Job shop scheduling by local search

Eindhoven University of Technology

Department of Mathematics and Computing Science

Job Shop Scheduling by Local Search

by

R.J.M. Vaessens, E.H.L. Aarts, J.K. Lenstra

94/15

Computing Science Note 94/15
Eindhoven, March 1994

COMPUTING SCIENCE NOTES

This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science Eindhoven University of Technology.
Since many of these notes are preliminary versions or may be published elsewhere, they have a limited distribution only and are not for review.
Copies of these notes are available from the author.

Copies can be ordered from:
Mrs. M. Philips
Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands
ISSN 0926-4515

editors: prof.dr.M.Rem
prof.dr.K.M.van Hee.

Job Shop Scheduling by Local Search

R.J.M. Vaessens [1]
E.H.L. Aarts [2,1]
J.K. Lenstra [1,3]

February 28, 1994

1. Eindhoven University of Technology, Department of Mathematics and Computing Science, P.O. Box 513, 5600 MB Eindhoven
2. Philips Research Laboratories, P.O. Box 80000, 5600 JA Eindhoven
3. CWI, P.O. Box 94079, 1090 GB Amsterdam

Abstract

We survey solution methods for the job shop scheduling problem with an emphasis on local search. We discuss both deterministic and randomized local search methods as well as the applied neighborhoods. We compare the computational performance of the various methods in terms of their effectiveness and efficiency on a standard set of problem instances.

Key words: job shop scheduling, local search, iterative improvement, shifting bottleneck heuristic, simulated annealing, taboo search, genetic algorithms, constraint satisfaction.

1 Introduction

In the job shop scheduling problem we are given a set of jobs and a set of machines. Each machine can handle at most one job at a time. Each job consists of a chain of operations, each of which needs to be processed during an uninterrupted time period of a given length on a given machine. The purpose is to find a schedule, that is, an allocation of the operations to time intervals on the machines, that has minimum length.

The problem is difficult to solve to optimality. For example, a relatively small instance with 10 jobs, 10 machines and 100 operations due to Fisher & Thompson [1963] remained unsolved until 1986. Many solution methods have been proposed, ranging from simple and fast dispatching rules to sophisticated branch-and-bound algorithms.

We survey algorithms for the job shop scheduling problem with an emphasis on local search. During the last decade many different types of local search algorithms for job shop scheduling have been developed, and some of them have proved to be very effective.

The paper is structured as follows. In Section 2 several models for the job shop scheduling problem are presented. In Section 3 the complexity of the problem and methods for its solution are reviewed. Section 4 introduces local search and Section 5 discusses representations and neighborhoods for the problem. Sections 6 and 7 describe constructive and iterative algorithms with local search, respectively; Section 8 describes some other techniques. Section 9 contains computational results, and Section 10 gives some concluding remarks.

2 The job shop scheduling problem

The job shop scheduling problem is formally defined as follows. Given are a set O of l operations, a set $\mathcal{M}$ of m machines, and a set $\mathcal{J}$ of n jobs. For each operation $v \in O$ there is a processing time $p(v) \in \mathbb{N}$, a unique machine $M(v) \in \mathcal{M}$ on which it requires processing, and a unique job $J(v) \in \mathcal{J}$ such that $v \in J(v)$. On O a binary relation A is defined, which represents *precedences* between operations: if $(v, w) \in A$, then v has to be performed before w. A is such that all operations v with the same value $J(v)$ are totally ordered.

A schedule is a function $S : O \to \mathbb{N} \cup \{0\}$ that for each operation v defines a start time $S(v)$. A schedule S is *feasible* if

$$\begin{array}{ll} \forall v \in O: & S(v) \geq 0, \\ \forall v, w \in O, (v, w) \in A: & S(v) + p(v) \leq S(w), \\ \forall v, w \in O, v \neq w, M(v) = M(w): & S(v) + p(v) \leq S(w) \text{ or } S(w) + p(w) \leq S(v). \end{array}$$

The *length* of a schedule S is $\max_{v \in O} S(v) + p(v)$, i.e., the earliest time at which all operations are completed. The problem is to find an *optimal* schedule, i.e., a feasible schedule of minimum length.

A feasible schedule is *left-justified*, if no operation can start earlier without changing the processing order on any machine. It is *active* if no operation can start earlier without delaying another operation. Note that at least one optimal schedule is active and that each active schedule is left-justified.

An instance of the problem can be represented by means of a *disjunctive graph* $G = (O, A, E)$ [Roy & Sussmann, 1964]. The vertices in O represent the operations, the arcs in A represent the given precedences between the operations, and the edges in $E = \{\{v, w\} \mid v, w \in O, v \neq w, M(v) = M(w)\}$ represent the machine capacity constraints. Each vertex $v \in O$ has a weight, equal to the processing time $p(v)$.

For each $E' \subseteq E$, an *orientation* on E' is a function $\Omega : E' \to O \times O$ such that $\Omega(\{v, w\}) \in \{(v, w), (w, v)\}$ for each $\{v, w\} \in E'$; we write $\Omega(E') = \{\Omega(e) \mid e \in E'\}$. A *partial orientation* is an orientation on $E' \neq E$ and a *complete orientation* is one on E. An orientation Ω on E' is *feasible* if the digraph $(O, A \cup \Omega(E'))$ is acyclic. It represents for each machine its *machine ordering*, i.e., the order in which it processes its operations. Each feasible schedule S uniquely determines a feasible complete orientation, which is denoted by Ω_S.

Conversely, for each feasible complete orientation Ω, there is a unique left-justified feasible schedule, which is denoted by S_Ω. For all $v \in O$, $S_\Omega(v)$ equals the length of a longest path in the digraph $(O, A \cup \Omega(E))$ upto and excluding v. The length of S_Ω equals the length of a longest path in the digraph. Finding an optimal left-justified schedule is now equivalent to finding a feasible complete orientation that minimizes the longest path length in the corresponding digraph.

The search for such a schedule can be restricted to the set of active schedules. However, given a feasible complete orientation Ω, it is not clear at first sight whether S_Ω is active: activeness depends on the sizes of the processing times, while left-justifiedness does not. For this reason one often considers the larger set of left-justified schedules.

3 Complexity and algorithms

We give a brief review of results on the computational complexity of job shop scheduling, of the lower bounds and enumeration schemes that are used in branch-and-bound methods, and

of the approximative approaches that yield upper bounds on the optimum. Techniques of the latter type that proceed by local search are discussed in the rest of the paper.

3.1 Complexity

Some very special cases of the problem can be solved in polynomial time, but their immediate generalizations are *NP*-hard. The results are summarized in Table 1, where l_j denotes the numer of operations of the jth job. Note that, due to result (4b), it is *NP*-hard to find a job shop schedule that is shorter than $\frac{5}{4}$ times the optimum.

Table 1: The complexity of job shop scheduling

solvable in polynomial time		*NP*-hard (in the strong sense*)	
(1a)	$m = 2$, all $l_j \leq 2$	(1b)	$m = 2$, all $l_j \leq 3$; $m = 3$, all $l_j \leq 2$
(2a)	$m = 2$, all $p(v) = 1$	(2b)*	$m = 2$, all $p(v) \leq 2$; $m = 3$, all $p(v) = 1$
(3a)	$n = 2$	(3b)*	$n = 3$
(4a)	length ≤ 3	(4b)*	length ≤ 4

(1a) [Jackson, 1956]; (1b) [Lenstra, Rinnooy Kan & Brucker, 1977]; (2a) [Hefetz & Adiri, 1982]; (2b) [Lenstra & Rinnooy Kan, 1979]; (3a) [Akers, 1956; Brucker, 1988]; (3b) [Sotskov, 1991]; (4a,b) [Williamson, Hall, Hoogeveen, Hurkens, Lenstra & Shmoys, 1994].

3.2 Lower bounds

Optimization algorithms for the problem employ some form of tree search. A node in the tree is usually characterized by a partial orientation Ω on a subset $E' \subset E$. The question is then how to compute a lower bound on the length of any feasible schedule corresponding to a completion of Ω.

Németi [1964] and many subsequent authors obtained a lower bound by simply disregarding $E \setminus E'$ and computing the longest path length in the digraph $(O, A \cup \Omega(E'))$.

Bratley, Florian & Robillard [1973] obtained the stronger *single-machine bound* by relaxing the capacity constraints of all machines except one. Given a machine M', they propose to solve the job shop scheduling problem on the disjunctive graph $(O, A \cup \Omega(E'), \{\{v, w\} \mid M(v) = M(w) = M'\} \setminus E')$. This is a single-machine problem, where the arcs in $A \cup \Omega(E')$ define release and delivery times for the operations on M' and precedence constraints between them. Lageweg, Lenstra & Rinnooy Kan [1977] pointed out that many other lower bounds appear as special cases of this bound. For example, relaxing the capacity constraint of M' gives Németi's bound, and allowing preemption gives the bound used in current branch-and-bound codes. The bound itself is *NP*-hard to compute but can be found fairly efficiently [Baker & Su, 1974; McMahon & Florian, 1975; Lageweg, Lenstra & Rinnooy Kan, 1976; Carlier, 1982]. It has been strengthened by Carlier & Pinson [1990], who compute larger release and delivery times, and by Tiozzo [1988] and Dauzère-Peres & Lasserre [1993], who observe that the arcs also define delays between precedence-related operations.

Fisher, Lageweg, Lenstra & Rinnooy Kan [1983] investigated surrogate duality relaxations, in which either the machine capacity constraints or the precedence constraints among the operations of a job are weighted and aggregated into a single constraint. Balas [1985] described a first attempt to obtain bounds by polyhedral techniques. Applegate & Cook [1991] review the valid inequalities studied before and gave some new ones. The computational performance

of surrogate duality and polyhedral bounds reported until now is disappointing in view of what has been achieved for other hard problems.

3.3 Enumeration schemes

The traditional enumeration scheme generates all active schedules by constructing them from front to back [Giffler & Thompson, 1960]. At each node a machine on which the earliest possible completion time of any unscheduled operation is achieved is determined, and all unscheduled operations that can start earlier than this point in time on that machine are selected in turn.

Recent branch-and-bound algorithms use more flexible enumeration schemes. Carlier & Pinson [1989; 1990] and Applegate & Cook [1991] branch by selecting a single edge and orienting it in either of two ways. Brucker, Jurisch & Sievers [1994] follow Grabowski's 'block approach'. All these authors apply the preemptive single-machine bound and a host of elimination rules. For details we refer to the literature.

The celebrated 10×10 instance of Fisher & Thompson [1963] is within easy reach of these methods, but 15×15 instances seem to be the current limit. The main deficiency of the existing optimization algorithms for job shop scheduling is the weakness of the lower bounds. The situation is much brighter with respect to finding good upper bounds.

3.4 Upper bounds

Upper bounds on the optimum are usually obtained by generating a schedule and computing its length. An obvious first step is to apply a dispatch rule and to schedule the operations according to some priority function. Haupt [1989] surveys such rules. They tend to exhibit an erratic behaviour; the procedure 'bidir' proposed by Dell'Amico & Trubian [1993] is one of the safer alternatives. The next step is then to try to improve the schedule by some sort of local search.

