ELSEVIER

Discrete Applied Mathematics 112 (2001) 179–197

DISCRETE APPLIED MATHEMATICS

# Local search algorithms for a single-machine scheduling problem with positive and negative time-lags

Johann Hurink[a,*], Jens Keuchel[b]

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

[b] *Universität Mannheim, Lehrstuhl fuer Bildverarbeitung, Mustererkennung und Computergrafik, L13,17, D-68131 Mannheim, Germany*

Received 1 April 1998; revised 1 June 1999; accepted 1 August 2000

**Abstract**

Positive and negative time-lags are general timing restrictions between the starting times of jobs which have been introduced by Roy (C.R. Acad. Sci., 1959, T.248) in connection with the Metra Potential Method. Although very powerful, these relations have been considered only seldom in the literature since already for a single-machine problem with positive and negative time-lags the problem of finding a feasible solution is $\mathcal{NP}$-complete. In this paper a local search approach for a single-machine scheduling problem with positive and negative time-lags and the objective to minimize the makespan is presented. Since the existence of a feasible initial solution for starting the search can not be guaranteed, infeasible solutions are incorporated into the search process. Computational results based on instances resulting from shop problems are reported. 

*Keywords*: Time-lags; Tabu search; Scheduling

## 1. Introduction

Almost 40 years ago, Roy [21] introduced a simple but powerful tool to model project networks. For his metra potential method (MPM) he used two types of relations between activities. The first type expresses that between the starting points of two activities there must be a minimal (positive) time-lag. The second stipulates that there must be a maximal (negative) time-lag. Although these time-lags were extremely useful for modeling projects, they were hardly used over the ensuing three decades. Results of Bartusch et al. [3] and Brucker et al. [5] may explain this at first sight surprising fact. Whereas Bartusch et al. [3] show that arbitrary time-lags may be used to model release

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

0166-218X/01/$ - see front matter 
PII: S0166-218X(00)00315-2

times and deadlines of activities and time-dependent resource availability, Brucker et al. [5] show that many complex scheduling problems like general shop problems, problems with multi-processor tasks, problems with multi-purpose machines, and problems with changeover times can be reduced to a single-machine problem with arbitrary time-lags. As a consequence, already for very simple resource-constrained scheduling problems with arbitrary time-lags between the activities, the question whether or not a feasible schedule exists is $\mathcal{N}$P-complete.

In recent years, arbitrary time-lags were reanimated again. Several papers developing heuristics or branch-and-bound methods occurred mainly in the area of project scheduling [3,8,10,12,18,24]. Besides this, some research was carried out for single-machine problems. Balas et al. [4] propose an efficient adaption of Carliers algorithm for the single-machine problem with heads and tails $1|r_i, q_i|C_{\max}$ [9] to the case with positive time-lags. Wikum et al. [23] investigated the complexity of single-machine problems with minimal and/or maximal distances between jobs. Their main results show that scheduling problems of this type with a very simple structure are already $\mathcal{N}$P-hard. Furthermore, Brucker et al. [5] provide a branch-and-bound approach for solving a single-machine problem with arbitrary time-lags.

In this paper we will consider the application of local search to the single-machine problem with positive and negative time-lags and the objective to minimize the makespan. Based on the reductions presented by Brucker et al. [5] such heuristics may be seen as a type of general purpose heuristics since in principle they can also be used to solve various other types of scheduling problems. A main obstacle for developing local search heuristics for the considered problem is the fact that the determination of a starting solution is already a hard problem. We will overcome this by incorporating infeasible solutions in the search process. More precisely, we will consider all sequences of jobs which are admissible for the positive time-lags only as solutions regardless of whether a feasible schedule respecting this sequence exists or not.

The paper is organized as follows. In Section 2 we will give a formal definition of the considered problem and present a network representation for the instances. Afterwards, in Section 3 we will formulate the problem of finding the best schedule for a given sequence of the jobs (if such a schedule exists) as a longest path problem and develop an efficient method to determine such a schedule. Section 4 will be devoted to the description of local search methods. A basic method will be a tabu search approach which starts with an 'infeasible' sequence of the jobs and which tries to guide the search to a feasible sequence. This basic routine will be packed in an outer method which has two features. Whenever a feasible solution has been detected through the basic routine, the given instance will be changed in such a way that the current solution becomes infeasible and only improving solutions of the current solution correspond to feasible solutions of the changed instance. After the change of the instance the basic routine starts again with the current solution, which is now infeasible and tries to find a feasible and thus improving solution. If after a certain amount of time the basic routine is not able to find a feasible schedule, the outer method has a second task. It applies diversification operators to the current solution in order to reach another area of the

search space which perhaps contains a feasible solution. In Section 5 computational results using instances which are generated by reductions of benchmark instances for job-shop and open-shop problems are presented. The paper ends with some concluding remarks.

## 2. The problem

In this section we will give a formal definition of the considered problem and introduce some useful notations.

Let $J=\{1,\ldots,n\}$ be a set of $n$ jobs with processing times $p_1,\ldots,p_n$ to be scheduled without preemption on a single machine. Furthermore, we have a set of relations called *time-lags* of the form

$$S_i + d_{ij} \leqslant S_j \quad \text{for all } (i,j) \in R, \tag{1}$$

where $R \subseteq J \times J$ and $S_i$ denotes the starting time of job $i$, $i=1,\ldots,n$ (all data are assumed to be integral). If $d_{ij} \geqslant 0$, then (1) means that job $j$ cannot be started earlier than $d_{ij}$ time units after the starting time of job $i$ (minimal time-lag). On the other hand, if $d_{ij} < 0$, then job $j$ cannot be started earlier than $|d_{ij}|$ time units before the starting time of job $i$ (maximal time-lag).

Since we consider non-preemptive solutions, a schedule of the jobs is uniquely defined by a vector of starting times $S_1,\ldots,S_n$ for the jobs. Such a schedule $S=(S_1,\ldots,S_n)$ is called a *feasible schedule* if

- for all $i \neq j$ the intervals $[S_i, S_i + p_i[$ and $[S_j, S_j + p_j[$ are disjoint and
- the time-lags (1) are satisfied.

The goal is to find a feasible schedule which minimizes the makespan $C_{\max} = \max_{i=1,\ldots,n} \{S_i + p_i\}$ if such a schedule exists. As mentioned in the introduction, for this problem the question of whether or not a feasible solution exists, is already $\mathcal{NP}$-complete (see Bartusch et al. [3] and Brucker et al. [5]).

For convenience of notation, we will add two dummy jobs 0 and $n+1$ with processing times zero to the job set $J$. Job 0 is a starting job which must be "processed" before all other jobs $i$ (i.e. $d_{0i}=0$) and job $n+1$ is a final job which must be "processed" after all other jobs $i$ (i.e. $d_{i,n+1} = p_i$). Furthermore, we may assume that for each job-pair $(i,j) \in (J \cup \{0, n+1\})^2$, $i \neq j$, a relation $S_i + d_{ij} \leqslant S_j$ is defined (if this is not the case, we introduce a redundant relation with $d_{ij} = -\infty$).

