots,k-1$), then the L_{k+1}-value does not change and we have $L_{k+1}^{(k)} := L_{k+1}$, because jobs $2,\ldots,k-1$ are not moved to the right. Otherwise, if a job $j \in \{2,\ldots,k-1\}$ determines the L_{k+1}-value in π (i.e. $S_j + p_j + l_{j,k+1} = L_{k+1}$), the L_{k+1}-value may be decreased if j is shifted to the left. Then we have to recalculate L_{k+1} according to the new starting times of the jobs $2,\ldots,k-1$ and the old starting times of the remaining jobs:

$$L_{k+1}^{(k)} := \max \begin{cases} \max\limits_{\substack{\{j \mid j=2,\ldots,k-1;\\ j \to k+1 \in C\}}} \{S_j^{(k)} + p_j + l_{j,k+1}\}, \\ \max\limits_{\substack{\{j \mid j\neq 2,\ldots,k-1;\\ j \to k+1 \in C\}}} \{S_j + p_j + l_{j,k+1}\}. \end{cases} \tag{4.11}$$

To estimate the change of the objective value we again use a measure for the influence of this move on the starting times of the jobs scheduled after job 1 in addition to the amount $\Delta_1^{(k)}$ by which job 1 is moved to the right:

$$\Delta_1^{(k)\text{succ}} := \max_{\{j \mid 1 \to j \in C\}} \{S_1^{(k)} + p_1 + l_{1j} - \bar{S}_j^{\pi}\}. \tag{4.12}$$

Analogously to Proposition 2 the following proposition can be proved:

Proposition 3. *Let* $\Delta := \max\{\Delta_{k+1}^{(k)}, \Delta_1^{(k)\text{succ}}\}$. *Then the objective value of the permutation* $\pi^{(k)} = rshift_k(\pi)$ *can be estimated as follows*:
(a) *if* $\Delta \geqslant 0$ *holds, we have* $C_{\max}(\pi') = C_{\max}(\pi) + \Delta$, *and*
(b) *if* $\Delta < 0$ *holds, we have* $C_{\max}(\pi) \geqslant C_{\max}(\pi') \geqslant C_{\max}(\pi) - |\Delta|$.

Note that if $\Delta_1^{(k)\text{succ}} \geqslant 0$ holds for a permutation $\pi^{(k)}$, we also have $\Delta_1^{(h)\text{succ}} \geqslant \Delta_1^{(k)\text{succ}} \geqslant 0$ for all $h = k+1,\ldots,|B|-1$ since in these iterations the starting time of job 1 is not smaller than $S_1^{(k)}$. Thus, if we are only interested in a neighbor with a minimal Δ-value, we do not have to consider the remaining shifts for the current block.

The time complexity for evaluating all *rshift*-neighbors can be determined as follows. If we evaluate the neighbors $\pi^{(k)}$ iteratively for all blocks B and $k = 2,3,\ldots,|B|-1$, only three new starting times have to be determined for $\pi^{(k)}$, as mentioned above. Each of these calculations can be done in constant time if the $L_{k+1}^{(k)}$-values are available.

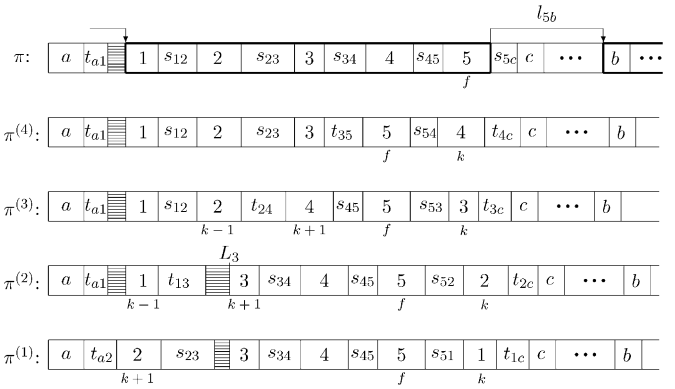

Fig. 3. Permutations π and $\pi^{(k)} = endshift_k(\pi)$.

The calculation of all relevant $L_{k+1}^{(k)}$-values according to (4.11) and all $\Delta_1^{(k)\text{succ}}$-values according to (4.12) can be done in $O(|C|)$ time since each conjunction has to be considered at most once. Hence, the whole process of evaluating all *rshift*-neighbors for all blocks requires $O(n + |C|)$ time.