An entirely different approach is taken by Sevast'janov [1994]. Using Steinitz' vector sum theorem, he develops polynomial-time algorithms for finding an upper bound with an absolute error that is independent of the number of jobs. Shmoys, Stein & Wein [1994] improve on his results.

4 Local search

Local search is based on the idea that a given solution may be improved by making small changes. Solutions are changed over and again, and better and better solutions are found.

We need the following notions. There is a set F of *feasible solutions*. Two functions are defined on F. The *cost function* is a mapping $c : F \rightarrow \mathbb{R}$, which in most cases is closely related to the function that is to be optimized. The *neighborhood function* is a mapping $N : F \rightarrow 2^F$, which defines for each solution $x \in F$ a *neighborhood* $N(x) \subseteq F$. Each solution in $N(x)$ is called a neighbor of x. Roughly speaking, the execution of a local search algorithm defines a walk in F such that each solution visited is a neighbor of the previously visited one.

A solution $x \in F$ is called a *local minimum* with respect to a neighborhood function N if $c(x) \leq c(y)$ for all $y \in N(x)$. The basic algorithm to find a local minimum is called *iterative improvement*. Starting at some initial feasible solution, its neighborhood is searched for a

solution of lower cost. If such a solution is found, the algorithm is continued from there; otherwise, a local minimum has been found.

The quality of the local minimum depends on the initial solution, on the neighborhood function, and on the method of searching the neighborhoods. An initial solution may be obtained by generating it randomly or by applying a heuristic rule. The choice of a good neighborhood is often difficult. There is a clear trade-off between small and large neighborhoods: if the number of neighbors is larger, the probability of finding a good neighbor may be higher, but looking for it takes more time. There are several alternatives for searching the neighborhood: one may take the first neighbor found of lower cost (*first improvement*), or take the best neighbor in the entire neighborhood (*best improvement*), or take the best of a sample of neighbors, provided it is an improving one.

Often, one problem remains. The local optima obtained may be of poor quality. Therefore, several variants of iterative improvement have been proposed. The main variants can be divided into threshold algorithms, taboo search algorithms, and genetic algorithms.

In *threshold algorithms*, a neighbor of a given solution becomes the new current solution, if the cost difference between the current schedule and its neighbor is below a certain threshold. One distinguishes three kinds of threshold algorithms.

In classical *iterative improvement* the thresholds are 0, so that only true improvements are accepted. In *threshold accepting* [Dueck & Scheuer, 1990] the thresholds are nonnegative. They are large in the beginning of the algorithm's execution and gradually decrease to become 0 in the end. General rules to determine appropriate thresholds are lacking. In *simulated annealing* [Kirkpatrick, Gelatt & Vecchi, 1983; Černy, 1985] the thresholds are positive and stochastic. Their values equal $-T \ln u$, where T is a control parameter (often called 'temperature'), whose value gradually decreases in the course of the algorithm's execution according to a 'cooling schedule', and u is drawn from a uniform distribution on (0,1]. Each time a neighbor is compared with the current solution, u is drawn again. Under certain mild conditions simulated annealing is guaranteed to find an optimal solution asymptotically.

In *taboo search* one selects from a subset of *permissible neighbors* of the current solution a solution of minimum cost. In basic taboo search a neighbor is permissible if it is not on the 'taboo list' or satisfies a certain 'aspiration criterion'. The taboo list is recalculated at each iteration. It is often implicitly defined in terms of forbidden moves from the current solution to a neighbor. The aspiration criterion expresses possibilities to overrule the taboo-status of a neighbor. For details see Glover [1989; 1990] or Glover, Taillard & De Werra [1993].

Genetic algorithms [Holland, 1975] are based on an extended notion of neighborhood function. A *hyper-neighborhood function* is a mapping $N_h : F^s \rightarrow 2^F$ with $s \geq 2$, which defines for each s-tuple $\mathbf{x} = (x_1, \ldots, x_s) \in F^s$ a set $N_h(\mathbf{x}) \subseteq F$ of neighboring solutions. Each solution in $N_h(\mathbf{x})$ is called a hyper-neighbor of $\mathbf{x}$. At each iteration a set of solutions, often called 'population', is given. From this population several subsets of size s consisting of 'parents' are selected, and for each such subset some hyper-neighbors, called 'off-spring', are determined by operations called 'recombination' and 'mutation'. This set of hyper-neighbors and the current population are then combined and reduced to a new population by selecting a subset of solutions.

Several solution methods have been proposed that use local search in combination with other techniques like partial enumeration and backtracking. Such hybrid methods for job shop scheduling are dealt with below with an emphasis on nested forms of local search. Here, local search is applied at several levels with different neighborhoods. In this way the search can explore different regions of the solution space.

5 Solution representations and neighborhood functions

A crucial ingredient of a local search algorithm is the definition of a neighborhood function in combination with a solution representation. Below, several basic representations and neighborhood functions are introduced for the job shop scheduling problem. For most threshold and taboo search algorithms, only left-justified or active schedules are represented. This is done by specifying the start times of the operations or, equivalently, the corresponding machine orderings of the operations. Other representations are used too, especially in combination with genetic algorithms.

To be able to define the neighborhood functions, we need some extra notions. Given an instance and an operation v, $jp(v)$ and $js(v)$ denote the immediate predecessor and successor of v in the precedence relation A, provided they exist. Given a feasible schedule S and an operation v, $mp_S(v)$ and $ms_S(v)$ denote the immediate predecessor and successor of v in the orientation Ω_S, provided they exist. If the schedule S is clear from context, we delete the superscript S. Furthermore, $jp^2(v)$ denotes $jp(jp(v))$, provided it exists, and a similar notation is used for js, mp_S and ms_S. Two operations v and w are *adjacent* when $S(v)+p(v) = S(w)$. A *block* is a maximal sequence of size at least one, consisting of adjacent operations that are processed on the same machine and belong to a longest path. An operation of a block is *internal* if it is neither the first nor the last operation of that block.

Several neighborhood functions have been proposed in the literature. Most of these are not defined on a schedule S itself but on the corresponding orientation Ω_S. If Ω_S is changed into another feasible orientation Ω', $S_{\Omega'}$ is the corresponding neighbor of S. In this way neighbors of a given schedule are always left-justified.

The following properties [Balas, 1969; Matsuo, Suh & Sullivan, 1988; Nowicki & Smutnicki, 1993] are helpful in obtaining reasonable neighborhood functions.

1. Given a feasible orientation, reversing an oriented edge on a longest path in the corresponding digraph results again in a feasible orientation.

2. If reversing an oriented edge of a feasible orientation Ω that is not on a longest path results in a feasible orientation Ω', then $S_{\Omega'}$ is at least as long as S_Ω.

3. Given a feasible orientation Ω, reversing an oriented edge (v, w) between two internal operations of a block results in a feasible schedule at least as long as S_Ω.

4. Given is a feasible orientation Ω. If v is the first and w the second operation of the first block of a longest path and w is internal, then reversing (v, w) results in a feasible schedule at least as long as S_Ω. The same is true if w is the last and v the second last operation of the last block of a longest path and v is internal.

In view of these properties, the simplest neighborhood functions are based on the reversal of exactly one edge of a given orientation. Van Laarhoven, Aarts & Lenstra [1992] propose a neighborhood function N_1 which obtains a neighbor by reversing two adjacent operations of a block. Matsuo, Suh & Sullivan [1988] use a neighborhood function N_{1a} with the same reversals, except those involving two internal operations. Nowicki and Smutnicki [1993] use a neighborhood function N_{1b}, excluding from N_{1a} the reversal of the first two operations of the first block when the second one is internal and the reversal of the last two operations of the last block when the first is internal. For N_{1b}, neither a schedule with one block only nor one with blocks of size one only has a neighbor; note that such schedules are optimal.

Dell'Amico & Trubian [1993] propose several neighborhood functions that may reverse more than one edge. Their neighborhood function N_2 obtains for any two operations v and $w = ms(v)$ on a longest path a neighboring orientation by permuting $mp(v)$, v and w, or

by permuting v, w and $ms(w)$, such that v and w are reversed and a feasible orientation results. Their neighborhood function N_{2a} excludes from N_2 the solutions for which both v and $w = ms(v)$ are internal. Their neighborhood function N_3 considers blocks of size at least two: a neighbor is obtained by positioning an operation v immediately in front of or after the other operations of its block, provided that the resulting orientation is feasible; otherwise, v is moved to the left or the right as long as the orientation remains feasible.

Adams, Balas & Zawack [1988] propose a neighborhood function N_4 which may completely change one machine ordering. For every machine M' with an operation on a longest path, a neighbor is obtained by replacing the orientation on M' by any other feasible orientation.

The following neighborhood functions obtain a neighbor by changing several machine orderings at the same time. Relatively small modifications are made by the neighborhood function N_5 of Matsuo, Suh & Sullivan [1988], which reorients at most three edges simultaneously. A neighbor is obtained by reversing two adjacent operations v and $w = ms(v)$ of a block (except when they are both internal) and in addition by reversing $jp^t(w)$ and $mp(jp^t(w))$ for some $t \geq 1$ and by reversing $js(v)$ and $ms(js(v))$. The latter reversals are executed only if certain additional conditions are satisfied; see their paper for details. Aarts, Van Laarhoven, Lenstra & Ulder [1994] use a variant N_{5a}, where $t \leq 1$.

Applegate & Cook [1991] propose a neighborhood function N_6 which drastically changes the given orientation. Their neighborhood contains all feasible orientations that can be obtained by simultaneously replacing the orientation on $m - t$ machines by any other feasible orientation. Here, t is a small number depending on m.

Storer, Wu & Vaccari [1992] use completely different representations of schedules. These are based on a modified version of the Giffler-Thompson algorithm (see Section 3.3). Suppose that at a certain point the earliest possible completion time of any unscheduled operation is equal to C and is achieved by operation v, and that T is the earliest possible start time on machine $M(v)$. Then all unscheduled operations on $M(v)$ that can start no later than $T + \delta(C - T)$ are candidates for the next position on $M(v)$. Here, δ is a priori chosen in [0,1) (in experiments 0, 0.05 or 0.1); if δ approaches 1 all active schedules can be generated, while $\delta = 0$ gives only so-called *non-delay* schedules. Two representations are defined.

The representation R_7 represents a schedule by modified processing times for the operations. Using these, the modified Giffler-Thompson algorithm with the shortest processing time rule as selection rule uniquely determines a feasible orientation Ω, and S_Ω, computed with the original processing times, is the corresponding schedule. The neighborhood function N_7 now obtains a neighbor by increasing the processing times by amounts of time that are independently drawn from a uniform distribution on $(-\theta, \theta)$. Here, θ is a priori chosen (in experiments 10, 20 or 50). The representation R_8 represents a schedule by dividing the scheduling horizon into several (in experiments 5, 10 or 20) time windows and assigning one of a given set of dispatch rules to each window. The modified Giffler-Thompson algorithm determines a schedule by applying the dispatch rule of the corresponding window. The neighborhood function N_8 changes the dispatch rule for a window of a given schedule.