To represent the time-lags $R$, we will make use of a network $N(R)$ which is defined as follows:

- the vertex set $V=\{0,1,\ldots,n,n+1\}$ consists of all jobs $1,\ldots,n$ and the two dummy jobs 0 and $n+1$;
- for each pair $(i,j)$ of jobs $i,j=0,\ldots,n+1$ with $i \neq j$ there is an arc $(i,j)$ of length $d_{ij}$.

If we calculate in this network $N(R)$ the longest path lengths $l(i)$ from the starting job 0 to all jobs $i$, $i=1,\ldots,n+1$ (if this is not possible since $N(R)$ contains a positive

cycle, the given instance has no feasible schedule), the schedule where job $i$ starts at time $l(i)$, $i = 1,\ldots,n$, respects the time-lags, but is not necessarily feasible since jobs may overlap.

Before discussing methods of how to solve these conflicts of overlapping jobs, we would like to mention two techniques to sharpen the given time-lags (for details and further methods, see [3,5]). Firstly, we may replace the length $d_{ij}$ of a given time-lag by the length of a longest path from $i$ to $j$ in $N(R)$. Furthermore, if $d_{ij} \in ]-p_j, p_i[$, we may replace $d_{ij}$ by $p_i$ since job $j$ cannot be scheduled before job $i$ and $d_{ij} > 0$ implies $d_{ij} \geqslant p_i$. In the following we will assume that the given data of the problem has been modified according to these two rules.

## 3. Fixed job sequences

In the previous section we have noticed that a main scheduling decision for the considered problem is to determine an order in which the jobs will be scheduled. However, in contrast to most other non-preemptive one-machine scheduling problems, a fixed decision for all pairs of jobs (i.e. a fixed sequence for the jobs) does not directly lead to a corresponding schedule, since due to negative time-lags (maximal distances) we cannot apply simple list scheduling heuristics. In the remaining part of this section we will present an efficient method to determine a best schedule of the jobs which respects the given time-lags and a fixed sequence of the jobs if such a schedule exists.

Let $\pi$ be an arbitrary sequence of the jobs. In each schedule $(S_i)_{i=1}^n$ respecting the sequence $\pi$ we must have

$$S_{\pi(i)} + p_{\pi(i)} \leqslant S_{\pi(i+1)} \quad \text{for all } i \in \{1,\ldots,n-1\}, \tag{2}$$

i.e. between the jobs $\pi(i)$ and $\pi(i+1)$ we have a time-lag of length $p_{\pi(i)}$. Thus, each feasible schedule respecting $\pi$ has to respect the time-lags $d_{ij}^{\pi}$ defined by

$$d_{ij}^{\pi} = \begin{cases} \max\{p_i, d_{ij}\} & \text{if } i = \pi(k),\ j = \pi(k+1), \\ d_{ij} & \text{otherwise.} \end{cases}$$

By $N(R^{\pi})$ we will denote the network $N(R)$, where the arc lengths are given by the values $d_{ij}^{\pi}$.

To calculate the best schedule respecting $\pi$ we may calculate in the network $N(R^{\pi})$ the lengths $l^{\pi}(i)$ of the longest paths from the starting job 0 to all jobs $i$, $i=1,\ldots,n+1$. If the network contains a positive cycle, clearly no feasible schedule respecting $\pi$ exists. Otherwise, the schedule where each job $i$ starts at time $l^{\pi}(i)$ is feasible (due to (2) no jobs overlap) and has a makespan of length $C_{\max}^{\pi} = l^{\pi}(n+1)$ which is minimal over all schedules respecting $\pi$.

In principle, the values $l^{\pi}(1),\ldots,l^{\pi}(n+1)$ can be calculated using the Floyd–Warshall algorithm (see, e.g., [2]). The complexity of this algorithm is $O(n^3)$ and in the case where no positive cycle exists, this algorithm really executes $n^3$ steps. Since our goal

is to carry out local search on the set of all possible job sequences, the calculation of the values $l^{\pi}(1),\ldots,l^{\pi}(n+1)$ will be executed very often which will mainly determine the computational time for the local search approaches. Therefore, in the following we will present an alternative longest path method which takes advantage of the structure of the time-lags $d_{ij}^{\pi}$. The resulting method will also have a worst time complexity of $O(n^3)$. However, in the worst case approximately at the most $\frac{1}{4}n^3$ steps are executed and on average the method will stop much earlier. Furthermore, the new method is more suitable than the Floyd–Warshall algorithm to identify a positive cycle in the case where one exists.

Before describing the longest path method, we will first introduce an additional notation. A schedule $S=(S_1,\ldots,S_n)$ is called *partially feasible* w.r.t. $\pi$ if it respects all minimal time-lags defined by $d_{ij}^{\pi}$. Our method will iteratively transform partially feasible schedules into partially feasible schedules. During this process the starting times of the jobs will not decrease.

**Algorithm** *CalculateSchedule* $(\pi)$
*calculate an initial partially feasible schedule S w.r.t.* $\pi$;
**WHILE** *S is not feasible* **DO**
$S$:=*improve*$(S,\pi)$;

An initial partially feasible schedule is obtained by calculating the longest paths from the starting job 0 to all other jobs considering only the non-negative time-lags defined by the values $d_{ij}^{\pi}$ of the given sequence $\pi$. The procedure *improve*$(S,\pi)$ considers the jobs in order $\pi(1),\ldots,\pi(n)$. If the job $\pi(i)$ violates a time-lag in the current schedule $S$, we increase $S_{\pi(i)}$ to the first point where job $\pi(i)$ respects all time-lags.

**Procedure** *improve*$(S,\pi)$
**FOR** $i=1$ **TO** $n$ **DO**
$S_{\pi(i)}:=\max\{S_j+d_{j,\pi(i)}^{\pi}\mid j=1,\ldots,n;\ j\neq\pi(i)\}$.

Obviously, no starting time $S_i$ is decreased in *improve*$(S,\pi)$ and at the end of the procedure we again have a partially feasible schedule w.r.t. $\pi$.

It remains to show that the algorithm *CalculateSchedule*$(\pi)$ terminates and to estimate its running time.

**Lemma 1.** *If the network* $N(R^{\pi})$ *does not contain a cycle of positive length, the algorithm CalculateSchedule*$(\pi)$ *terminates after at the most* $(n-1)/2$ *iterations of the WHILE-loop.*

**Proof.** The values $S_k$ during the algorithm *CalculateSchedule*$(\pi)$ are always equal to the length of a path $P_k$ from job 0 to job $k$ in $N(R^{\pi})$, $k=1,\ldots,n$. Initially these paths consist of only arcs of positive length.