Finally, we consider the operator $endshift_k$ with $k \in \{1, \ldots, |B| - 1\}$ which shifts the kth job of a block B on a critical path P^π to the end. Since a block contains at least 2 jobs by definition, $endshift_k$ can be applied to any block B. Without loss of generality, let $B{:=}(1, 2, \ldots, f)$, let b be the conjunctive successor of job f on P^π, and let c be the job directly processed after job f in π (see Fig. 3 where $f = 5$). Then $\pi^{(k)} = endshift_k(\pi)$ is feasible if no conjunction $k \to j \in C$ with $j \in \{k+2, \ldots, f\}$ exists and $\pi^{(k)}$ has the form

$$\pi^{(k)} = (\ldots, 1, \ldots, k-1, k+1, \ldots, f, k, c, \ldots, b, \ldots).$$

As for the previously considered operators, it can be shown that jobs $k+1, \ldots, f$ are not moved to the right and job k is not moved to the left.

At first, we only calculate the new starting times $S_j^{(k)} {:=} S_j^{\pi^{(k)}}$ for the moved job k, the final job f of the considered block and its machine successor c. For $k = |B| - 1, \ldots, 1$ we determine the new starting time of the final job f by calculating the amount of time by which it can be shifted to the left. Due to the removal of job k between jobs $k-1$ and $k+1$, the jobs after $k+1$ can be moved by at most $s_{k-1,k} + p_k + s_{k,k+1} - t_{k-1,k+1}$ time units to the left respecting the setup times and the machine capacity. On the other hand, the maximal amount of time by which the whole sequence $(k+1, \ldots, f)$ can be shifted to the left according to the time-lags is given by

$$V_{k+1}^f {:=} \min_{h=k+1,\ldots,f} \{S_h - L_h^{k-1}\}, \tag{4.13}$$

where L_h^{k-1} is a lower bound for the starting time of job h in $\pi^{(k)}$ with respect to the time-lags l_{jh} with $j \neq k+1, \ldots, h-2$, i.e. $L_h^{k-1} = \max_{\{j | j \neq k+1, \ldots, h-2;\ j \to h \in C\}} \{S_j + p_j + l_{jh}\}$. Note that the other time-lags l_{jh} with $j = k+1, \ldots, h-2$ do not have to be considered

for moving the whole sequence $(k+1,\ldots,f)$ to the left. Thus, the new starting time of job f is given by

$$S_f^{(k)} = S_f - \min\{V_{k+1}^f, s_{k-1,k} + p_k + s_{k,k+1} - t_{k-1,k+1}\}. \tag{4.14}$$

Since we assume that permutations π and $\pi^{(k)}$ are feasible, neither a conjunction $j \to k \in C$ nor a conjunction $k \to j \in C$ exists with $j \in \{k+2,\ldots,f\}$. Therefore, the new starting time of job k is only determined by the job directly processed before k on the machine, i.e.

$$S_k^{(k)} = S_f^{(k)} + p_f + s_{fk}. \tag{4.15}$$

Finally, the starting time of the new machine successor c of k is given by

$$S_c^{(k)} = \max\{S_k^{(k)} + p_k + t_{kc}, L_c^{(k)}\}, \tag{4.16}$$

where the value $L_c^{(k)}$ in (4.16) again denotes a lower bound for the starting time of job c in $\pi^{(k)}$ with respect to the time-lags l_{jc} with $j \neq k$, i.e. $L_c^{(k)} = \max_{\{j\neq k | j\to c\in C\}}\{S_j^{(k)} + p_j + l_{jc}\}$.