Genetic algorithms use two types of representations: the natural one, which is also used for most threshold and taboo search algorithms, and a more artificial one, using binary strings.

For the former type of representation Yamada & Nakano [1992] propose a hyper-neighborhood function N_{h1} which, given two schedules S and S', determines a neighbor using the Giffler-Thompson algorithm. When this algorithm has to choose from two or more operations, it takes, for a small $\epsilon > 0$, the operation that is first in S with probability $\frac{1-\epsilon}{2}$, the operation that is first in S' with probability $\frac{1-\epsilon}{2}$, and a random operation from the other available

operations with probability ϵ.

Aarts, Van Laarhoven, Lenstra & Ulder [1994] propose a hyper-neighborhood function N_{h2}. Given two schedules S and S', N_{h2} determines a neighbor by repeating the following step $\lfloor \frac{nm}{2} \rfloor$ times: choose a random arc (w, v) of S' and change S by reversing arc (v, w), provided it belongs to a longest path of S.

The latter type of representation encodes a schedule or its orientation into a string over a finite - usually binary - alphabet. Such representations facilitate the application of hyper-neighborhood functions involving operations like 'crossover' and 'mutation'; see Goldberg [1989, pp. 166-175]. There are a number of drawbacks, however. A schedule or orientation may have several representatives, or none. Conversely, a string does not have to represent a schedule, and if it does, it may be nontrivial to calculate the corresponding schedule. Although attempts have been made to circumvent these difficulties, the hyper-neighborhood functions that operate on strings often have no meaningful effect in the context of the underlying problem. We will consider genetic algorithms using string representations in less detail.

6 Constructive algorithms with local search

In this section we discuss the *shifting bottleneck procedure* and its variants. These algorithms construct a complete schedule and apply local search to partial schedules on the way. A partial schedule is characterized by a partial orientation Ω on a subset $E' \in E$. Its length is defined as the longest path length in the digraph $(O, A \cup \Omega(E'))$.

The basic idea of the algorithms described here is as follows. The algorithm goes through m stages. At each stage, it orients all edges between operations on a specific machine. In this way, at the beginning of any stage all edges related to some machines have been oriented, while the edges related to the other machines are not yet oriented. Furthermore, at the end of each stage, it applies iterative best improvement to the current partial schedule using neighborhood function N_4, which revises the orientation on a machine scheduled before. Orienting or reorienting the edges related to one machine in an optimal way requires the solution of a single-machine problem, where the partial schedule defines release and delivery times and delayed precedence constraints. The algorithms discussed hereafter mainly differ by the order in which the m machines are considered, by the implementation of iterative best improvement, and by the single-machine algorithm used.

The original shifting bottleneck procedure SBI of Adams, Balas & Zawack [1988] orients at each stage the edges related to the *bottleneck machine*. This is the unscheduled machine for which the solution value to the corresponding single-machine problem is maximum; the delays between precedence-related operations are not taken into account. After scheduling a machine, iterative best improvement is applied during three cycles. In each cycle each scheduled machine is reconsidered once. The first cycle handles the machines in the order in which they were sequenced. After a cycle is completed, the machines are reordered according to decreasing solution values to the single-machine problems in the last cycle. When all of the machines have been scheduled, the cycles continue as long as improvements are found. Furthermore, after a phase of iterative best improvement, the orientations on several machines that have no operations on a longest path are deleted, and then these machines are rescheduled one by one.

Applegate & Cook [1991] use almost the same algorithm. The main difference is that at

each stage iterative improvement cycles continue until no further improvements are found.

Dauzère-Peres & Lasserre [1993] were the first to take the delays between precedence-related operations into account. They develop a heuristic for the single-machine problem with delayed precedences and incorporate it into a shifting bottleneck variant.

Balas, Lenstra & Vazacopoulos [1994] develop an algorithm to solve the single-machine problem with delayed precedences to optimality, and use it to determine the bottleneck machine in their shifting bottleneck procedure SB3. Their local search strategy differs from the one of Adams, Balas & Zawack [1988] in some minor details; for instance, the number of cycles is limited to six. Again, after scheduling a new machine, they first apply iterative improvement, then delete the orientations on several non-critical machines, and reschedule these machines one by one. They also propose an extension SB4, which takes the best solution of SB3 and a variant of SB3 that reverses the order of the two reoptimizations procedures: first reschedule some non-critical machines, then apply regular iterative improvement. SB4 needs roughly twice the computation time of SB3 but finds several better schedules.

The shifting bottleneck procedure and its variants have been incorporated into other algorithms. Most of these employ some form of partial enumeration. Dorndorf & Pesch [1994] embed a variant in a genetic algorithm; see Section 7.3.

Adams, Balas & Zawack [1988] develop an algorithm SBII, which applies SBI to the nodes of a partial enumeration tree. A node corresponds to a subset of machines that have been scheduled in a certain way. In each of its descendants one more machine is scheduled. The schedule is obtained by first solving the single-machine problem, with release and delivery times defined by the parent node, and then applying iterative improvement as in SBI. Descendants are created only for a few machines with highest solution values to the single-machine problem. A penalty function is used to limit the size of the tree. For details about the branching rule, the penalty function and the search strategy we refer the reader to the original paper.

Applegate & Cook [1991] develop an algorithm Bottle-t, which employs partial enumeration in a different way. Bottle-t applies their shifting bottleneck variant described above as long as more than t machine are left unscheduled. For the last t machines it branches by selecting each remaining unscheduled machine in turn. The values $t = 4$, 5 and 6 were tested.

7 Iterative algorithms with local search

The algorithms presented in this section start from one or more given feasible schedules and manipulate these in an attempt to find better schedules. They can naturally be divided into threshold algorithms, taboo search algorithms, and genetic algorithms.

7.1 Threshold algorithms

The basic threshold algorithms are iterative improvement, threshold accepting, and simulated annealing. We also consider some closely related variants. Unless stated otherwise, a schedule is represented in the ordinary way by the starting times or the orientation.

Iterative improvement is the simplest threshold algorithm. Aarts, Van Laarhoven, Lenstra & Ulder [1994] test iterative improvement with the neighborhood functions N_1 and N_{5a}. To obtain a fair comparison with other algorithms they apply a *multi-start* strategy, i.e., they

run the algorithm with several randomly generated start solutions until a limit on the total running time is reached, and take the best solution found over all individual runs.

The algorithm Shuffle of Applegate & Cook [1991] uses the neighborhood function N_6. At each iteration, the schedule on a small number of heuristically selected machines remains fixed, and the schedule on the remaining machines is optimally revised by their branch-and-bound algorithm 'edge finder'. As initial solution they take the result of Bottle-5.

Storer, Wu & Vaccari [1992] propose a variant of iterative improvement, called PS10, with representation R_7 and neighborhood function N_7. Given a solution, a fixed number of neighbors (in experiments 100 or 200) is determined, the best one of which becomes the new solution. They also test a standard iterative first improvement algorithm, called HSL10, with representation R_8 and neighborhood function N_8. Neighbors are generated randomly and the algorithm stops after a fixed number of iterations (in experiments 1000 or 2000).

Threshold accepting has only been implemented by Aarts, Van Laarhoven, Lenstra & Ulder [1994]. Their algorithm TA1 uses the neighborhood function N_1. Threshold values are determined empirically.

Simulated annealing has been tested by several authors. Van Laarhoven, Aarts & Lenstra [1992] use the neighborhood function N_1. Aarts, Van Laarhoven, Lenstra & Ulder [1994] use N_1 (algorithm SA1) and N_{5a} (algorithm SA2).

Matsuo, Suh & Sullivan's [1988] 'controlled search simulated annealing' algorithm CSSA is a bi-level variant, which also incorporates standard iterative improvement. Given a schedule S, a neighbor S' is selected using the neighborhood function N_5. S' is accepted or rejected by the simulated annealing criterion. In the latter case, S' is subjected to iterative improvement using N_5 again, and if the resulting local optimum improves on S, it is accepted as the new solution. Their method also differs from most other implementations of simulated annealing in that the acceptance probability for a schedule that is inferior to the current schedule is independent of the difference in schedule length.

7.2 Taboo search algorithms

The taboo search algorithm TS1 of Taillard [1994] uses the neighborhood function N_1. After an arc (v, w) has been reversed, the reversal of w and its machine successor is put on the taboo list. Every 15 iterations a new length of the taboo list is randomly selected from a range between 8 and 14. The length of a neighbor is estimated in such a way that the estimate equals the length of the new schedule when both operations involved are still on a longest path, and that it is a lower bound otherwise. Then, from the permissible neighbors the schedule of minimum estimated length is selected as the new schedule.

The algorithm TS2 of Barnes & Chambers [1994] also uses N_1. Their taboo list has a fixed length. If no permissible moves exist, the list is emptied. The length of each neighbor is calculated exactly, not estimated. A start solution is obtained by taking the best from the active and non-delay schedules obtained by applying seven dispatch rules.

The algorithm TS3 of Dell'Amico & Trubian [1993] uses the union of the neighborhoods generated by N_{2a} and N_3. The items on the taboo list are forbidden reorientations of arcs. Depending on the type of neighbor, one or more such items are on the list. The length of the list depends on the fact whether the current schedule is shorter than the previous one and the best one, or not. Furthermore, the minimal and maximal allowable lengths of the list are changed after a given number of iterations. When all neighbors are taboo and do not satisfy the aspiration criterion, a random neighbor is chosen as the next schedule. A start

solution is obtained by a procedure called 'bidir', which applies list scheduling simultaneously from the beginning and the end of the schedule, i.e., an operation is available when all of its predecessors or all its successors have been scheduled.

Nowicki & Smutnicki [1993] introduce an algorithm TSAB that combines taboo search with a backtracking scheme. In the taboo search part of their algorithm, the neighborhood function is a variant of N_{1b}, which only allows reorientations of arcs on a single longest path. The items on the taboo list are forbidden reorientations of arcs. The length of the list is fixed to 8. If no permissible neighbor exists, the following is done. If there is one neighbor only, which as a consequence is taboo, this one becomes the new schedule. Otherwise, the oldest items on the list are removed one by one until there is one non-taboo neighbor, and this one is chosen. A start solution is obtained by generating an active schedule using the shortest processing time rule or an insertion algorithm.

The backtracking scheme forces the taboo search to restart from promising situations encountered before. Suppose that at a certain point a new best schedule S is found. Let $R(S)$ be the set of feasible arc reversals in S, let T be the new taboo list, including the inverse of the reversal needed to obtain S, and let r be the reversal that will be made in the next iteration. Then, if $|R(S)| \geq 2$, the triple $(S, R(S) \setminus \{r\}, T)$ is stored on a list. This list has a maximum length of 5; if it is full, the oldest triple is deleted before a new one is stored. Each time the taboo search algorithm stops (by reaching a maximum number of iterations without improving the best schedule), the backtracking scheme initiates a new round of taboo search starting from the schedule, the set of reversals and the taboo list of the last stored triple. When the set of reversals has one element only, the triple is deleted from the list; otherwise, it is replaced by the same triple with the reversal that will be made in the next iteration excluded. Note that during this new round new triples can be added to the list.