**Claim 1.** *If in the lth iteration of the WHILE-loop the value* $S_k$ *increases, each path* $P_k$ *corresponding to the new value contains exactly l arcs of negative length.*

**Proof of Claim 1.** We will prove the claim by induction on the order in which the values $S_k$ are increased. Assume that the value $S_i$ has been increased in the $l$th iteration and that $j$ is a job with $S_i = S_j + d^{\pi}_{ji}$. Since the redefinition of $S_i$ in iteration $l-1$ assured that job $i$ fulfilled all time-lags at that moment, $S_j$ must have been increased since that time. If $j$ comes after $i$ in $\pi$, the time-lag $d^{\pi}_{ji}$ is negative and $S_j$ has been increased in iteration $l-1$. Thus, by induction the path corresponding to $S_j$ has exactly $l-1$ arcs of negative length and therefore the path corresponding to $S_i$ has exactly $l$ arcs of negative length. If $j$ comes before $i$ in $\pi$, the time-lag $d^{\pi}_{ji}$ is positive and $S_j$ has been increased in iteration $l$. Thus, by induction the path corresponding to $S_j$ has exactly $l$ arcs of negative length and therefore the path corresponding to $S_i$ also has exactly $l$ arcs of negative length.

Next we will bound the number of arcs of negative length which may occur in a path $P_k$ corresponding to a value $S_k$.

**Claim 2.** *In each path $P_k$ corresponding to a value $S_k$ no two consecutive arcs may have negative lengths.*

**Proof of Claim 2.** If two consecutive arcs $(i,j)$ and $(j,k)$ on a longest path $P_k$ have negative lengths, we have $d^{\pi}_{ij} + d^{\pi}_{jk} = d_{ij} + d_{jk} \leqslant d_{ik} \leqslant d^{\pi}_{ik}$ (the first inequality follows from the assumption that the original values $d_{ij}$ are transitive closed). Thus, in the iteration where the arc $(i,j)$ led to an increase of the value $S_j$, the arc $(i,k)$ ensured that $S_k \geqslant S_i + d^{\pi}_{ik} \geqslant S_i + d^{\pi}_{ij} + d^{\pi}_{jk} = S_j + d^{\pi}_{jk}$. Therefore the arc $(j,k)$ cannot be used to increase $S_k$ before $S_j$ has been changed again, which is a contradiction.

Since we assume that the network $N(R^{\pi})$ has no cycles of positive length, a path $P_k$ in $N(R^{\pi})$ from 0 to a vertex $k$ contains at the most $n$ arcs. Furthermore, on such a path $P_k$, before the first arc with negative length, at least one arc which represents a real minimal time-lag (i.e. $d^{\pi}_{ij} > p_i$) and at least one arc with length $d^{\pi}_{ij} = p_i$ must occur. The second part of the statement follows from the fact that the time-lags given by the instance are assumed to be transitive closed. Thus, by Claim 2 the path $P_k$ has at the most $\lceil (n-2)/2 \rceil \leqslant (n-1)/2$ arcs of negative length. Claim 1 now proves the lemma.

To estimate the running time of the algorithm *CalculateSchedule*$(\pi)$ it remains to estimate the complexity of the procedure *improve*$(S,\pi)$.

**Lemma 2.** *If the network $N(R^{\pi})$ does not contain a cycle of positive length, in the $l$th iteration of the WHILE-loop of the algorithm CalculateSchedule$(\pi)$ at the most $n-2l$ values $S_i$ can be changed by the procedure improve$(S,\pi)$.*

**Proof.** If in the $l$th iteration one value $S_i$ is changed, the network $N(R^{\pi})$ contains at least one longest path $P_k$ from 0 to a vertex $k$ with at least $l$ arcs of negative length. As mentioned in the proof of Lemma 1, each path $P_k$ starts with at least two arcs of

non-negative length and contains no two consecutive arcs with negative lengths. Thus, before the $l$th arc of negative length on $P_k$ at least $2l$ vertices (excluding 0) occur. The longest paths to these vertices contain less than $l$ arcs of negative length and, thus, the corresponding $S_i$-values are already fixed before iteration $l$ (see Claim 1 in the proof of Lemma 1). □

Using these two lemmata we get

**Theorem 3.** *Algorithm CalculateSchedule* $(\pi)$ *can be implemented to run in* $O(n^3)$.

**Proof.** Lemma 1 states that we can terminate the algorithm if after the iteration $(n-1)/2$ the current schedule is not feasible. In this case the network $N(R^\pi)$ contains a positive cycle and the sequence $\pi$ does not represent a feasible solution.

The calculation of the initial solution needs $O(n^2)$ steps. By combining Lemmata 1 and 2 we can bound the number of changes of $S_i$-values by

$$\sum_{l=1}^{(n-1)/2} n-2l = \left(\frac{n-1}{2}\right)^2.$$

If directly after each change of an $S_i$-value, we calculate the influence of the new $S_i$-values together with the time-lags $d_{ij}^\pi$ on the $S_j$-values ($n-1$ steps), we get the stated upper bound on the number of steps. □

If the algorithm terminates with a feasible schedule $S$, it will be of importance for the definition of suitable neighborhoods (see Section 4) to identify the longest paths corresponding to the starting times of the jobs. To be able to reconstruct these paths, a vector *pre* will be introduced, where $pre(i)$ will denote the predecessor of $i$ on the longest path $P_i$ determining the current value of $S_i$. After an initialization depending on the initial partially feasible schedule, these values always have to be updated if in the procedure *improve* $(S,\pi)$ a value $S_i$ is increased.

It remains to consider the case in further detail, where no feasible schedule w.r.t. $\pi$ exists. The algorithm *CalculateSchedule*$(\pi)$ is able to detect infeasibility at two places: If after $\lceil (n-2)/2 \rceil$ iterations the current partially feasible schedule is not feasible (Lemma 1) or if in iteration $l$ more than $n-2l$ values $S_i$ are updated (Lemma 2), no feasible schedule w.r.t. $\pi$ can exist.

In the first case, each violated maximal time-lag $d_{ij}$ in the current schedule $S$ leads in connection with the path $P_i$ determining $S_i$ (since $S_i$ must have been changed in the last iteration, $P_i$ contains at least $n$ arcs) to a positive cycle which can be determined by the use of the vector *pre*. In the second case, it is easy to show that a positive cycle can be determined using the vector *pre* starting with an arbitrary job $i$ for which the $S_i$-value has been changed in the $l$th iteration (for details, see [16]).

Besides these two direct ways of determining positive cycles, it is possible to incorporate some additional tests into the algorithm *CalculateSchedule*$(\pi)$ without increasing the worst-case bound in Theorem 3. These tests are based on the stored values

of the lengths of longest paths between arbitrary vertices. Initially these values $l_{ij}$, $(i,j=0,\ldots,n)$ are defined by the values $d_{ij}^{\pi}$. Whenever a value $S_j$ is increased during the algorithm, it is checked whether or not the time-lag $d_{ij}^{\pi}$, which was responsible for this increase, may be used to increase the values of other longest paths: If $l_{ki}+d_{ij}^{\pi} > l_{kj}$, we increase $l_{kj}$ to $l_{ki}+d_{ij}^{\pi}$ (these updates correspond to a subset of steps which are also executed during the Floyd–Warshall algorithm). To detect positive cycles, we always check if $l_{kj} + l_{jk} > 0$ when a value $l_{kj}$ is increased. If this is the case, the network $N(R^{\pi})$ contains a positive cycle and we may stop. To reconstruct this cycle, we also have to store the predecessors on longest paths. Computational tests have shown that on average the incorporation of these additional tests pays off. This indicates that the additional tests help to detect positive cycles much earlier. On the other hand, the Floyd–Warshall algorithm has been shown to be much slower than our method.