The calculation of the $L_c^{(k)}$-value can be done very similarly to the case of the *rshift*-operator: If no job $k+1,\ldots,f$ defines the value L_c in π (i.e. $S_j + p_j + l_{jc} < L_c$ for $j = k+1,\ldots,f$), then the L_c-value does not change and we have $L_c^{(k)} := L_c$, because jobs $k+1,\ldots,f$ are not moved to the right. Otherwise, if a job $j \in \{k+1,\ldots,f\}$ determines the L_c-value in π (i.e. $S_j + p_j + l_{jc} = L_c$), the L_c-value may be decreased if j is shifted to the left. Then we have to recalculate L_c according to the new starting times of the jobs $k+1,\ldots,f$ and the old starting times of the remaining jobs:

$$L_c^{(k)} := \max\left[\begin{array}{l} \max\limits_{\substack{\{j | j=k+1,\ldots,f;\\ j\to c\in C\}}} \{S_j^{(k)} + p_j + l_{jc}\}, \\ \\ \max\limits_{\substack{\{j | j\neq k+1,\ldots,f;\\ j\to c\in C\}}} \{S_j + p_j + l_{jc}\} \end{array}\right]. \tag{4.17}$$

Unfortunately, at this point we may need the new starting times $S_j^{(k)}$ for some of the jobs $j = k+1,\ldots,f-1$ (we did not have to calculate them in (4.14) to (4.16)). They are given by

$$S_{k+1}^{(k)} = \max\{S_{k-1} + p_{k-1} + t_{k-1,k+1}, L_{k+1}\}, \tag{4.18}$$

$$S_j^{(k)} = \max\{S_{j-1}^{(k)} + p_{j-1} + s_{j-1,j}, L_j^{k-1}\} \quad \text{for } j = k+2,\ldots,f-1 \tag{4.19}$$

with the same lower bound values L_j^{k-1} for $j = k+2,\ldots,f-1$ as in (4.13).

Again, we define a measure for the possible movements of jobs scheduled after job k, which is given by

$$\Delta_k^{(k)\text{succ}} := \max_{\{j | k\to j\in C\}} \{S_k^{(k)} + p_k + l_{kj} - \tilde{S}_j^{\pi}\}.$$

Furthermore, we define measures for the movements of
- the machine successor c of f by $\Delta_c^{(k)} := S_c^{(k)} - \bar{S}_c$, and
- the conjunctive successor b of f on the critical path by $\Delta_b^{(k)} := S_f^{(k)} + p_f + l_{fb} - S_b$.

Analogously to the preceding propositions the following proposition can be proved:

Proposition 4. *Let* $\Delta := \max\{\Delta_c^{(k)}, \Delta_b^{(k)}, \Delta_k^{(k)\text{succ}}\}$. *Then the objective value of the permutation* $\pi^{(k)} = endshift_k(\pi)$ *can be estimated as follows*:
(a) *if* $\Delta \geqslant 0$ *holds, we have* $C_{\max}(\pi') = C_{\max}(\pi) + \Delta$, *and*
(b) *if* $\Delta < 0$ *holds, we have* $C_{\max}(\pi) \geqslant C_{\max}(\pi') \geqslant C_{\max}(\pi) - |\Delta|$.

Contrary to the first two operators in the process of evaluating an *endshift*$_k$-neighbor no information from the evaluation of the previous neighbor *endshift*$_{k+1}$ can be used. Although all L_h^{k-1}-values in (4.13) can be determined in a pre-processing step in $O(n + |C|)$ time, calculating all minima in (4.13) for a block B requires $O(|B|^2)$ time. Furthermore, the lower bound values $L_c^{(k)}$ in (4.17) for a block B can only be calculated in $O(|B|^2 + |C|)$ time, since in each iteration k we may have to determine the new starting times $S_j^{(k)}$ in (4.19) for $j = k+1, \ldots, f-1$. Thus, the whole process for all blocks B on a critical path P^π needs $O(|C| + \sum_{B \in P^\pi} |B|^2) = O(|C| + n^2)$ time.

Summarizing, the conclusions of Propositions 2–4 and can be used as follows in a local search algorithm. For a given permutation π we are able to determine an estimate Δ^* (the minimum of all considered Δ-values) and a solution $\pi^* \in \mathcal{N}_2(\pi)$ (a permutation for which Δ equals Δ^*) by spending $O(n + |C|)$ time for all *api*- and *rshift*-neighbors and $O(n^2 + |C|)$ time for all *endshift*-neighbors with the following meaning:
- if $\Delta^* \geqslant 0$ holds, we know that π^* is the best neighbor of π in $\mathcal{N}_2(\pi)$ and that Δ^* is the amount by which π^* is worse than π,
- if $\Delta^* < 0$ holds, we know that π^* is not worse than π and that Δ^* defines an upper bound for the improvement of the best neighbor of π in $\mathcal{N}_2(\pi)$.