7.3 Genetic algorithms

The genetic algorithm GA1 of Yamada & Nakano [1992] determines for every chosen pair of schedules of the current population, two hyper-neighbors by using N_{h1}. From these four schedules two are selected for the next population: first the best schedule is chosen, and next the best unselected hyper-neighbor is chosen.

Aarts, Van Laarhoven, Lenstra & Ulder [1994] propose a genetic algorithm that incorporates iterative first improvement. In each iteration there is a population of solutions that are locally optimal with respect to either N_1 (algorithm GLS1) or N_{5a} (algorithm GLS2). The population is doubled in size by applying N_{h2} to randomly selected pairs of schedules of the population. Each hyper-neighbor is subjected to iterative first improvement, using N_1 or N_{5a}, and the extended population of local optima is reduced to its original size by choosing the best schedules. Then a next iteration is started. Start solutions are generated randomly, and iterative first improvement is applied to them before the genetic algorithm is started.

In the work of Davis [1985], Falkenauer & Bouffouix [1991] and Della Croce, Tadei & Volta [1994] a string represents for each machine a preference list, which defines a preferable ordering of its operations. From such a list a schedule is calculated. Davis [1985] and Falkenauer & Bouffouix [1991] restrict themselves to non-delay schedules; Della Croce, Tadei & Volta [1994] are able to represent other schedules as well. Falkenauer & Bouffouix [1991] and Della Croce, Tadei & Volta [1994] use the linear order crossover as hyper-neighborhood function. See the original papers for details.

Nakano & Yamada [1991] consider problem instances with exactly one operation for each job-machine pair. For each machine and each pair of jobs, they represent the order in which that machine executes those jobs by one bit. Thus, a schedule is represented by a string of $\frac{mn(n-1)}{2}$ bits. Since such a string may not represent a feasible orientation, they propose a method for finding a feasible string that is close to a given infeasible one. Two hyper-neighbors are obtained by cutting two strings at the same point and exchanging their left parts.

Dorndorf & Pesch [1994] propose a 'priority rule based genetic algorithm' P-GA, which uses the Giffler-Thompson algorithm. They use a string $(p_1, \ldots, p_{l-1})$, where p_i is a dispatch rule that resolves conflicts in the ith iteration of the algorithm. Two hyper-neighbors are obtained by cutting two strings at the same point and exchanging their left parts. These authors also propose a second genetic algorithm, called SB-GA, which uses a shifting bottleneck procedure (see Section 6). A solution is represented by a sequence of the machines. A corresponding schedule is generated by a variant of SBI: each time it has to select an unoriented machine, it chooses the first unoriented machine in this sequence. The hyper-neighborhood function used is the cycle crossover; see Goldberg [1989, p. 175]. In contrast to SBI, reoptimization is applied only when less than six machines are left unscheduled.

8 Other techniques

8.1 Constraint satisfaction

Constraint satistisfaction algorithms consider the decision variant of the job shop scheduling problem: given an overall deadline, does there exist a feasible schedule meeting the deadline? Most algorithms of this type apply tree search and construct a schedule by assigning start times to the operations one by one. A *consistency checking* process removes inconsistent start times of not yet assigned operations. If it appears that a partial schedule cannot be completed to a feasible one, a *dead end* is encountered, and the procedure has to undo several assignments. Variable and value ordering heuristics determine the selection of a next operation and its start time. The algorithm stops when a feasible schedule meeting the deadline has been found or been proved not to exist. Note that it is also possible to establish lower bounds on the optimum with this technique.

Sadeh [1991] developed an algorithm of this type, but its performance was poor. Nuijten, Aarts, Van Erp Taalman Kip & Van Hee [1993] improved it by designing new variable and value orderings and extensive consistency checking techniques. The resulting algorithm improved upon Sadeh's method in terms of both solution quality and speed, but still could not compete with the best job shop scheduling algorithms. The authors therefore modified their algorithm such that, when a dead end occurs, it restarts the search from the beginning, and they also randomized the selection of a next operation and its start time. Their 'randomized constraint satisfaction' algorithm RCS performs quite well.

8.2 Neural networks

Foo & Takefuji [1988a,b] describe a solution approach based on the deterministic neural network model with a symmetrically interconnected network, introduced by Hopfield & Tank [1985]. The job shop scheduling problem is represented by a 2-dimensional matrix of neurons. Zhou, Cherkassky, Baldwin, & Olson [1991] develop a neural network algorithm which uses a linear cost function instead of a quadratic one. For each operation there is one neuron in

the network, and also the number of interconnections is linear in the number of operations. The algorithm improves the results of Foo & Takefuji both in terms of solution quality and network complexity. Altogether, applications of neural networks to the job shop scheduling problem are at an initial stage, and the reported computational results are poor upto now.

9 Computational results

The computational merits of job shop scheduling algorithms have often been measured by their performance on the notorious 10×10 instance FT10 of Fisher & Thompson [1963]. Applegate & Cook [1991] found that several instances of Lawrence [1984] (LA21, LA24, LA25, LA27, LA29, LA38, LA40) pose a more difficult computational challenge. We have included the available computational results for these instances and, in addition, for two relatively easy instances (LA2, LA19) and for all remaining 15×15 instances of Lawrence (LA36, LA37, LA39). Each of these 13 instances has exactly one operation for each job-machine pair.

Tables 2 and 3 present the computational results for most algorithms discussed in Sections 6, 7 and 8, as far as these are available from the literature. Table 2 gives the schedule lengths obtained and Table 3 the corresponding running times.

In Table 2 the values UB and LB are the best known upper and lower bounds for the corresponding instances. For each method mentioned, a superscript b followed by a number x indicates that the results reported are the best ones obtained after x runs of the algorithm. If a superscript m occurs, the results reported are means over several runs. A superscript 1 indicates that the results reported are obtained by a single run. A hyphen denotes that no result is available. Furthermore, for each method and for each instance, we computed the relative error, i.e., the percentage that the best solution value found is above LB. For each algorithm, the values MRE and SRE are the mean relative error and the standard deviation of the relative error. Note that UB has already an MRE of 0.40.

Table 3 gives the CPU-times for the corresponding results of Table 2. If the result in Table 2 is a mean over several runs, the average CPU-time of a single run is given; if it is a best result over several runs, the total CPU-time is given. The value SCT is the sum of CPU-times over all instances; it was estimated if not all individual CPU-times were available. Also the computer used is mentioned. For each algorithm, the value CISCT is a computer-independent sum of CPU-times, computed using the work of Dongarra [1993].

Figure 1 shows for each algorithm the mean relative error and the corresponding computer-independent sum of CPU-times. Note that the time-axis has a logarithmic scale.

The *shifting bottleneck* procedure SBI of Adams, Balas & Zawack [1988] is fast but gives poor results. The variants SB3 and SB4 of Balas, Lenstra, & Vazacopoulos [1994], which take the delayed precedences into account, clearly give better results. The algorithm SBII of Adams, Balas & Zawack [1988], which uses partial enumeration, also improves upon SBI. Applegate & Cook [1991] obtain reasonable results with their algorithms Bottle-5 and Bottle-6. Note that the values given in Tables 2 and 3 do not correspond to those presented in their paper. We computed our values using the enumeration scheme described in their paper; their values were obtained using a different scheme [Applegate & Cook, 1993]. The results suggest that the straight shifting bottleneck procedure (as in SBI, SB3 and SB4) benefits from some form of partial enumeration (as in SBII and Bottle-t).

Among *threshold algorithms* the best results are obtained by the simulated annealing

Table 2: Performance comparison of various algorithms: schedule lengths for 13 instances

Authors	Algorithm	FT10	LA2	LA19	LA21	LA24	LA25	LA27	LA29	LA36	LA37	LA38	LA39	LA40	MRE	SRE
UB		930	655	842	1046	935	977	1236	1160	1268	1397	1196	1233	1222	0.40	1.00
LB		930	655	842	1040	935	977	1235	1120	1268	1397	1184	1233	1222		
AdamsBZ	SBI[1]	1015	720	875	1172	1000	1048	1325	1294	1351	1485	1280	1321	1326	8.41	2.98
BalasLV	SB3[1]	981	667	902	1111	976	1012	1272	1227	1319	1425	1318	1278	1266	5.11	2.87
BalasLV	SB4[1]	940	667	878	1071	976	1012	1272	1227	1319	1425	1294	1278	1262	4.07	2.57
AdamsBZ	SBII[1]	930	669	860	1084	976	1017	1291	1239	1305	1423	1255	1273	1269	3.85	2.54
AppleCook	Bottle-4[1]	938	667	863	1094	983	1029	1307	1220	1326	1444	1299	1301	1295	4.98	2.52
AppleCook	Bottle-5[1]	938	662	847	1084	983	1001	1288	1220	1316	1444	1299	1291	1295	4.24	2.81
AppleCook	Bottle-6[1]	938	-	842	1084	958	1001	1286	1218	1299	1442	1268	1279	1255	3.51	2.42
AppleCook	Shuffle1[1]	938	655	842	1055	971	997	1280	1219	1295	1437	1294	1268	1276	3.25	2.93
AppleCook	Shuffle2[1]	938	655	842	1046	965	992	1269	1191	1275	1422	1267	1257	1238	2.14	2.23
StorerWV	PS10[1]	976	-	-	-	-	-	-	-	-	-	-	-	-		
StorerWV	HSL10[1]	1006	-	-	-	-	-	-	-	-	-	-	-	-		
AartsVLU	TA1^{m5}	1003	693	925	1104	1014	1075	1289	1262	1385	1469	1323	1305	1295	7.93	2.64
VLaarAL	SA^{m5}	985	663	853	1067	966	1004	1273	1226	1300	1442	1227	1258	1247	3.32	2.20
VLaarAL	SA^{b5}	951	655	848	1063	952	992	1269	1218	1293	1433	1215	1248	1234	2.26	2.11
AartsVLU	SA1^{m5}	969	669	855	1083	962	1003	1282	1233	1307	1440	1235	1258	1256	3.59	2.14
AartsVLU	SA1$^{1,t\to\infty}$	-	-	-	1053	935	983	1249	1185	-	-	1208	-	1225	1.07	1.55
AartsVLU	SA2^{m5}	977	658	854	1078	960	1019	1275	1225	1308	1451	1243	1263	1254	3.63	2.16
MatsuoSS	CSSA[1]	946	655	842	1071	973	991	1274	1196	1292	1435	1231	1251	1235	2.40	1.86
Taillard	TS1^{b5}	930	-	-	1047	-	-	1240	1170	-	-	1202	-	-		
BarnesC	TS2[1]	935	655	843	1053	946	988	1256	1194	1278	1418	1211	1237	1239	1.45	1.68
Dell'AT	TS3^{m5}	935	655	846	1057	943	980	1252	1194	1289	1423	1210	1254	1235	1.56	1.66
Dell'AT	TS3^{b5}	935	655	842	1048	941	979	1242	1182	1278	1409	1203	1242	1233	1.01	1.42
NowickiS	TSAB[1]	930	655	842	1055	948	988	1259	1164	1275	1422	1209	1235	1234	1.19	1.12
NowickiS	TSAB^{b3}	930	655	842	1047	939	977	1236	1160	1268	1407	1196	1233	1229	0.54	0.98
YamadaN	GA1^{b600}	930	-	-	-	-	-	-	-	-	-	-	-	-		
AartsVLU	GLS1^{m5}	978	668	863	1084	970	1016	1303	1290	1324	1449	1285	1279	1273	5.14	3.40
AartsVLU	GLS2^{m5}	982	659	859	1085	981	1010	1300	1260	1310	1450	1283	1279	1260	4.69	2.99
AartsVLU	GLS2$^{1,t\to\infty}$	-	-	-	1055	938	985	1265	1217	-	-	1248	-	1233		
DellaTV	GA2^{m3}	965	-	-	1113	-	-	-	-	1330	-	-	-	-		
DellaTV	GA2^{b3}	946	-	-	1097	-	-	-	-	1305	-	-	-	-		
NakanoY	GA3[1]	965	-	-	-	-	-	-	-	-	-	-	-	-		
DornPesch	P-GA[1]	960	681	880	1139	1014	1014	1378	1336	1373	1498	1296	1351	1321	8.23	4.24
DornPesch	SB-GA(40)^{m2}	938	666	863	1074	960	1008	1272	1204	1317	1484	1251	1282	1274	3.74	1.84
DornPesch	SB-GA(60)^{m2}	-	-	848	1074	957	1007	1269	1210	1317	1446	1241	1277	1252	3.49	1.82
NuijtAVV	RCS^{b5}	930	655	843	1069	942	981	1285	1231	1292	1411	1278	1238	1247	2.41	3.16