## 4. A local search approach

In this section we will present a local search method for a single-machine problem with positive and negative time-lags. Firstly, we will discuss how we choose the search space and how we define the objective values of the solutions. Subsequently, we will introduce the neighborhoods which form the basis of the local search approach (for an introduction to the basic ideas of local search and neighborhoods, see e.g. Papadimitriou and Steiglitz [20] or Aarts and Lenstra [1]). Finally, we will describe the main structure of the developed local search heuristic.

Since for the single-machine problem with positive and negative time-lags the search for a feasible solution is already $\mathcal{N}$P-hard, we somehow have to deal with infeasible solutions. In our approach we will consider the set of all sequences of the jobs which are compatible with the non-negative time-lags as search space $\mathcal{S}$, i.e.

$$\mathcal{S} = \{\pi = (\pi_1,\ldots,\pi_n) \,|\, \text{if } d_{\pi_i\pi_j} \geqslant 0 \text{ then } i < j\}.$$

In general, this set $\mathcal{S}$ will also contain sequences for which no corresponding feasible schedule exists. This causes problems for defining suitable objective values for the sequences. Whereby for 'feasible' sequences we may define the objective value as the makespan of the best schedule w.r.t. $\pi$ (calculated by the algorithm *CalculateSchedule*($\pi$)), 'infeasible' sequences are harder to tackle. For such sequences $\pi$ we somehow have to measure the degree of infeasibility. To define such a measure, we will make use of the starting times of a partial feasible schedule $S$ w.r.t. $\pi$. More precisely, we will apply the algorithm *CalculateSchedule*($\pi$) and use the partial feasible schedule $S$ with which we started the iteration in which the infeasibility was detected. The objective value of $\pi$ will now be defined by a measure on the infeasibility of this partial feasible schedule $S$. Possible measures are:

- $Z_1$: number of violated time-lags in $S$.
- $Z_2$: value of the maximal violation of a time-lag in $S$.
- $Z_3$: sum of the violations of time-lags in $S$.

At this point, it is important to note that these measures are not 'sharp', since they crucially depend on the time at which we stop the algorithm *CalculateSchedule*($\pi$). It may occur that the partial feasible schedule, which we would achieve after one additional iteration, has a totally different measure of infeasibility.

Although we have the possibility of calculating the objective values for feasible and infeasible sequences with the same routine, it is rather inconvenient to have two different types of measures. To avoid this, we will artificially make all considered sequences infeasible. Whenever a feasible solution is detected, we will modify the given instance by increasing the time-lag $d_{n+1,0}$ to the value $-(K-1)$, where $K$ is the makespan of the feasible solution. By bounding the distance between the starting job 0 and the final job $n+1$ by $K-1$, only schedules with a makespan smaller than $K$ remain feasible. Thus, the current schedule becomes infeasible and the search for a feasible solution corresponds to the search for a solution which improves the best solution found so far.

Based on the above-mentioned technique to always modify the given instance when a feasible schedule has been detected, we may define the neighborhoods for the local search method on the base of infeasible sequences. Since the neighborhoods mainly determine navigation through the search space, their definition should be based on the goal to find feasible sequences. In the following theorem we will prove that this goal can only be achieved by specific changes to the given infeasible solution.

To state the theorem we first have to introduce a special decomposition of cycles. Let $C=(i_1,\ldots,i_k,i_{k+1}=i_1)$ be a cycle in a network $N(R^\pi)$ (w.l.o.g. we will assume $d^\pi_{i_k i_1}=d_{i_k i_1}$). A subsequence $B=(i_j,\ldots,i_l)$ of $C$ is called a *block* if the lengths of the arcs within $B$ are determined by the sequence $\pi$, and if the lengths of the two arcs which combine $B$ with the rest of the cycle are determined by original time-lags, i.e. $d^\pi_{i_r i_{r+1}}=p_{i_r}>d_{i_r i_{r+1}}$, $r=j,\ldots,l-1$, $d^\pi_{i_{j-1}i_j}=d_{i_{j-1}i_j}$, and $d^\pi_{i_l i_{l+1}}=d_{i_l i_{l+1}}$. It is straightforward to see that each cycle $C$ decomposes uniquely into a sequence of blocks. Based on this decomposition of cycles into blocks we can prove:

**Theorem 4.** *Let $\pi$ be a given infeasible sequence of the jobs, let $C=(i_1,\ldots,i_k, i_{k+1}=i_1)$ be a cycle in $N(R^\pi)$ with positive length, and let $B_1,\ldots,B_r$ be the blocks of $C$.*

*If, for some sequence $\pi'$, a corresponding feasible schedule exists, one of the following two properties holds*:

- *in $\pi'$ at least one job of a block $B_i$ different to the first job in $B_i$ is sequenced before all other jobs in $B_i$ or*
- *in $\pi'$ at least one job of a block $B_i$ different to the last job in $B_i$ is sequenced after all other jobs in $B_i$.*

**Proof.** Let $f(B_i)$ ($l(B_i)$) be the first (last) job of block $B_i$, $i=1,\ldots,r$. Assume that for a feasible sequence $\pi'$ none of the stated properties holds. Thus, all jobs of $B_i$ are sequenced between $f(B_i)$ and $l(B_i)$ in the sequence $\pi'$. This implies that the length of a longest path from $f(B_i)$ to $l(B_i)$ in $N(R^{\pi'})$ is at least $\sum_{i\in B_i\setminus\{l(B_i)\}} p_i$. Since, furthermore, we have $d^{\pi'}_{l(B_i)f(B_{i+1})}\geqslant d^\pi_{l(B_i)f(B_{i+1})}$, $i=1,\ldots,r$ ($f(B_{r+1})=f(B_1)$), the length

of $C$ in $N(R^{\pi'})$ is greater than or equal to the length of $C$ in $N(R^{\pi})$ and, thus, positive. This implies that no feasible schedule w.r.t. $\pi'$ exists, which is a contradiction. □

Properties similar to those stated in the above theorem have been developed for shop problems and are used as a basis for branch-and-bound methods (see, e.g. Brucker et al. [7,6]) as well as for local search heuristics (see, e.g. Dell'Amico and Trubian [11], Hurink et al. [15], Nowicki and Smutnicki [19]). The basis of these results is given in a work by Grabowski et al. [14], in which the notation of a block is introduced in connection with a one-machine scheduling problem.

In principle, the above theorem describes possible ways in which a cycle of positive length can be destroyed. Since the destruction of such cycles is necessary to obtain feasible solutions, we may use Theorem 4 to define neighborhoods. Let $\pi$ be a given infeasible sequence and let $C$ be a cycle of positive length in $N(R^{\pi})$. Furthermore, let $B_1,\ldots,B_r$ be the blocks of $C$. The neighborhood $\mathcal{N}(\pi)$ of the sequence $\pi$ is defined as the set of all sequences which are obtained by one of the following operators:

- A job $k \in B_i \setminus \{l(B_i)\}$ is moved directly after job $l(B_i)$.
- A job $k \in B_i \setminus \{f(B_i)\}$ is moved directly in front of job $f(B_i)$.