Based on these estimates a solution is chosen from the set of neighbors as a starting solution for the next iteration (usually the permutation π^*).

The neighborhoods introduced in this section and, thus, also the calculation of the presented estimates for the neighbors depend on the chosen critical path P^π for the solution S^π associated with the current solution π. However, in general for a given permutation π many critical paths P^π may exist. In the following, we discuss some aspects about choosing a critical path P^π. We will show that we can always choose a critical path where all jobs on it belong to blocks or a path without any blocks.

Proposition 5. *For each permutation* π *a critical path* $P^\pi = (0 = i_0, i_1, \ldots, i_k,\ i_{k+1} = n+1)$ *exists with the property that for no subsequence* $(i_{\lambda-1}, i_\lambda, i_{\lambda+1})$ *of* P^π *both conjunctions* $i_{\lambda-1} \to i_\lambda \in C$ *and* $i_\lambda \to i_{\lambda+1} \in C$ *exist.*

Proof. Let $P^\pi = (0 = i_0, i_1, \ldots, i_k, i_{k+1} = n+1)$ be a critical path with $i_{\lambda-1} \to i_\lambda \in C$ and $i_\lambda \to i_{\lambda+1} \in C$ for an index $\lambda \in \{1, \ldots, k\}$. Then also the arc $i_{\lambda-1} \to i_{\lambda+1} \in C$ exists and $l_{i_{\lambda-1} i_{\lambda+1}} \geqslant l_{i_{\lambda-1} i_\lambda} + p_{i_\lambda} + l_{i_\lambda i_{\lambda+1}}$ holds, since we assume that the l_{ij}-distances are transitively

adjusted. Since P^π is a longest path, we must have equality $l_{i_{\lambda-1}i_{\lambda+1}} = l_{i_{\lambda-1}i_\lambda} + p_{i_\lambda} + l_{i_\lambda i_{\lambda+1}}$. Thus, we may replace P^π by $(i_0, i_1, \ldots, i_{\lambda-1}, i_{\lambda+1}, \ldots, i_k, i_{k+1})$. Repeating this process yields a critical path with the requested property. □

Note that if a critical path P^π without any blocks exists, we have $P^\pi = (0, n+1)$ and the permutation π is optimal, since $C_{\max}(\pi) = l_{0,n+1}$ equals a trivial lower bound for the optimal makespan. Thus, we may assume that all jobs on the chosen critical path belong to some block.

The following proposition gives another possible restriction on the critical paths. It can be proved in a similar way as Proposition 5.

Proposition 6. *For each permutation π a critical path $P^\pi = (0 = i_0, i_1, \ldots, i_k,\ i_{k+1} = n+1)$ exists with the property that for each pair of jobs i, j which belong to the same block and which are linked by a conjunction $i \to j \in C$ the inequality $S_i^\pi + p_i + l_{ij} < S_j^\pi$ holds.*

As a consequence of this proposition we may assume that no starting time of a job in a block is determined by a conjunctive predecessor in the same block. If this situation were to occur (i.e. in a block $S_i^\pi + p_i + l_{ij} = S_j^\pi$ holds for a conjunction $i \to j \in C$), all changes in the subsequence between i and j would not improve $C_{\max}(\pi)$ since the length of the critical path containing the arc $i \xrightarrow{l_{ij}} j \in C$ is not shortened. Thus, by choosing the critical path according to Proposition 6, some non-improving solutions are not considered as candidates for neighbored permutations.

After having calculated the starting times S_j^π for a permutation π according to (2.3), a special critical path fulfilling the properties of Propositions 5 and 6 denoted by CP^π can be determined by the following backward calculation: starting with $j := n+1$ in each step among the jobs i with $S_i^\pi + d_{ij}^\pi = S_j^\pi$ the job with the smallest starting time S_i^π is chosen. After adding the arc (i, j) to CP^π the process is repeated with $j := i$ in the same way until the starting job $i = 0$ is reached.

CP^π has the property that for all jobs $j \in \{i_{b+1}, \ldots, i_{f-1}\}$ of any block $(i_b, \ldots, i_f)$ on CP^π no job i exists with $S_i^\pi + p_i + l_{ij} = S_j^\pi$. This implies $S_j^\pi > L_j^\pi$, i.e. the starting times of all jobs different to the first in any block are determined by their machine predecessors.