Table 3: Performance comparison of various algorithms: computation times for 13 instances

Author	Method	FT10	LA2	LA19	LA21	LA24	LA25	LA27	LA29	LA36	LA37	LA38	LA39	LA40	SCT	computer	CISCT
AdamsBZ	SBI[1]	10.10	1.69	7.40	2.19	25.5	27.9	45.5	48.0	46.9	61.4	57.7	71.8	76.7	482.8	VAX 780/11	$6.0 \cdot 10^1$
BalasLV	SB3[1]	5.82	0.90	4.33	10.92	10.82	13.00	19.42	21.10	27.80	26.25	29.63	25.40	26.23	221.6	Sparc 330	$5.5 \cdot 10^2$
BalasLV	SB4[1]	11.17	1.43	8.80	19.88	19.82	22.70	37.90	39.00	55.83	53.28	59.26	50.63	52.41	432.1	Sparc 330	$1.1 \cdot 10^3$
AdamsBZ	SBII[1]	851	12.5	240	362	434	430	837	892	735	837	1079	669	899	10742	VAX 780/11	$1.3 \cdot 10^3$
AppleCook	Bottle-4[1]	6.8	1.3	10.5	17.5	25.9	21.6	31.4	31.5	22.6	13.7	46.5	41.7	22.4	293.4	Sparc st'n ELC	$7.3 \cdot 10^2$
AppleCook	Bottle-5[1]	7.1	7.8	64.8	46.4	62.7	48.5	92.1	91.4	152.9	56.4	95.7	134.3	23.7	883.8	Sparc st'n ELC	$2.2 \cdot 10^3$
AppleCook	Bottle-6[1]	7.6	-	201.5	300.7	200.0	100.4	666.5	280.4	320.7	561.7	181.6	191.9	154.1	3175	Sparc st'n ELC	$7.9 \cdot 10^3$
AppleCook	Shuffle1[1]	24.7	7.9	72.7	954.8	421.4	74.2	97.6	94.6	170.9	64.0	103.7	178.1	42.6	2307	Sparc st'n ELC	$5.8 \cdot 10^3$
AppleCook	Shuffle2[1]	24.7	7.9	72.7	87478	65422	98.2	604.2	15358	3348	1577	17799	6745	150.1	198685	Sparc st'n ELC	$5.0 \cdot 10^5$
StorerWV	PS10[1]		-	-	-	-	-	-	-	-	-	-	-	-			
StorerWV	HSL10[1]		-	-	-	-	-	-	-	-	-	-	-	-			
AartsVLU	TA1^{m5}	99.4	18.6	93.8	243.4	234.8	254.8	492.0	471.0	602.2	636.2	635.6	592.2	596.8	4971	VAX 8650	$3.5 \cdot 10^3$
VLaarAL	SA^{m5}	779	117	830	1991	2098	2133	4535	4408	5346	5287	5480	5766	5373	44143	VAX 785	$8.4 \cdot 10^3$
VLaarAL	SA^{b5}	3895	585	4150	9955	10490	10665	22675	22040	26730	26435	27400	28830	26865	220715	VAX 785	$4.2 \cdot 10^4$
AartsVLU	SA1^{m5}	99.4	18.6	93.8	243.4	234.8	254.8	492.0	471.0	602.2	636.2	635.6	592.2	596.8	4971	VAX 8650	$3.5 \cdot 10^3$
AartsVLU	SA1$^{1,t\to\infty}$	-	-	-						-	-		-		$4 \cdot 10^4$	VAX 8650	$3 \cdot 10^4$
AartsVLU	SA2^{m5}	99.4	18.6	93.8	243.4	234.8	254.8	492.0	471.0	602.2	636.2	635.6	592.2	596.8	4971	VAX 8650	$3.5 \cdot 10^3$
MatsuoSS	CSSA[1]	987	3.03	115	205	199	180	286	267	624	577	672	660	603	5378	VAX 780/11	$6.7 \cdot 10^2$
Taillard	TS1^{b5}		-	-		-	-			-	-		-	-			
BarnesC	TS2[1]	15.8	28.9	217.1	173.5	27.7	176.1	248.2	195.6	221.4	232.0	180.9	258.7	89.5	2065	IBM RS 6000	$2.5 \cdot 10^4$
Dell'AT	TS3^{m5}	155.8	18.8	103.8	198.8	181.8	191.7	254.2	281.3	238.4	242.2	256.6	237.8	236.6	2598	PC 386	$1.3 \cdot 10^3$
Dell'AT	TS3^{b5}	779.0	94.0	519.0	994.0	909.0	958.5	1271	1407	1192	1211	1283	1189	1183	12989	PC 386	$6.5 \cdot 10^3$
NowickiS	TSAB[1]	30	8	60	21	184	155	66	493	623	443	165	325	322	2895	AT 386 DX	$1.4 \cdot 10^3$
NowickiS	TSAB^{b3}														8685	AT 386 DX	$4.3 \cdot 10^3$
YamadaN	GA1^{b600}	$36 \cdot 10^5$	-	-	-	-	-	-	-	-	-	-	-	-		Sparc st'n 2	
AartsVLU	GLS1^{m5}	99.4	18.6	93.8	243.4	234.8	254.8	492.0	471.0	602.2	636.2	635.6	592.2	596.8	4971	VAX 8650	$3.5 \cdot 10^3$
AartsVLU	GLS2^{m5}	99.4	18.6	93.8	243.4	234.8	254.8	492.0	471.0	602.2	636.2	635.6	592.2	596.8	4971	VAX 8650	$3.5 \cdot 10^3$
AartsVLU	GLS2$^{1,t\to\infty}$	-	-	-						-	-		-		$4 \cdot 10^4$	VAX 8650	$3 \cdot 10^4$
DellaTV	GA2^{m3}	628	-	-	1062	-	-	-	-	1880	-	-	-	-		PC 486/25	
DellaTV	GA2^{b3}	1884	-	-	3186	-	-	-	-	5640	-	-	-	-		PC 486/25	
NakanoY	GA3[1]		-	-	-	-	-	-	-	-	-	-	-	-			
DornPesch	P-GA[1]	932.6	108	191	352	352	350	565	570	524	520	525	525	526	6041	DEC station 3100	$9.7 \cdot 10^3$
DornPesch	SB-GA(40)^{m2}	106.7	16.4	77.4	134.8	137.3	134.2	242.5	241.0	335.6	350.5	335.7	327.2	348.0	2787	DEC station 3100	$4.5 \cdot 10^3$
DornPesch	SB-GA(60)^{m2}	-	-	161.3	292.8	289.0	228.9	446.2	453.1	688.1	665.9	665.9	687.5	698.4	5523	DEC station 3100	$8.8 \cdot 10^3$
NuijtAVV	RCS^{b5}	1279	64	1176	4313	2610	5167	5781	3336	6691	3860	5747	5059	4208	49291	Sparc st'n ELC	$1.2 \cdot 10^5$

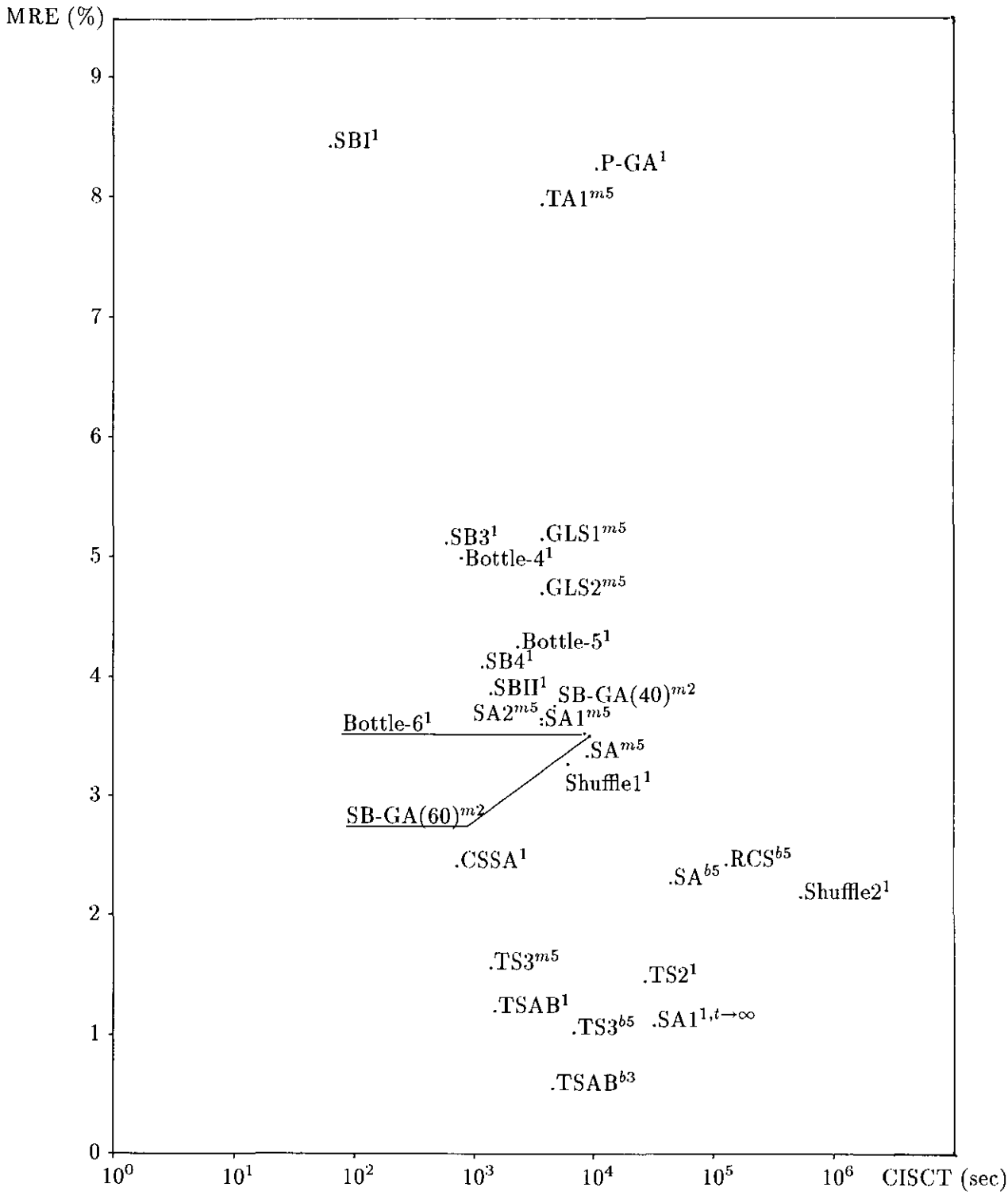

Figure 1: Relation between mean relative error and computer independent sum of CPU-times

algorithm of Aarts, Van Laarhoven, Lenstra & Ulder [1994] and the iterative improvement algorithm Shuffle of Applegate & Cook [1991].