Obviously, the number of sequences in a neighborhood $\mathcal{N}(\pi)$ depends on the number of jobs in the cycle $C$ and the partition of this cycle into blocks. For the defined neighborhood we can prove that it is possible to reach a globally optimal solution independently of the initial solution.

**Theorem 5.** *The given neighborhood $\mathcal{N}$ is weakly connected, i.e. for each solution $\pi$ a sequence of solutions $\pi_1,\ldots,\pi_k$ exists such that $\pi_1=\pi$, $\pi_{i+1}\in\mathcal{N}(\pi_i)$, and $\pi_k$ is a globally optimal solution.*

**Proof.** w.l.o.g. we may assume that initially the length of the arc $(n+1,0)$ is given by $-K$, where $K$ is the makespan of an optimal solution. Thus, we only have to prove that it is possible to reach a feasible solution starting from an arbitrary solution $\pi$.

Let $\pi'$ be a feasible solution. Due to Theorem 4 there exists a job $j$ in some block $B_i$ of a positive cycle in $N(R^{\pi})$ which is sequenced in $\pi'$ before or after all other jobs of block $B_i$. If we move to the corresponding neighbor (move $j$ before or after $B_i$) we get a solution where $j$ is sequenced relative to all other jobs of $B_i$ in the same way as in $\pi$.

If we iteratively repeat this process, we never will destroy the relative order between $j$ and the other jobs of $B_i$ since we always choose the moving job according to Theorem 4. Thus, after a finite number of steps we will get to a feasible solution. □

The definition of the neighborhood of a sequence $\pi$ depends not only on $\pi$ but also on the chosen positive cycle $C$ in $N(R^{\pi})$. The connectivity of the defined neighborhood is independent of this choice. However, for a practical application we have to make a deterministic choice: If within the procedure *CalculateSchedule*($\pi$) infeasibility is detected, we reconstruct the positive cycle which corresponds to the last

considered time-lag. Otherwise, if *CalculateSchedule*($\pi$) produces a feasible schedule with makespan $K$, a critical path corresponding to this schedule together with the arc $(n+1,0)$ with new length $-(K-1)$ results in a cycle with length 1.

After defining the search space, the objective values, and the neighborhood, we can now describe the main structure of the developed local search approach. The basis of this approach is a tabu search method, which will be embedded in an outer method where some diversification of the search will be realized. The reason for applying diversification is that we do not want to give up the search at the first point at which the tabu search method gets stuck. Technically, this will be realized as follows: Whenever the tabu search method stops, we apply some diversification operator to the current best solution and again start tabu search with the changed solution. If this new run leads to an improvement, we repeat this process. Otherwise, if the last run of tabu search did not lead to an improvement, we apply some other diversification operator to the current best solution. We stop when all possible diversification operators have been applied without success. The structure of the outer method can be summarized as follows:

**Algorithm** *Local Search*
*calculate an initial sequence* $\pi$;
$\pi' := \pi$;
$\pi := TabuSearch(\pi)$;
**WHILE** *diversification is possible* **DO**
    **BEGIN**
    $\pi' :=$ *diversification*($\pi$);
    $\pi' :=$ *TabuSearch*($\pi'$);
    **IF** $\pi'$ *is better than* $\pi$ **THEN**
        $\pi := \pi'$;
    **END**;

In the above description the statement *diversification is possible* means that not all possible diversification operators have already been applied to the current best solution $\pi$. In our application of this algorithm we have considered two possible diversification operators. They are dependent on the blocks $B_1, \ldots, B_r$ of the chosen positive cycle $C$ in the network $N(R^\pi)$. More precisely, the first diversification operator $\mathrm{div}_1$ changes all blocks and the second diversification operator $\mathrm{div}_2$ only changes every second block of $C$ in the following way:

if $|B_i| \geqslant 4$: exchange $f(B_i)$ with its successor and
    $l(B_i)$ with its predecessor,
if $|B_i| \leqslant 3$: exchange $f(B_i)$ and $l(B_i)$ if this is allowed.

The initial sequence $\pi$ for the local search approach has been calculated by a priority rule-based heuristic. In the $i$th step of this heuristic a job will be sequenced at position $i$, i.e. job $\pi_i$ will be determined, and a starting time $S_{\pi_i}$ for this job will be fixed. Job $\pi_i$ is chosen by a priority rule from the set of jobs which have predecessors w.r.t.

the minimal time-lags only in the set $\{\pi_1,\ldots,\pi_{i-1}\}$ of already sequenced jobs. The starting time of job $\pi_i$ will be determined such that job $\pi_i$ is scheduled after job $\pi_{i-1}$ and respects all minimal time-lags from jobs in $\{\pi_1,\ldots,\pi_{i-1}\}$. Thus, the resulting schedule $S$ is partially feasible w.r.t. $\pi$. As priority rules we have used

P1: Choose the job with minimal earliest possible starting time.
P2: Choose the job with minimal latest possible starting time.
P3: Choose the job with maximal number of successors with respect to the minimal time-lags only.

It remains to describe the elements of the tabu search method we used in more detail.

- *Tabu list management*: We have chosen a tabu list of fixed length. This length will be defined depending on the size of the instance (e.g. $const \cdot \sqrt{n}$ or $const \cdot n$). The entries of the tabu list are tuples $(i,j)$, where $i$ and $j$ are jobs. If we apply an operator which moves a job $i$ directly before (after) a job $j$ we insert the tuple $(i,j)$ $((j,i))$ into the tabu list. The application of an operator is declared tabu if this operator exchanges two jobs which form a tuple of the tabu list.
  Besides this direct use of the tabu list, we have also used it to realize a diversification strategy. The idea behind this strategy is to help the tabu search method to occupy different regions of the search space. Our attempt to promote this goal is to forbid operators, if they make changes which already have been used often in the search process. More precisely, we store the number of times the tuple $(i,j)$ has been inserted into the tabu list in a variable $n_{ij}$, $i,j \in \{1,\ldots,n\}$. If we now try to apply an operator which exchanges two jobs $i$ and $j$, for which $n_{ij}+n_{ji} \geqslant FREQ$ ($FREQ$ is a given constant) holds, we declare this operator tabu, insert $(i,j)$ into the tabu list, and reset the values of $n_{ij}$ and $n_{ji}$ to 0.
- *Reduction of the size of the neighborhoods*: For the presented approach, the evaluation of one neighbored sequence is time-consuming since we have to execute a longest path calculation. Furthermore, the computational time for one iteration of tabu search depends directly on the size of the neighborhood. Thus, it is of importance to have small neighborhoods to get reasonable computational times for the tabu search approach. Theorem 4 gave us a first approach to get small neighborhoods. To get a further reduction of the number of neighbored sequences to be evaluated we make use of some easily calculable estimates of the change of the objective value. More precisely, for each possible operator op we calculate an estimate $E(\mathrm{op})$ of the change of the objective value if we apply operator op to the current solution. The correct objective values only will be calculated for a subset of neighbored sequences with good estimates.
  This approach is only useful if we have estimates which can be calculated much faster than the complete objective values. We have calculated the estimates as follows: If job $i$ is moved to the beginning (end) of a block $B$, we consider the potential new starting time $S'_i = S_{f(B)}$ ($S'_i = S_{l(B)} + p_{l(B)} - p_i$) for job $i$ and measure the infeasibility of the time-lags to job $i$ (from job $i$) by one of the measures $Z_1, Z_2, Z_3$ defined at the beginning of this section.