5. A tabu search algorithm

In this section we describe a tabu search algorithm for the single-machine problem which is based on the concepts of the previous sections.

The *tabu search* method is a metaheuristic approach which was designed by Glover [6,7]. In each iteration of this local search method the current solution is usually replaced by the best solution in its neighborhood. Due to the fact that (contrary to the iterative improvement method) also non-improving solutions are accepted, it is possible

to leave local minima during the search process. In order to avoid cycling a so-called *tabu list TL* is used in which typical properties (attributes) of the visited solutions are stored. All solutions having one of the properties stored in the tabu list are declared as tabu and excluded from the search space. The properties are chosen such that a visited solution will not be reached again as long as the corresponding property is in the tabu list. A disadvantage of this procedure is that solutions which have never been visited may be also forbidden by the tabu list. To overcome this difficulty so-called *aspiration criteria* are used which allow the acceptance of neighbors even if they are forbidden due to the tabu status. For example, solutions which improve the best solution found so far should always be accepted.

Besides the tabu list, which has the function of a short-term memory, often also a long-term memory is kept which is used for *diversification*. In this long-term memory promising solutions which are visited during the search process are stored. If within the search process the best solution found so far is not improved in a certain number of iterations, the search process is stopped and restarted with a solution from the long-term memory (diversification). The whole tabu search procedure stops if a maximal number of restarts has been completed.

In the following, we will describe how these general concepts are applied in our tabu search algorithm for the single-machine problem. We use the neighborhood $\mathcal{N}_2$ defined by the operators api_k, $rshift_k$, and $endshift_k$. The properties stored in the tabu list depend on the current solution and the operator which will be applied to it.

More precisely, if an operator is applied to a block $B:=(i_1, i_2, \ldots, i_{k-1}, i_k, i_{k+1}, \ldots, i_f)$ on a critical path P^π associated with a permutation π, besides the objective value $C_{\max}(\pi)$ we store a pair of jobs as an attribute, which characterizes the order in π before applying the operator:

- for the operator api_k we use the pair of interchanged jobs as an attribute, i.e. we store (i_k, i_{k+1}),
- for the operator $rshift_k$ we use the shifted job i_1 and its new predecessor (an old successor) i_k as an attribute, i.e. we store (i_1, i_k), and
- for the operator $endshift_k$ we use the shifted job i_k and its old immediate successor i_{k+1} as an attribute, i.e. we store (i_k, i_{k+1}).

A neighbored solution $\pi' \in \mathcal{N}_2(\pi)$ of a permutation π is defined as tabu if π' results from π by reconstructing the order of a pair of jobs belonging to the tabu list. More specifically, π' is tabu if

- $\pi' = api_k(\pi)$ and the pair (i_{k+1}, i_k) of interchanged jobs is contained in the tabu list,
- $\pi' = rshift_k(\pi)$ and a pair (i_λ, i_1) with $\lambda \in \{2, \ldots, k\}$ is contained in the tabu list, or
- $\pi' = endshift_k(\pi)$ and a pair (i_λ, i_k) with $\lambda \in \{k+1, \ldots, f\}$ is contained in the tabu list.

A solution π' satisfies the aspiration criterion if its objective value is smaller than the objective values of all solutions in the tabu list which declare the new solution tabu. Since we do not calculate the correct objective values for all neighbored solutions but only estimates, we cannot apply this approach directly. However, a straightforward adaption is to test the aspiration criteria based on the estimates. If Δ denotes the

estimate for the operator we want to apply to π, instead of using the correct objective value $C_{\max}(\pi')$ we test the condition $C_{\max}(\pi) + \Delta < C_{\max}(\hat{\pi})$. If $\Delta > 0$ holds, the use of the estimate does not change the situation since in this case the estimate equals the correct objective value. In the other case, where $\Delta \leqslant 0$, using the estimate may lead to another decision than using the correct value since the estimate only defines an upper bound for the improvement.