Regarding *iterative improvement*, Aarts, Van Laarhoven, Lenstra & Ulder [1994] report that their multi-start algorithm is inferior to threshold accepting and simulated annealing.

Applegate & Cook's Shuffle algorithm works well, due to a neighborhood function that allows major changes in the schedule. We used our own outcomes of Bottle-5 as start solutions. The number t of machines to fix was chosen such that edge finder could rapidly fill in the remainder of the schedule. We set t=1 for the instances FT10, LA2 and LA19, t=2 for LA21, LA24 and LA24, and t=5 for the other instances. The results for these values of t are reported under Shuffle1. We also carried out some more time consuming runs with t=1 for FT10 and LA2-LA24, t=3 for LA29, LA36 and LA37, and t=4 for LA27, LA38, LA39 and LA40. The outcomes, reported under Shuffle2, are good but expensive.

Storer, Wu, & Vaccari [1992] give very few computational results for their variants of iterative improvement. Their results for the instance FT10 are poor. It seems that their search strategy or their neighborhood function is not powerful enough.

The *threshold accepting* algorithm of Aarts, Van Laarhoven, Lenstra & Ulder [1994] competes with their simulated annealing algorithm in case simulated annealing finds an optimal schedule. Otherwise, threshold accepting is outperformed by simulated annealing. Almost all instances in our table belong to the latter category.

The *simulated annealing* algorithm of Van Laarhoven, Aarts & Lenstra [1992] produces reasonable results. Five runs on FT10 with a standard setting of the cooling parameters produced an average schedule length of 985.8, with a minimum of 951; a much slower cooling schedule yields solution values of 930 (twice), 934, 935 and 938. Reasonable results are also obtained by the simulated annealing algorithm of Aarts, Van Laarhoven, Lenstra & Ulder [1994] with a standard cooling schedule, but an extremely slow cooling schedule gives very good results. To compute MRE and SRE for the latter cooling schedule, we estimated the values for the missing entries. It is remarkable that their algorithm with the standard cooling schedule has a similar behaviour for the neighborhood functions N_1 (SA1) and N_{5a} (SA2). Good results are obtained by the simulated annealing variant CSSA of Matsuo, Suh, & Sullivan [1988]. In comparison to other approximative approaches, simulated annealing may require large running times, but it yields consistently good solutions with a modest amount of human implementation effort and relatively little insight into the combinatorial structure of the problem type under consideration.

The advent of *taboo search* has changed the picture. Methods of this type produce excellent solutions in reasonable times, although these benefits come at the expense of a non-trivial amount of testing and tuning. Although few data are available, Taillard's [1994] algorithm TS1 seems to perform extremely well. Also very good results are obtained by algorithm TS2 of Barnes & Chambers [1991]. Dell'Amico & Trubian's [1993] algorithm TS3 obtained even better results. Apparently, their complicated neighborhood function is very effective. The algorithm TSAB of Nowicki & Smutnicki [1993], which applies taboo search and traces its way back to promising but rejected changes, is the current champion for job shop scheduling. For our 13 instances it achieves a mean relative error of only 0.54 %.

For many *genetic algorithms* no results for our 13 instances are available. Sometimes, only the result for FT10 is given. Yamada & Nakano [1992] found a schedule of length 930 four times among 600 trials. They also tested their algorithm GA1 on four 20-job 20-machine instances, but their outcomes are on average 5.9 % above the best known upper bounds [Wennink, 1994]. The results obtained by Aarts, Van Laarhoven, Lenstra & Ulder [1994] are

not very strong. Their algorithm GLS2 (using neighborhood function N_{5a}) performs slightly better than GLS1 (using N_1).

As for genetic algorithms using string representations, the results obtained by Della Croce, Tadei, & Volta's [1994] algorithm GA2 and by Nakano & Yamada's [1991] algorithm GA3 are poor. The algorithm P-GA of Dorndorf & Pesch [1994] is even worse. Their algorithm SB-GA, which incorporates a shifting bottleneck variant, produces reasonable results. Values are reported for runs with population sizes of 40 and 60.

The *constraint satisfaction* algorithm of Nuijten, Aarts, Van Erp Taalman Kip & Van Hee [1993] produces good results but needs a lot of time. For the *neural network* approaches no computational results are available that allow a proper comparison with other techniques.

10 Conclusion

10.1 Review

Current optimization algorithms for job shop scheduling can handle problem instances no harder than the 15×15 instances of Lawrence in reasonable amounts of running time. If one wants to obtain approximate solutions to larger instances, one has to resort to local search.

From the local search algorithms discussed in this survey, taboo search seems to work best. For the 13 instances investigated, the algorithm of Nowicki & Smutnicki, which combines taboo search with backtracking, outperforms the other local search algorithms developed so far. Also the implementations of taboo search by Dell'Amico & Trubian and of simulated annealing by Aarts, Van Laarhoven, Lenstra, & Ulder perform very well, but the latter only if large running times are allowed.

The various shifting bottleneck procedures produce schedules of moderate quality. Better results are obtained in combination with some type of local search or backtracking.

Genetic algorithms have a poor performance until now. Often the neighborhood function applied in combination with the schedule representation chosen does not generate meaningful changes and it is hard to find improvements. Only when some kind of standard local search is embedded at a second level, the computational results are satisfactory.

10.2 Preview

There is still considerable room for improving local search approaches to the job shop scheduling problem. As shown in Figure 1, none of the existing algorithms achieves an average error of less than 2% within 100 seconds. And our benchmark instances are still small ones.

We have observed that many approaches operate at two levels, with, for instance, schedule construction, partial enumeration or local search with big changes at the top level, and local search with smaller changes at the bottom level. Such hybrid approaches are in need of a more systematic investigation. The type of backtracking proposed by Nowicki & Smutnicki is a promising technique and can be combined with almost any local search algorithm without difficulties. It might also be interesting to design a three-level approach with neighborhoods of smaller size towards the bottom.

The flexibility of local search and the results reported here provide a promising basis for the application of local search to more general scheduling problems. An example of practical interest is the multiprocessor job shop, where each production stage has a set of parallel

machines rather than a single one. Finding a schedule involves assigment as well as sequencing decisions. This is a difficult problem, for which no effective solution methods exist.

Applying local search to large instances of scheduling problems requires the design of data structures that allow fast incremental computations of, for example, longest paths. Johnson [1990] has shown that sophisticated data structures play an important role in the application of local search to large traveling salesman problems.

Our survey has been predominantly of a computational nature. There are several related theoretical questions about the complexity of local search. A central concept in this respect is PLS-completeness [Johnson, Papadimitriou, & Yannakakis, 1988]. Many of the neighbourhood functions defined in Section 5 define a PLS-problem, which may be PLS-complete. There are also complexity issues regarding the parallel execution of local search. For example, for some of the neighborhood functions it may be possible to verify local optimality in polylog parallel time.

References

E.H.L. Aarts, P.J.M. van Laarhoven, J.K. Lenstra, N.L.J. Ulder (1994), A computational study of local search algorithms for job shop scheduling, *ORSA J. Comput.*, to appear.

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

S.B. Akers (1956), A graphical approach to production scheduling problems, *Oper. Res. 4*, 244-245.

D. Applegate, W. Cook (1991), A computational study of the job-shop scheduling problem, *ORSA J. Comput. 3*, 149-156.

D. Applegate, W. Cook (1993), Personal communication.

K.R. Baker, Z.-S. Su (1974), Sequencing with due-dates and early start times to minimize maximum tardiness, *Naval Res. Logist. Quart. 21*, 171-176.

E. Balas (1969), Machine sequencing via disjunctive graphs: an implicit enumeration algorithm, *Oper. Res. 17*, 941-957.

E. Balas (1985), On the facial structure of scheduling polyhedra, *Math. Programming Stud. 24*, 179-218.

E. Balas, J.K. Lenstra, A. Vazacopoulos (1994), One machine scheduling with delayed precedence constraints, *Management Sci.*, to appear.

J.W. Barnes, J.B. Chambers (1994), Solving the job shop scheduling problem using tabu search, *IIE Trans.*, to appear.

P. Bratley, M. Florian, P. Robillard (1973), On sequencing with earliest starts and due dates with application to computing bounds for the $(n/m/G/F_{\max})$ problem, *Naval Res. Logist. Quart. 20*, 57-67.

P. Brucker (1988), An efficient algorithm for the job-shop problem with two jobs, *Computing 40*, 353-359.

P. Brucker, B. Jurisch, B. Sievers (1994), A branch & bound algorithm for the job-shop scheduling problem, *Discrete Appl. Math.*, to appear.

J. Carlier (1982), The one-machine sequencing problem, *European J. Oper. Res. 11*, 42-47.

J. Carlier, E. Pinson (1989), An algorithm for solving the job-shop problem, *Management Sci. 35*, 164-176.

J. Carlier, E. Pinson (1990), A practical use of Jackson's preemptive schedule for solving the job-shop problem, *Ann. Oper. Res. 26*, 269-287.

V. Černy (1985), Thermodynamical approach to the traveling salesman problem, *J. Optim. Theory Appl. 45*, 41-51.

L. Davis (1985), Job shop scheduling with genetic algorithms. J.J. Grefenstette (ed.) (1985), *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, Carne-

gie-Mellon University, Pittsburgh, Pennsylvania, 136-140.

S. DAUZÈRE-PERES, J.-B. LASSERRE (1993), A modified shifting bottleneck procedure for job-shop scheduling, *Int. J. Prod. Res. 31*, 923-932.