- *Calculation of the best non-tabu neighbor*: In a standard tabu search approach the best non-tabu neighbor is chosen as the next solution. However, as already mentioned, the objective measures for the infeasible sequences are not 'sharp'. To incorporate this into our search process, we have not determined the next sequence as the best non-tabu neighbor w.r.t. one of the objective measures but as follows: By a primary measure $Z$ we calculate a subset $S_1$ of good neighbors. More precisely, $S_1$ is the subset of all neighbored sequences for which the objective measure is within a distance $\Delta$ ($\Delta$ is a given constant) of the objective measure of the best neighbor. From this subset $S_1$ we choose as the next solution the best w.r.t. a second objective measure $Z'$.

  To reduce the computational times for one iteration of tabu search, we have also tested a 'first-improvement' strategy to determine the next solution. If, during the evaluation of the neighborhood of the current solution, a solution is detected which improves the current solution, we stop the evaluation of the neighborhood and directly move to the improving solution. Again, we will consider the unsharpness of the objective measures by only declaring a solution as an improving solution if its measure is at least $\Delta$ units better than that of the current solution. If the neighborhood contains no improving solution (in the sense just defined), we use the selection mechanism described above. This strategy may reduce the computational times for one iteration at the price of a potential decrease in the quality of the next solution.
- *Stopping condition*: We will stop the tabu search method if in the last *maxiter* iterations the current best solution has not been improved, where *maxiter* is a given constant.

## 5. Computational results

In the previous section we have described a local search approach for the single-machine problem with positive and negative time-lags. We implemented this method using the programming language *C* and we tested the algorithms on a SUN ULTRA, 167 MHz.

First, we will describe how several of the parameters of the local search approach were fixed. These decisions are the outcome of preliminary tests (a more detailed report on these tests is given in Keuchel [16]).

- *Initial solution*: The tests have shown that there is no significant difference between the three priority rules P1, P2, and P3. We have used P1.
- *Objective measure*: As a primary measure $Z$ the sum-objective $Z_3$ gave the best results. The unsharpness $\Delta$, by which the set of good neighbors $S_1$ w.r.t. the primary measure is determined, has been fixed to $n/8 + 34$. To select one solution from $S_1$ we choose the solution with maximal value $Z_3/Z_1$.
- *Tabu list management*: The length of the tabu list is $\sqrt{2n} - 2$ and the frequency counter *FREQ* for the diversification of the search was fixed at 7.

Table 1
Dimensions of the test instances

| Open-shop | $m \times n$ (orig.) | $n$ (red.) | Job-shop | $m \times n$ (orig.) | $n$ (red.) |
|---|---|---|---|---|---|
| tai01–tai10 | $4 \times 4$ | 34 | FT1 | $6 \times 6$ | 74 |
| tai11–tai20 | $5 \times 5$ | 52 | FT2 | $10 \times 10$ | 202 |
| tai21–tai30 | $7 \times 7$ | 100 | LA01–LA05 | $5 \times 10$ | 102 |
| tai81–tai90 | $8 \times 8$ | 130 | LA06–LA10 | $5 \times 15$ | 152 |
| tai91–tai100 | $9 \times 9$ | 164 | LA11–LA15 | $5 \times 20$ | 202 |
| tai31–tai40 | $10 \times 10$ | 202 | LA16–LA20 | $10 \times 10$ | 202 |

- *Reduction of the size of the neighborhood*: To reduce the neighborhoods, we will use as a measure for the estimates the sum-measure $Z_3$. Based on the estimates, the neighborhood of a sequence is reduced as follows: For each block $B$ the interchanges of the first two and the last two jobs are considered independently of their estimate. For blocks $B$ with $|B| \geqslant 4$ one right-shift and one left-shift is chosen additionally and for blocks $B$ with $|B| \geqslant 7$ two right-shifts and two left-shifts are chosen additionally.

To test the local search approach, no benchmark instances from the literature are available. However, Brucker et al. [5] have described rather direct reductions of shop problems to a single-machine problem with positive and negative time-lags which were used to generate test instances for a branch-and-bound method. Their computational results show that the reduced instances are very difficult. Thus, we will use these instances to test the local search approach, too. In detail, we considered the following problems:

- *Job-shop instances*: These instances result from a reduction of job-shop benchmark instances from Fisher and Thompson [13] (FT1, FT2) and from Lawrence [17]. The dimensions of the original job-shop instances ($n$ denotes the number of jobs and $m$ the number of machines) and of the reduced instances are given in Table 1.
- *Open-shop instances*: These instances result from a reduction of open-shop benchmark problems from Taillard [22] (tai01–tai40) and Brucker et al. [6] (tai81–tai100). Problems tai91–tai100 (tai81–tai90) are obtained from the instances tai31–tai40 from Taillard by removing the last (and the second-last) job and machine, i.e. by removing the last (and the second-last) row and column. The dimensions of the original open-shop instances (again $n$ denotes the number of jobs and $m$ the number of machines) and of the reduced instances are given in Table 1.

In a series of tests we tried to find out the influence of the selection strategies (Best-Improvement or First-Improvement), the size of the neighborhood (complete or reduced neighborhood), and the number of iterations which may pass without any global improvement within the tabu search approach (*maxiter*) on the efficiency of the developed local search approach. Tables 2–4 contain the results of six different versions of the local search approach. The tables contain the following informations:

- *instance*: denotes the group of considered instances in the corresponding row (tai-all represents the complete set of open-shop instances),
- *num*: number of instances in the instance group,

Table 2
Best-Improvement strategy and complete neighborhood

| Instance | No. | Best-improvement complete | | | | | |
|---|---|---|---|---|---|---|---|
| | | *maxiter* = 250 | | | *maxiter* = 500 | | |
| | | diff | opt | time | diff | opt | time |
| FT1 | 1 | 0.0 | 1 | 26.43 | 0.0 | 1 | 49.75 |
| FT2 | 1 | 73.0 | 0 | 1093.92 | 22.0 | 0 | 2883.65 |
| LA01–LA05 | 5 | 3.8 | 3 | 104.25 | 3.8 | 3 | 173.37 |
| LA06–LA10 | 5 | 0.0 | 5 | 16.22 | 0.0 | 5 | 16.31 |
| LA11–LA15 | 5 | 0.0 | 5 | 185.97 | 0.0 | 5 | 182.35 |
| LA16–LA20 | 5 | 22.8 | 1 | 684.04 | 14.8 | 1 | 1298.51 |
| tai01–tai10 | 10 | 3.5 | 1 | 7.16 | 1.5 | 4 | 14.99 |
| tai11–tai20 | 10 | 13.1 | 0 | 36.76 | 11.7 | 0 | 67.30 |
| tai21–tai30 | 10 | 18.6 | 0 | 292.42 | 16.6 | 0 | 606.74 |
| tai81–tai90 | 10 | 8.7 | 0 | 684.48 | 7.1 | 1 | 1142.43 |
| tai91–tai100 | 10 | 18.7 | 1 | 2045.15 | 17.4 | 1 | 3127.04 |
| tai31–tai40 | 10 | 38.6 | 0 | 4545.58 | 32.1 | 0 | 7730.94 |
| tai-all | 60 | 16.9 | 2 | — | 14.4 | 6 | — |