Following Dell' Amico and Trubian [3] we use a variable length $|TL|$ for the tabu list (a so-called dynamic tabu list) according to the following rules:

- If in some iteration a solution π is found which improves the best solution found so far, we empty the tabu list (by defining $|TL|:=0$) since we know that this solution has never been visited before.
- If we are in an improving phase of the search (i.e. the objective value of the current solution is better than the value of the previous iteration) and the length of the tabu list is greater than a certain constant $TL_{\min}$, we decrease the length by one.
- If we are not in an improving phase of the search and the length of the tabu list is smaller than a certain constant $TL_{\max}$, we increase the length by one.

If the tabu list contains a large number of solutions, it may be time consuming to test whether a solution is tabu or not. Therefore, instead of a tabu list we use a tabu matrix TM of size $n \times n$. If an operator is applied to a permutation π in iteration h, we store both the objective value $C_{\max}(\pi)$ and the number h in the entry belonging to the corresponding pair of jobs (i, j), i.e. we set $TM(i, j):=(C_{\max}(\pi), h)$. The matrix representation may use more memory (n^2) than the tabu list, but the test whether a solution is tabu or not can always be done in time linear in the number of jobs interchanged by the operator.

In the process of choosing a neighbored solution $\pi' \in \mathcal{N}_2(\pi)$ of π we use the "best-fit"-strategy, i.e. in each iteration we chose a non-tabu neighbor π' with minimal $C_{\max}(\pi')$-value (or with minimal estimate Δ). First tests have shown that evaluating the complete neighborhood in each iteration is time consuming. Therefore, we partition the neighborhood $\mathcal{N}_2(\pi)$ into three different subsets $\mathcal{N}_2(\pi) = \mathcal{N}_2^{api}(\pi) \cup \mathcal{N}_2^{rshift}(\pi) \cup \mathcal{N}_2^{endshift}(\pi)$ according to the three types of operators. Instead of evaluating all neighbors from the set $\mathcal{N}_2(\pi)$ in each iteration, we restrict the choice of the next neighbor to one of these subsets. In this way the evaluation of the neighborhood can be further accelerated and some additional strategies may be used to intensify or diversify the search process. We guide the choice of the next operator according to the following types of strategies:

- depending on the iteration number: the choice of the next operator type is only determined by the current number of iterations, e.g. we may always repeat the sequence $api, rshift, api, endshift, api, \ldots$,
- depending on the objective values: the operator type is changed if the objective value is not improved for a certain number of iterations or when all neighbors belonging to the last operator type are tabu.

Initially, we calculate feasible schedules by a simple priority-based heuristic (for details see [10]). Based on certain priority rules iteratively a job from the set of all

available jobs is chosen to be scheduled next. In order to choose such a job we tested four different possibilities: a job with a smallest head, with a largest tail, with a largest number of successors, or a job which leads to the earliest next decision point. Each of the four resulting permutations is used as a starting solution for the tabu search procedure.

6. Computational results

In this section we report some computational results for the described tabu search algorithm. We implemented the procedure in C and tested it on a Sun Ultra 2 work station (167 MHz) with operating system Solaris 2.5 and 320 MB general storage.

Since no test instances for the job-shop problem with transportation times were available from the literature, we modified $m \times n$ benchmark problems for the classical job-shop problem, where m denotes the number of machines and n the number of jobs. We used the well-known 6×6 and 10×10 instances $P1$ and $P2$ from Muth and Thompson [13]. In both instances the number of operations per job is equal to the number of machines (i.e. $n_j = m$ for $j = 1, \ldots, n$) and each job is processed on each machine exactly once. The processing times of the operations in the instance $P1$ are from the interval $[1, 10]$ and in $P2$ from the interval $[1, 100]$. Various test instances were obtained by adding transportation and empty moving times with different characteristics.

For the transportation times t_{jkl} we distinguished the following four cases: job- and machine-dependent times t_{jkl} randomly generated from the interval $[1, 10]$ (adjusted in such a way that the triangle inequality holds), job-independent transportation times t_{kl} analogously to the first case, job-independent transportation times $t_{kl} = D|k - l|$ with different values D (proportional to the distance between the corresponding machines, which are assumed to be ordered in a single line), and constant transportation times $t_{jkl} = T$.