F. DELLA CROCE, R. TADEI, G. VOLTA (1994), A genetic algorithm for the job shop problem, *Comput. Oper. Res.*, to appear.

M. DELL'AMICO, M. TRUBIAN (1993), Applying tabu search to the job-shop scheduling problem, *Ann. Oper. Res. 41*, 231-252.

J.J. DONGARRA (1993), *Performance of various computers using standard linear equations software*, Report CS-89-85, Computer Science Department, University of Tennessee, Knoxville, Tennessee.

U. DORNDORF, E. PESCH (1994), Evolution based learning in a job shop scheduling environment, *Comput. Oper. Res.*, to appear.

G. DUECK, T. SCHEUER (1990), Threshold accepting; a general purpose optimization algorithm, *J. Comput. Phys. 90*, 161-175.

E. FALKENAUER, S. BOUFFOUIX (1991), A genetic algorithm for job shop, *Proceedings of the 1991 IEEE Internationial Conference on Robotics and Automation*, IEEE Computer Society Press, Los Alamitos, California, 824-829.

M.L. FISHER, B.J. LAGEWEG, J.K. LENSTRA, A.H.G. RINNOOY KAN (1983), Surrogate duality relaxation for job shop scheduling, *Discrete Appl. Math. 5*, 65-75.

H. FISHER, G.L. THOMPSON (1963), Probabilistic learning combinations of local job-shop scheduling rules. J.F. MUTH, G.L. THOMPSON (eds.) (1963), *Industrial Scheduling*, Prentice Hall, Englewood Cliffs, New Jersey, 225-251.

Y.P.S. FOO, Y. TAKEFUJI (1988a), Stochastic neural networks for solving job-shop scheduling: part 1. Problem representation, *IEEE International Conference on Neural Networks*, IEEE San Diego section & IEEE TAB Neural Network Committee, San Diego, California, 275-282.

Y.P.S. FOO, Y. TAKEFUJI (1988b), Stochastic neural networks for solving job-shop scheduling: part 2. Architecture and simulations, *IEEE International Conference on Neural Networks*, IEEE San Diego section & IEEE TAB Neural Network Committee, San Diego, California, 283-290.

B. GIFFLER, G.L. THOMPSON (1960), Algorithms for solving production scheduling problems, *Oper. Res. 8*, 487-503.

F. GLOVER (1989), Tabu Search - Part I, *ORSA J. Comput. 1*, 190-206.

F. GLOVER (1990), Tabu Search - Part II, *ORSA J. Comput. 2*, 4-32.

F. GLOVER, E. TAILLARD, D. DE WERRA (1993), A user's guide to tabu search, *Ann. Oper. Res. 41*, 3-28.

D.E. GOLDBERG (1989), *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, Massachusetts.

R. HAUPT (1989), A survey of priority rule-based scheduling, *OR Spektrum 11*, 3-16.

N. HEFETZ, I. ADIRI (1982), An efficient optimal algorithm for the two-machines unit-time jobshop schedule-length problem, *Math. Oper. Res. 7*, 354-360.

J.H. HOLLAND (1975), *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor, Michigan.

J.J. HOPFIELD, D.W. TANK (1985), Neural computation of decisions in optimization problems, *Biol. Cybernet. 55*, 141-152.

J.R. JACKSON (1956), An extension of Johnson's results on job lot scheduling, *Naval Res. Logist. Quart. 3*, 201-203.

D.S. JOHNSON (1990), Data structures for traveling salesmen, J.R. GILBERT, R. KARLSSON (eds.) (1990), *SWAT90, 2nd Scandinavian Workshop on Algorithm Theory*, Springer, Berlin, 287-305.

D.S. JOHNSON, C.H. PAPADIMITRIOU, M. YANNAKAKIS (1988), How easy is local search?, *J. Comput. System Sci. 37*, 79-100.

S. KIRKPATRICK, C.D. GELATT, JR., M.P. VECCHI (1983), Optimization by simulated annealing, *Science 220*, 671-680.

B.J. LAGEWEG, J.K. LENSTRA, A.H.G. RINNOOY KAN (1976), Minimizing maximum lateness on one machine: computational experience and some applications, *Statist. Neerlandica 30*, 25-41.

B.J. LAGEWEG, J.K. LENSTRA, A.H.G. RINNOOY KAN (1977), Job-shop scheduling by implicit

enumeration, *Management Sci. 24*, 441-450.

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

J.K. LENSTRA, A.H.G. RINNOOY KAN (1979), Computational complexity of discrete optimization problems, *Ann. Discrete Math. 4*, 121-140.

J.K. LENSTRA, A.H.G. RINNOOY KAN, P. BRUCKER (1977), Complexity of machine scheduling problems, *Ann. Discrete Math. 1*, 343-362.

H. MATSUO, C.J. SUH, R.S. SULLIVAN (1988), *A controlled search simulated annealing method for the general jobshop scheduling problem*, Working paper 03-04-88, Graduate School of Business, University of Texas, Austin.

G.B. MCMAHON, M. FLORIAN (1975), On scheduling with ready times and due dates to minimize maximum lateness, *Oper. Res. 23*, 475-482.

R. NAKANO, T. YAMADA (1991), Conventional genetic algorithm for job shop problems, R.K. BELEW, L.B. BOOKER (eds.) (1991), *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Diego, California, 474-479.

L. NÉMETI (1964), Das Reihenfolgeproblem in der Fertigungsprogrammierung und Linearplanung mit logischen Bedingungen, *Mathematica (Cluj) 6*, 87-99.

E. NOWICKI, C. SMUTNICKI (1993), *A fast taboo search algorithm for the job shop problem*, Preprint 8/93, Institute of Engineering Cybernetics, Technical University of Wroclaw.

W.P.M. NUIJTEN, E.H.L. AARTS, D.A.A. VAN ERP TAALMAN KIP, K.M. VAN HEE (1993), *Job shop scheduling by constraint satisfaction*, Computing Science Note 93/39, Department of Mathematics and Computing Science, Eindhoven University of Technology, Eindhoven.

B. ROY, B. SUSSMANN (1964), *Les problèmes d'ordonnancement avec constraints disjonctives*, Note DS No. 9 bis, SEMA, Montrouge.

N. SADEH (1991), *Look-ahead Techniques for Micro-opportunistic Job Shop Scheduling*, Ph.D. thesis, School of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania.

S.V. SEVAST'JANOV (1994), Scheduling problems and vectors summing algorithms: a survey, *Discrete Appl. Math.*, to appear.

D.B. SHMOYS, C. STEIN, J. WEIN (1994), Improved approximation algorithms for shop scheduling problems, *ORSA J. Comput.*, to appear.

Y.N. SOTSKOV (1991), The complexity of shop-scheduling problems with two or three jobs, *European J. Oper. Res. 53*, 326-336.

R.H. STORER, S.D. WU, R. VACCARI (1992), New search spaces for sequencing problems with application to job shop scheduling, *Management Sci. 38*, 1495-1509.

E. TAILLARD (1994), Parallel taboo seach technique for the jobshop scheduling problem, *ORSA J. Comput.*, to appear.

F. TIOZZO (1988), *Building a decision support system for operation scheduling in a large industrial department: a preliminary algorithmic study*, Internal report, Department of Mathematics and Informatics, University of Udine, Italy.

P.J.M. VAN LAARHOVEN, E.H.L. AARTS, J.K. LENSTRA (1992), Job shop scheduling by simulated annealing, *Oper. Res. 40*, 113-125.

M. WENNINK (1994), Personal communication.

D.P. WILLIAMSON, L.A. HALL, J.A. HOOGEVEEN, C.A.J. HURKENS, J.K. LENSTRA, D.B. SHMOYS (1994), *Short shop schedules*, COSOR Memorandum 94-06, Department of Mathematics and Computing Science, Eindhoven University of Technology, Eindhoven.

T. YAMADA, R. NAKANO (1992), A genetic algorithm applicable to large-scale job-shop problems, R. MÄNNER, B. MANDERICK (eds.) (1992), *Parallel Problem Solving from Nature, 2*, North-Holland, Amsterdam, 281-290.

D.N. ZHOU, V. CHERKASSKY, T.R. BALDWIN, D.E. OLSON (1991), A neural network approach to job-shop scheduling, *IEEE Trans. Neural Networks 2*, 175-179.

Computing Science Notes

Department of Mathematics and Computing Science
Eindhoven University of Technology

In this series appeared:

91/01	D. Alstein	Dynamic Reconfiguration in Distributed Hard Real-Time Systems, p. 14.
91/02	R.P. Nederpelt H.C.M. de Swart	Implication. A survey of the different logical analyses "if...,then...", p. 26.
91/03	J.P. Katoen L.A.M. Schoenmakers	Parallel Programs for the Recognition of *P*-invariant Segments, p. 16.
91/04	E. v.d. Sluis A.F. v.d. Stappen	Performance Analysis of VLSI Programs, p. 31.
91/05	D. de Reus	An Implementation Model for GOOD, p. 18.
91/06	K.M. van Hee	SPECIFICATIEMETHODEN, een overzicht, p. 20.
91/07	E.Poll	CPO-models for second order lambda calculus with recursive types and subtyping, p. 49.
91/08	H. Schepers	Terminology and Paradigms for Fault Tolerance, p. 25.
91/09	W.M.P.v.d.Aalst	Interval Timed Petri Nets and their analysis, p.53.
91/10	R.C.Backhouse P.J. de Bruin P. Hoogendijk G. Malcolm E. Voermans J. v.d. Woude	POLYNOMIAL RELATORS, p. 52.
91/11	R.C. Backhouse P.J. de Bruin G.Malcolm E.Voermans J. van der Woude	Relational Catamorphism, p. 31.
91/12	E. van der Sluis	A parallel local search algorithm for the travelling salesman problem, p. 12.
91/13	F. Rietman	A note on Extensionality, p. 21.
91/14	P. Lemmens	The PDB Hypermedia Package. Why and how it was built, p. 63.
91/15	A.T.M. Aerts K.M. van Hee	Eldorado: Architecture of a Functional Database Management System, p. 19.
91/16	A.J.J.M. Marcelis	An example of proving attribute grammars correct: the representation of arithmetical expressions by DAGs, p. 25.