Table 3
Best-Improvement strategy and reduced neighborhood

| Instance | No. | Best-improvement reduced | | | | | |
|---|---|---|---|---|---|---|---|
| | | *maxiter* = 500 | | | *maxiter* = 1000 | | |
| | | diff | opt | time | diff | opt | time |
| FT1 | 1 | 0.0 | 1 | 33.30 | 0.0 | 1 | 67.87 |
| FT2 | 1 | 20.0 | 0 | 2437.14 | 13.0 | 0 | 3472.09 |
| LA01–LA05 | 5 | 3.4 | 2 | 81.12 | 3.4 | 2 | 135.48 |
| LA06–LA10 | 5 | 0.0 | 5 | 12.30 | 0.0 | 5 | 12.25 |
| LA11–LA15 | 5 | 0.0 | 5 | 53.24 | 0.0 | 5 | 52.81 |
| LA16–LA20 | 5 | 16.4 | 0 | 627.22 | 15.0 | 0 | 1275.92 |
| tai01–tai10 | 10 | 2.1 | 4 | 11.02 | 0.9 | 7 | 21.89 |
| tai11–tai20 | 10 | 10.3 | 1 | 48.42 | 8.9 | 0 | 87.93 |
| tai21–tai30 | 10 | 23.9 | 0 | 316.84 | 21.1 | 0 | 614.59 |
| tai81–tai90 | 10 | 5.3 | 3 | 832.16 | 5.5 | 3 | 1038.12 |
| tai91–tai100 | 10 | 18.5 | 0 | 1960.92 | 17.1 | 0 | 3288.73 |
| tai31–tai40 | 10 | 29.7 | 0 | 4186.37 | 26.7 | 0 | 7444.30 |
| tai-all | 60 | 15.0 | 8 | — | 13.4 | 10 | — |

- *diff*: average absolute difference between the achieved objective values and the optimal values (the presentation of relative difference is not useful, since by the reduction the optimal values of the single-machine instances are approximately $n + m$ times larger than the optimal values of the original shop instances),
- *opt*: number of instances for which the optimal values were found,
- *time*: average computational time (in s).

The results in Tables 2 and 3 show that an increase of the *maxiter* value leads — as expected — to an improvement of the results. However these improvements are

Table 4
First-improvement strategy

| Instance | No. | First-improvement *maxiter* = 500 | | | | | |
|---|---|---|---|---|---|---|---|
| | | Complete | | | Reduced | | |
| | | diff | opt | time | diff | opt | time |
| FT1 | 1 | 0.0 | 1 | 36.67 | 0.0 | 1 | 26.68 |
| FT2 | 1 | 25.0 | 0 | 1222.04 | 45.0 | 0 | 1006.81 |
| LA01–LA05 | 5 | 6.0 | 2 | 131.83 | 7.8 | 2 | 67.01 |
| LA06–LA10 | 5 | 0.0 | 5 | 12.29 | 0.0 | 5 | 8.58 |
| LA11–LA15 | 5 | 0.0 | 5 | 82.73 | 0.0 | 5 | 34.90 |
| LA16–LA20 | 5 | 15.0 | 0 | 844.81 | 24.4 | 0 | 524.55 |
| tai01–tai10 | 10 | 1.5 | 4 | 9.80 | 2.5 | 2 | 8.21 |
| tai11–tai20 | 10 | 11.5 | 0 | 39.80 | 8.9 | 0 | 31.62 |
| tai21–tai30 | 10 | 19.8 | 0 | 437.52 | 19.2 | 0 | 282.12 |
| tai81–tai90 | 10 | 6.5 | 4 | 821.94 | 5.8 | 2 | 677.00 |
| tai91–tai100 | 10 | 19.0 | 1 | 2649.45 | 16.6 | 1 | 1568.79 |
| tai31–tai40 | 10 | 38.0 | 0 | 5808.12 | 33.4 | 0 | 4343.44 |
| tai-all | 60 | 16.1 | 9 | — | 14.4 | 5 | — |

mostly not very large and they cost a lot of additional computational time. On the other hand the results of these two tables (columns with $maxiter = 500$) indicate that a reduction of the neighborhood does not change the quality of the solutions a lot, but reduces the computational times significantly. If we spend this additional time for further iterations (enlarging *maxiter*) the results of the reduced neighborhood are better than that of the complete neighborhood (column reduced, $maxiter = 500$ vs. column complete, $maxiter = 250$ and column reduced, $maxiter = 1000$ vs. column complete, $maxiter = 500$).

Another element to reduce computational times is given by the first-improvement strategy. Results for this strategy are given in Table 4. If we compare these results with the corresponding results for the best-improvement strategy (see Tables 2 and 3), we observe that the computational times in several cases do not decrease very much. This may result from an increase of the number of executed iterations. On the other hand, the replacement of best-improvement by first-improvement also does not lead to a big decrease in quality. Sometimes the results are even better.

For determining a 'best' version, we have to divide the six considered versions into two group of each three versions which roughly use the same computational times. If we want to spend much computational time, the best-improvement strategy with the reduced neighborhood and $maxiter = 1000$ leads to the best results. On the other hand, if we want to use less computational time, the job-shop instances should be solved using the best-improvement strategy with the reduced neighborhood and $maxiter = 500$ and the open-shop instances should be solved using the first-improvement strategy with the reduced neighborhood and $maxiter = 500$.

To get an impression of the efficiency of these two versions, we may compare them with results achieved by the branch and bound method for single-machine scheduling