Analogously, we distinguished the following three cases for the empty moving times: randomly generated values t'_{kl}, values $t'_{kl} = d|k - l|$ depending on the machine distances and constant times $t'_{kl} = t$. Test instances for the single-machine problem were generated from these job-shop instances with transportation times. As described in Section 1, this problem arises for a single robot when the machine orders are given. In order to generate some instances where the optimal values are known, we calculated the optimal schedule for the job-shop problem with transportation times and a single robot where all empty moving times are zero. Since this problem is a classical job-shop problem (where the robot is considered as an additional machine), we were able to use the branch-and-bound algorithm of Brucker et al. [2] to calculate an optimal schedule. We then ignored the calculated order of transport operations for the robot and used only the m machine orders as input for the single-machine problem. In this way we obtained several instances with $6 \times 5 = 30$ or $10 \times 9 = 90$ transport operations (arising from the 6×6 and 10×10 job-shop instances $P1$ and $P2$, respectively). Since the processing times in instance $P2$ are very large (from the interval $[1, 100]$), the time horizon for

Table 1
Comparison of the quality of the heuristic procedures

		UB^{pr}	II	TS_1	TS_2	TS
30	$\Delta(LB)_{av}$	12.8	8.3	1.3	1.2	0.6
	$\Delta(LB)_{max}$	27.4	17.1	6.1	4.3	3.0
90	$\Delta(LB)_{av}$	20.9	13.8	7.3	7.1	6.4
	$\Delta(LB)_{max}$	43.4	30.9	20.1	20.5	16.7

the modified instances is often also very large ($C_{max} \in [1000, 3000]$). Therefore, we also generated some instances in which the processing times are scaled by a factor $0 < f < 1$, i.e. we replaced the processing times p_{ij} by $\lceil f \times p_{ij} \rceil$.

After some first computational tests with a large test set we tried identifying interesting instances which are not easy to solve (i.e. for which priority rules or simple iterative improvement procedures did not get a solution value which equals a lower bound). In the following, we will only report results for these instances (30 with 30 transport operations and 10 instances with 90 transport operations).

Several parameters may be varied concerning the tabu search algorithm presented in Section 5. We tested 13 strategies for choosing the next operator which is applied to the current solution. Since no strategy clearly outperformed the others and the computation times were small, we decided to include them all into the algorithm (i.e. we start with each starting solution 13 times). We performed two restarts from the best solution found so far (where additionally the strategy was changed). Finally, we tested different parameter values for the minimal and maximal tabu list lengths (TL_{min} and TL_{max}), and the maximal number of iterations after which the search process is stopped when the best solution is not improved within these iterations (*maxiter*). From all tested versions we chose the two versions TS_1 with $TL_{min} = 3$, $TL_{max} = 16$, $maxiter = 500$, and TS_2 with $TL_{min} = 4$, $TL_{max} = 14$, $maxiter = 1500$ (cf. also [10]).

In order to estimate the quality of these procedures we compared the results with the best objective value UB^{pr} calculated with the four priority rules, with the objective value II obtained by an iterative improvement procedure (started with each of the four initial solutions once and stopped when no improving neighbor of the current solution exists) and with the best objective value found by one of the tested versions of the tabu search algorithms (column TS). Furthermore, we calculated lower bounds LB for the instances using the techniques of constraint propagation and linear programming (cf. [10]).

For each heuristic solution value UB we determined the relative deviation $\Delta(LB) = (UB - LB)/LB$ from the lower bound value LB. In Table 1 we report the average and maximal relative deviations $\Delta(LB)_{av}$ and $\Delta(LB)_{max}$ (in %). All instances and detailed results for them can be found on the web-site

`http://www.mathematik.uni-osnabrueck.de/research/OR/software.html`.

Table 1 shows that the quality of the tabu search algorithm is quite good. While for the instances with 30 operations the mean relative deviation from the lower bound

Table 2
Comparison of the computational times for the two implementations

	30				90			
	TS_1^f	TS_1^s	TS_2^f	TS_2^s	TS_1^f	TS_1^s	TS_2^f	TS_2^s
Average CPU	0:37	1:03	1:47	2:57	2:09	6:34	5:43	17:30
Maximal CPU	0:43	1:17	2:07	3:39	2:59	8:58	8:09	24:30

is $\Delta(LB)_{av} = 0.6\%$, for the instances with 90 operations we have $\Delta(LB)_{av} = 6.4\%$ (for these larger instances only weaker lower bounds based on constraint propagation could be calculated). On average the tabu search algorithm improved the best starting solution by approximately 10%.