91/17	A.T.M. Aerts P.M.E. de Bra K.M. van Hee	Transforming Functional Database Schemes to Relational Representations, p. 21.
91/18	Rik van Geldrop	Transformational Query Solving, p. 35.
91/19	Erik Poll	Some categorical properties for a model for second order lambda calculus with subtyping, p. 21.
91/20	A.E. Eiben R.V. Schuwer	Knowledge Base Systems, a Formal Model, p. 21.
91/21	J. Coenen W.-P. de Roever J.Zwiers	Assertional Data Reification Proofs: Survey and Perspective, p. 18.
91/22	G. Wolf	Schedule Management: an Object Oriented Approach, p. 26.
91/23	K.M. van Hee L.J. Somers M. Voorhoeve	Z and high level Petri nets, p. 16.
91/24	A.T.M. Aerts D. de Reus	Formal semantics for BRM with examples, p. 25.
91/25	P. Zhou J. Hooman R. Kuiper	A compositional proof system for real-time systems based on explicit clock temporal logic: soundness and complete ness, p. 52.
91/26	P. de Bra G.J. Houben J. Paredaens	The GOOD based hypertext reference model, p. 12.
91/27	F. de Boer C. Palamidessi	Embedding as a tool for language comparison: On the CSP hierarchy, p. 17.
91/28	F. de Boer	A compositional proof system for dynamic proces creation, p. 24.
91/29	H. Ten Eikelder R. van Geldrop	Correctness of Acceptor Schemes for Regular Languages, p. 31.
91/30	J.C.M. Baeten F.W. Vaandrager	An Algebra for Process Creation, p. 29.
91/31	H. ten Eikelder	Some algorithms to decide the equivalence of recursive types, p. 26.
91/32	P. Struik	Techniques for designing efficient parallel programs, p. 14.
91/33	W. v.d. Aalst	The modelling and analysis of queueing systems with QNM-ExSpect, p. 23.
91/34	J. Coenen	Specifying fault tolerant programs in deontic logic, p. 15.

91/35	F.S. de Boer J.W. Klop C. Palamidessi	Asynchronous communication in process algebra, p. 20.
92/01	J. Coenen J. Zwiers W.-P. de Roever	A note on compositional refinement, p. 27.
92/02	J. Coenen J. Hooman	A compositional semantics for fault tolerant real-time systems, p. 18.
92/03	J.C.M. Baeten J.A. Bergstra	Real space process algebra, p. 42.
92/04	J.P.H.W.v.d.Eijnde	Program derivation in acyclic graphs and related problems, p. 90.
92/05	J.P.H.W.v.d.Eijnde	Conservative fixpoint functions on a graph, p. 25.
92/06	J.C.M. Baeten J.A. Bergstra	Discrete time process algebra, p.45.
92/07	R.P. Nederpelt	The fine-structure of lambda calculus, p. 110.
92/08	R.P. Nederpelt F. Kamareddine	On stepwise explicit substitution, p. 30.
92/09	R.C. Backhouse	Calculating the Warshall/Floyd path algorithm, p. 14.
92/10	P.M.P. Rambags	Composition and decomposition in a CPN model, p. 55.
92/11	R.C. Backhouse J.S.C.P.v.d.Woude	Demonic operators and monotype factors, p. 29.
92/12	F. Kamareddine	Set theory and nominalisation, Part I, p.26.
92/13	F. Kamareddine	Set theory and nominalisation, Part II, p.22.
92/14	J.C.M. Baeten	The total order assumption, p. 10.
92/15	F. Kamareddine	A system at the cross-roads of functional and logic programming, p.36.
92/16	R.R. Seljée	Integrity checking in deductive databases; an exposition, p.32.
92/17	W.M.P. van der Aalst	Interval timed coloured Petri nets and their analysis, p. 20.
92/18	R.Nederpelt F. Kamareddine	A unified approach to Type Theory through a refined lambda-calculus, p. 30.
92/19	J.C.M.Baeten J.A.Bergstra S.A.Smolka	Axiomatizing Probabilistic Processes: ACP with Generative Probabilities, p. 36.
92/20	F.Kamareddine	Are Types for Natural Language? P. 32.

92/21	F.Kamareddine	Non well-foundedness and type freeness can unify the interpretation of functional application, p. 16.
92/22	R. Nederpelt F.Kamareddine	A useful lambda notation, p. 17.
92/23	F.Kamareddine E.Klein	Nominalization, Predication and Type Containment, p. 40.
92/24	M.Codish D.Dams Eyal Yardeni	Bottum-up Abstract Interpretation of Logic Programs, p. 33.
92/25	E.Poll	A Programming Logic for Fω, p. 15.
92/26	T.H.W.Beelen W.J.J.Stut P.A.C.Verkoulen	A modelling method using MOVIE and SimCon/ExSpect, p. 15.
92/27	B. Watson G. Zwaan	A taxonomy of keyword pattern matching algorithms, p. 50.
93/01	R. van Geldrop	Deriving the Aho-Corasick algorithms: a case study into the synergy of programming methods, p. 36.
93/02	T. Verhoeff	A continuous version of the Prisoner's Dilemma, p. 17
93/03	T. Verhoeff	Quicksort for linked lists, p. 8.
93/04	E.H.L. Aarts J.H.M. Korst P.J. Zwietering	Deterministic and randomized local search, p. 78.
93/05	J.C.M. Baeten C. Verhoef	A congruence theorem for structured operational semantics with predicates, p. 18.
93/06	J.P. Veltkamp	On the unavoidability of metastable behaviour, p. 29
93/07	P.D. Moerland	Exercises in Multiprogramming, p. 97
93/08	J. Verhoosel	A Formal Deterministic Scheduling Model for Hard Real-Time Executions in DEDOS, p. 32.
93/09	K.M. van Hee	Systems Engineering: a Formal Approach Part I: System Concepts, p. 72.
93/10	K.M. van Hee	Systems Engineering: a Formal Approach Part II: Frameworks, p. 44.
93/11	K.M. van Hee	Systems Engineering: a Formal Approach Part III: Modeling Methods, p. 101.
93/12	K.M. van Hee	Systems Engineering: a Formal Approach Part IV: Analysis Methods, p. 63.
93/13	K.M. van Hee	Systems Engineering: a Formal Approach Part V: Specification Language, p. 89.

93/14	J.C.M. Baeten J.A. Bergstra	On Sequential Composition, Action Prefixes and Process Prefix, p. 21.
93/15	J.C.M. Baeten J.A. Bergstra R.N. Bol	A Real-Time Process Logic, p. 31.
93/16	H. Schepers J. Hooman	A Trace-Based Compositional Proof Theory for Fault Tolerant Distributed Systems, p. 27
93/17	D. Alstein P. van der Stok	Hard Real-Time Reliable Multicast in the DEDOS system, p. 19.
93/18	C. Verhoef	A congruence theorem for structured operational semantics with predicates and negative premises, p. 22.
93/19	G-J. Houben	The Design of an Online Help Facility for ExSpect, p.21.
93/20	F.S. de Boer	A Process Algebra of Concurrent Constraint Programming, p. 15.
93/21	M. Codish D. Dams G. Filé M. Bruynooghe	Freeness Analysis for Logic Programs - And Correctness?, p. 24.
93/22	E. Poll	A Typechecker for Bijective Pure Type Systems, p. 28.
93/23	E. de Kogel	Relational Algebra and Equational Proofs, p. 23.
93/24	E. Poll and Paula Severi	Pure Type Systems with Definitions, p. 38.
93/25	H. Schepers and R. Gerth	A Compositional Proof Theory for Fault Tolerant Real-Time Distributed Systems, p. 31.
93/26	W.M.P. van der Aalst	Multi-dimensional Petri nets, p. 25.
93/27	T. Kloks and D. Kratsch	Finding all minimal separators of a graph, p. 11.
93/28	F. Kamareddine and R. Nederpelt	A Semantics for a fine λ-calculus with de Bruijn indices, p. 49.
93/29	R. Post and P. De Bra	GOLD, a Graph Oriented Language for Databases, p. 42.
93/30	J. Deogun T. Kloks D. Kratsch H. Müller	On Vertex Ranking for Permutation and Other Graphs, p. 11.
93/31	W. Körver	Derivation of delay insensitive and speed independent CMOS circuits, using directed commands and production rule sets, p. 40.
93/32	H. ten Eikelder and H. van Geldrop	On the Correctness of some Algorithms to generate Finite Automata for Regular Expressions, p. 17.

93/33	L. Loyens and J. Moonen	ILIAS, a sequential language for parallel matrix computations, p. 20.
93/34	J.C.M. Baeten and J.A. Bergstra	Real Time Process Algebra with Infinitesimals, p.39.
93/35	W. Ferrer and P. Severi	Abstract Reduction and Topology, p. 28.
93/36	J.C.M. Baeten and J.A. Bergstra	Non Interleaving Process Algebra, p. 17.
93/37	J. Brunekreef J-P. Katoen R. Koymans S. Mauw	Design and Analysis of Dynamic Leader Election Protocols in Broadcast Networks, p. 73.
93/38	C. Verhoef	A general conservative extension theorem in process algebra, p. 17.
93/39	W.P.M. Nuijten E.H.L. Aarts D.A.A. van Erp Taalman Kip K.M. van Hee	Job Shop Scheduling by Constraint Satisfaction, p. 22.
93/40	P.D.V. van der Stok M.M.M.P.J. Claessen D. Alstein	A Hierarchical Membership Protocol for Synchronous Distributed Systems, p. 43.
93/41	A. Bijlsma	Temporal operators viewed as predicate transformers, p. 11.
93/42	P.M.P. Rambags	Automatic Verification of Regular Protocols in P/T Nets, p. 23.
93/43	B.W. Watson	A taxomomy of finite automata construction algorithms, p. 87.
93/44	B.W. Watson	A taxonomy of finite automata minimization algorithms, p. 23.
93/45	E.J. Luit J.M.M. Martin	A precise clock synchronization protocol,p.
93/46	T. Kloks D. Kratsch J. Spinrad	Treewidth and Patwidth of Cocomparability graphs of Bounded Dimension, p. 14.
93/47	W. v.d. Aalst P. De Bra G.J. Houben Y. Kornatzky	Browsing Semantics in the "Tower" Model, p. 19.
93/48	R. Gerth	Verifying Sequentially Consistent Memory using Interface Refinement, p. 20.

94/01	P. America M. van der Kammen R.P. Nederpelt O.S. van Roosmalen H.C.M. de Swart	The object-oriented paradigm, p. 28.
94/02	F. Kamareddine R.P. Nederpelt	Canonical typing and Π-conversion, p. 51.
94/03	L.B. Hartman K.M. van Hee	Application of Marcov Decision Processe to Search Problems, p. 21.
94/04	J.C.M. Baeten J.A. Bergstra	Graph Isomorphism Models for Non Interleaving Process Algebra, p. 18.
94/05	P. Zhou J. Hooman	Formal Specification and Compositional Verification of an Atomic Broadcast Protocol, p. 22.
94/06	T. Basten T. Kunz J. Black M. Coffin D. Taylor	Time and the Order of Abstract Events in Distributed Computations, p. 29.
94/07	K.R. Apt R. Bol	Logic Programming and Negation: A Survey, p. 62.
94/08	O.S. van Roosmalen	A Hierarchical Diagrammatic Representation of Class Structure, p. 22.
94/09	J.C.M. Baeten J.A. Bergstra	Process Algebra with Partial Choice, p. 16.
94/10	T. verhoeff	The testing Paradigm Applied to Network Structure. p. 31.
94/11	J. Peleska C. Huizing C. Petersohn	A Comparison of Ward & Mellor's Transformation Schema with State- & Activitycharts, p. 30.
94/12	T. Kloks D. Kratsch H. Müller	Dominoes, p. 14.
94/13	R. Seljée	A New Method for Integrity Constraint checking in Deductive Databases, p. 34.
94/14	W. Peremans	Ups and Downs of Type Theory, p. 9.