Table 5
Comparison of favorite local search versions with results from literature

| Instance | No. | Long runs | | Short runs | | UB | B and B |
|---|---|---|---|---|---|---|---|
| | | diff | time | diff | time | diff | time |
| FT1 | 1 | 0.0 | 67.87 | 0.0 | 33.30 | 1 | 7 |
| FT2 | 1 | 13.0 | 3472.09 | 20.0 | 2437.14 | 161 | 853 792 |
| LA01–LA05 | 5 | 3.4 | 135.48 | 3.4 | 81.12 | 79 | 103 |
| LA06–LA10 | 5 | 0.0 | 12.25 | 0.0 | 12.30 | 21 | 284 |
| LA11–LA15 | 5 | 0.0 | 52.81 | 0.0 | 53.24 | 4 of 5 | |
| LA16–LA20 | 5 | 15.0 | 1275.92 | 16.4 | 627.22 | 0 of 5 | |
| tai01–tai10 | 10 | 0.9 | 21.89 | 2.5 | 8.21 | 59 | 4.4 |
| tai11–tai20 | 10 | 8.9 | 87.93 | 8.9 | 31.62 | 81 | 250 |
| tai21–tai30 | 10 | 21.1 | 614.59 | 19.2 | 282.12 | 121 | 7700 |
| tai81–tai90 | 10 | 5.5 | 1038.12 | 5.8 | 677.00 | 8 of 10 | |
| tai91–tai100 | 10 | 17.1 | 3288.73 | 16.6 | 1568.79 | 4 of 10 | |
| tai31–tai40 | 10 | 26.7 | 7444.30 | 33.4 | 4343.44 | 0 of 10 | |

problems with positive and negative time-lags of Brucker et al. [5]. Table 5 contains besides the results of the two favorite local search versions the computational times of the branch and bound method (Column B and B: an expression $x$ of $y$ means that the branch and bound method was only able to solve $x$ of the $y$ instances within two days) and the difference between a simple initial upper bound (column UB) used by Brucker et al. [5] and the optimal value. Although the computational times of the local search heuristics are quite large, they are much shorted than that of the branch and bound method and for larger instances the branch and bound method often is even not able to find the optimum. Furthermore, since the branch and bound method of Brucker et al. is a destructive method which tries to reach the optimum from below, this method does not provide upper bounds as intermediate results. The only other known (simple) upper bound (see column UB in Table 5) is far away from the quality of the local search results.

The overall quality of the results achieved with the the local search methods is good. E.g., for the famous $(10 \times 10)$ instance of Fisher and Thompson we get as best result a value of 943 which is only 13 units away from the optimal value. Although the current best heuristics developed especially for the job-shop problem find the optimal value of this instance and also need less time than our method, our result is encouraging since for the job-shop problem it took several years of research to get to this state of the art. Since we do not use any special properties of the shop problems but use the general approach for single-machine scheduling problems with positive and negative time-lags, the results of our approach are satisfactory.

Summarizing we can state that the developed local search approach is able to tackle the considered single-machine scheduling problem with positive and negative time-lags. For the test instances – which result from reductions from shop problems and which have shown to be hard to solve (see Brucker et al. [5]) – our approach always finds feasible solutions and the overall quality of the results is good.

## 6. Conclusion

In this paper we presented a local search approach for a single-machine scheduling problem with positive and negative time-lags. A crucial difficulty for developing such a method results from the $\mathcal{NP}$-completeness of the problem deciding whether or not a feasible schedule exists for a given instance. We have dealt with this problem by incorporating also infeasible sequences into the search process. The resulting local search method can be considered as a very general heuristic since due to reductions presented in Brucker et al. [5] many scheduling problems can be reduced rather straightforward to the considered single-machine scheduling problem with positive and negative time-lags. Computational results on instances achieved from shop problems indicate that the presented method succeeds in finding feasible and also good solutions. Since the computational times are still rather high, it is an interesting topic for further research to look for possibilities to speed up the developed method.

## References

[1] E. Aarts, J.K. Lenstra (Eds.), Local Search in Combinatorial Optimization, Wiley, Chichester, 1997.
[2] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, Network Flows: Theory, Algorithms and Applications, Prentice-Hall, Englewood Cliffs, NJ, 1993.
[3] M. Bartusch, R.H. Möhring, F.J. Rademacher, Scheduling project networks with resource constraints and time windows, Ann. Oper. Res. 16 (1988) 201–240.
[4] E. Balas, J.K. Lenstra, A. Vazacopoulos, One-machine scheduling with delayed precedence constraints and its use in job shop scheduling, Manage. Sci. 41 (1995) 94–109.
[5] P. Brucker, T. Hilbig, J. Hurink, A branch & bound algorithm for Scheduling problems with positive and negative time-lags, Discrete Appl. Math. 94 (1999) 77–99.
[6] P. Brucker, J. Hurink, B. Jurisch, B. Wöstmann, A branch & bound algorithm for the open shop problem, Discrete Appl. Math. 76 (1997) 43–59.
[7] P. Brucker, B. Jurisch, B. Sievers, A fast branch & bound algorithm for the job-shop scheduling problem, Discrete Appl. Math. 49 (1994) 107–127.
[8] K. Brinkmann, K. Neumann, Heuristic procedures for resource-constrained project scheduling with minimal and maximal time lags: the minimum project-duration and resource-levelling problems, J. Decision Systems 5 (1996) 129–156.
[9] J. Carlier, The one-machine sequencing problem, European J. Oper. Res. 11 (1982) 42–47.
[10] B. De Reyck, W. Herroelen, A branch-and-bound procedure for the resource-constrained project scheduling problem with generalized precedence relations, European J. Operl. Res. 111 (1998) 152–174.
[11] M. Dell'Amico, M. Trubian, Applying tabu search to the job-shop scheduling problem, Ann. Oper. Res. 41 (1993) 231–252.
[12] S.E. Elmaghraby, J. Kamburowski, The analysis of activity networks under generalized precedence relations, Manage. Sci. 38 (1992) 1245–1263.
[13] H. Fisher, G.L. Thompson, Probabilistic learning combinations of local job-shop scheduling rules, in: J.F. Muth, G.L. Thompson (Eds.), Industrial Scheduling, Prentice-Hall, Englewood Cliffs, NJ, 1963, pp. 225–251.
[14] J. Grabowski, E. Nowicki, S. Zdrzalka, A block approach for single machine scheduling with release dates and due dates, European J. Operl. Res. 26 (1986) 278–285.
[15] J. Hurink, B. Jurisch, M. Thole, Tabu search for the job shop scheduling problem with multi-purpose machines, OR-Spektrum 15 (1994) 205–215.
[16] J. Keuchel, Lokale Suchverfahren für Einmaschinenprobleme mit positiven und negativen Time Lags, Diplomarbeit, Universität Osnabrück, 1997.

[17] S. Lawrence, Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques, GSIA, Carnegie Mellon University, 1984.
[18] K. Neumann, C. Schwindt, Projects with minimal and maximal time lags: construction of activity-on-node networks and applications, Technical Report WIOR-447, Institut für Wirtschaftstheorie und Operations Research, University of Karlsruhe, 1995.
[19] E. Nowicki, C. Smutnicki, A fast taboo search algorithm for the job shop problem, Manage. Sci. 42 (1993) 797–813.
[20] C.H. Papadimitriou, K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity, Prentice-Hall, Englewood Cliffs, NJ, 1982.
[21] B. Roy, Contribution de la théorie des graphes à l'étude de certains problèmes linéaires, C.R. Acad. Sci. T 248 (1959) 2437–2439.
[22] E. Taillard, Benchmarks for basic scheduling problems, European J. Oper. Res. 64 (1993) 278–285.
[23] E.D. Wikum, D.C. Llewellyn, G.L. Nemhauser, One-machine generalized precedence constrained scheduling problems, Oper. Res. Lett. 16 (1994) 87–99.
[24] J. Zhan, Heuristics for scheduling resource-constrained projects in MPM networks, European J. Operl. Res. 76 (1994) 192–205.