In our implementation of the tabu search algorithm we used the techniques to accelerate the evaluation of neighbored solutions as described in Section 4. From the theoretical point of view, the complexity of the evaluation process was reduced approximately by the factor $O(n)$. In order to study the improvement from the practical point of view, we implemented a second version of the tabu search procedure in which for each neighbor the correct objective value is determined. With respect to the quality of the calculated solutions no significant differences could be observed, but the computational times increased. A comparison of the average and maximal CPU times (in min:s) for the fast implementations TS_1^f and TS_2^f, and the slow versions TS_1^s and TS_2^s for the instances with 30 and 90 operations can be found in Table 2. The computational times could be reduced up to the factor 3 with our fast versions, which shows that our accelerations are also practically worthwhile.

7. Concluding remarks

We developed a tabu search procedure for a generalization of the traveling salesman problem with time windows where additionally generalized precedence constraints (minimal time-lags between certain pairs of nodes) are given. This problem arises as an important subproblem in a job-shop environment with a single transport robot.

The computational results show that the tabu search procedure calculates good upper bounds in a short amount of time. The presented algorithm is used as a subroutine in a two-stage local search algorithm for the job-shop problem with a single transport robot [10]. While in the outer stage machine orders are changed, in the inner stage the robot solution is modified by our tabu search procedure. First test results of our two-stage procedure compared with results obtained by an one-stage local search procedure are promising. Especially, for instances in which the transportation stage is not dominating (i.e. scheduling the machines and the robot are both important), the two-stage procedure is superior to the one-stage algorithm. Additional improvements and combinations of these two procedures are the topic of further research.

References

[1] N. Ascheuer, Hamiltonian path problems in the on-line optimization of flexible manufacturing systems, Ph.D. Thesis, Technische Universität Berlin, 1995.
[2] P. Brucker, B. Jurisch, B. Sievers, A branch & bound algorithm for the job-shop problem, Discrete Appl. Math. 49 (1994) 107–127.
[3] M. Dell' Amico, M. Trubian, Applying tabu search to the job-shop scheduling problem, Ann. Oper. Res. 41 (1993) 231–252.
[4] J. Desrosiers, Y. Dumas, M. Solomon, F. Soumis, Time constrained routing and scheduling, in: M.O. Ball, et al. (Eds.), Handbooks in Operations Research and Management Science, Vol. 8, Elsevier, Amsterdam, 1995, pp. 35–139.
[5] M. Gendreau, A. Hertz, G. Laporte, M. Stan, A generalized insertion heuristic for the traveling salesman problem with time windows, Oper. Res. 46 (1998) 330–335.
[6] F. Glover, Tabu search, Part I, ORSA J. Comput. 1 (1989) 190–206.
[7] F. Glover, Tabu search, Part II, ORSA J. Comput. 2 (1990) 4–32.
[8] J. Grabowski, E. Nowicki, S. Zdrzalka, A block approach for single machine scheduling with release dates and due dates, European J. Oper. Res. 26 (1986) 278–285.
[9] R. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy-Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, Ann. Discrete Math. 5 (1979) 287–326.
[10] S. Knust, Shop-scheduling problems with transportation, Ph.D. Thesis, Fachbereich Mathematik/Informatik Universität Osnabrück, 1999.
[11] P.J.M. Laarhoven, E.H.L. van, Aarts, J.K. Lenstra, Job shop scheduling by simulated annealing, Oper. Res. 40 (1992) 113–125.
[12] A. Mingozzi, L. Bianco, S. Ricciardelli, Dynamic programming strategies for the traveling salesman problem with time window and precedence constraints, Oper. Res. 45 (1997) 365–377.
[13] J.F. Muth, G.L. Thompson, Industrial Scheduling, Prentice-Hall, Englewood Cliffs, NJ, 1963.
[14] E. Nowicki, C. Smutnicki, A fast tabu search algorithm for the job shop problem, Management Sci. 42 (1996) 797–813.
[15] E. Nowicki, C. Smutnicki, A fast tabu search algorithm for the permutation flow-shop problem, European J. Oper. Res. 91 (1996) 160–175.