Solving the Continuous Flow-Shop Scheduling Problem by Metaheuristics

Andreas Fink[*] Stefan Voß[*]

January 31, 2001

Continuous flow-shop scheduling problems circumscribe an important class of sequencing problems in the field of production planning. The problem considered here is to find a permutation of jobs to be processed sequentially on a number of machines under the restriction that the processing of each job has to be continuous with respect to the objective of minimizing the total processing time (flow-time). This problem is $\mathcal{NP}$-hard. We consider the application of different kinds of metaheuristics from a practical point of view, examining the trade-off between running time and solution quality as well as the knowledge and efforts needed to implement and calibrate the algorithms. Computational results show that high-quality results can be obtained in an efficient way by applying metaheuristics software components with neither the need to understand their inner working nor the necessity to manually tune parameters.

Keywords: Metaheuristics, Heuristics, Tabu Search, Simulated Annealing, Continuous Flow-Shop Scheduling

1 Introduction

Flow-shop scheduling problems focus on processing a given set of jobs, where all jobs have to be processed in an identical order on a given number of machines. The continuous flow-shop scheduling problem has the additional restriction that the

[*]Technical University of Braunschweig; Department of Business Administration, Information Systems and Information Management; Abt-Jerusalem-Str. 7, 38106 Braunschweig, Germany; email: {a.fink, stefan.voss}@tu-bs.de; Tel. +49 531 391 3213, Fax +49 531 391 8144. Corresponding author: Andreas Fink.

accepted for publication in: European Journal of Operational Research, to appear

processing of each job has to be continuous, i.e., once the processing of a job begins, there must not be any waiting times between the processing of any consecutive tasks of this job. Such a no-wait constraint is usually due to technological restrictions of the production process.[1] Dudek et al. (1992) emphasize the practical importance of this problem type (e.g., in chemical or steel production processes).

In a statistical review of flow-shop scheduling research, Reisman et al. (1997) conclude that there is a lack of relevance to practice for the overall majority of research in this field. They emphasize "that flow-shop scheduling research is in dire need of a paradigm shift to enhance its probability of ever becoming a tool for the practice of OR/MS. [...] The literature should also seek out and recognize work that is satisfied with less elegant but implementable, better yet implemented, solutions." As Nievergelt (1994) has stated, "No systems, no impact!" That is, in practice we need easy-to-use application systems that incorporate the results of basic research in the corresponding fields. Therefore, we also have to deal with the issue of efficiently building and using such systems to bridge the gap between research and practice.

Most scheduling problems, like the continuous flow-shop scheduling problem considered in this paper, are $\mathcal{NP}$-hard, so heuristics are the primary way to tackle these problems. Simple heuristic strategies may be based on applying priority based dispatching rules. More effective heuristics represent specific algorithms which are developed for some special type of problem. Since problems from practice mostly embrace distinctive characteristics, applying heuristics may imply a costly development of a specialized algorithm which hinders the application of such methods in the real world. This major problem might be partly solved by applying metaheuristics, which are widely generic with respect to the type of problem. That is, metaheuristics are general schemes which must be completed by problem-specific aspects; see Osman and Kelly (1996), Reeves (1993) and Voß et al. (1999). So in practice one would like to apply metaheuristics by using metaheuristics software components which have to be adapted to the specific problem at hand in some well-defined manner. We have developed such reusable metaheuristics software components which aid in applying metaheuristics in practice according to the goal discussed in the preceding paragraph; cf. Fink (2000), Fink and Voß (1999a, 2002) and Fink et al. (1999, 2000).

The adaptation of metaheuristics to some type of problem may concern both the static definition of problem-specific concepts such as solution space or neighborhood structure and the tuning of run-time parameters (calibration). With respect

[1]Lack of any intermediate storage capacity between consecutive machines leads to a similar but less constrained problem type, since jobs can wait on machines (while blocking these machines) until the next machine becomes available; cf. Hall and Sriskandarajah (1996), who survey machine scheduling problems with blocking and no-wait characteristics.

to the latter aspect, we need robust problem solvers that are applicable to a wide range of problems without the need for careful and time consuming calibration. Ideally, a metaheuristic is auto-adaptive, i.e., one is not forced to manually adjust some sensitive parameter setting. Auto-adaptivity means an automatic calibration of parameters by some sort of intelligence.

In this paper, we discuss, from a practical point of view, the effectiveness of applying reusable metaheuristics software components to the continuous flow-shop scheduling problem. This includes analyzing the knowledge and efforts needed to adapt the metaheuristics and analyzing by experiments the trade-off between running time and solution quality. Our goal is to gain general insights in the effectiveness of applying different types of metaheuristics with respect to different demands for solution quality and different amounts of available resources such as knowledge about algorithms, implementation efforts and running time. In Section 2, we first describe the continuous flow-shop scheduling problem. Then, in Sections 3 and 4, we review different kinds of construction methods and metaheuristics. The implementation is briefly discussed in Section 5. In Section 6, we provide and discuss extensive computational results. Finally, we draw some conclusions and give directions for future research.

2 Problem review

Flow-shop scheduling problems are defined by a set of n jobs, where each job has to be processed in an identical order on a given number of m machines. Each machine can process only one job at a time. The parameters t_{ij}, $1 \leq i \leq n$, $1 \leq j \leq m$, denote the processing time of job i on machine j. For continuous flow-shop scheduling problems the processing of each job has to be continuous, which means that there must not be any waiting times between the processing of any consecutive tasks of this job. To allow processing of a job without interruption on all machines, the order in which the jobs are processed on a machine is the same for all machines (assuming non-zero processing times). If a job does not have to be processed on some machine (zero processing time on this machine), passing could occur without violating continuous processing. However, the continuous flow-shop problem is generally understood as a permutation flow-shop problem with the characteristic that the machine order is the same for all jobs.

Continuous processing of a job generally determines an inevitable delay d_{ik}, $1 \leq i \leq n$, $1 \leq k \leq n$, $i \neq k$, on the first machine between the start of job i and the start of job k when job k is processed directly after job i. The delay may be computed as

$$d_{ik} = \max_{1 \le j \le m} \left\{ \sum_{h=1}^{j} t_{ih} - \sum_{h=2}^{j} t_{k,h-1} \right\} .$$

Consider the following example with three jobs and three machines as illustration. The processing times t_{ij} are given as matrix T, which results in a corresponding asymmetric delay matrix D:

$$T = \begin{pmatrix} 1 & 3 & 3 \\ 1 & 2 & 2 \\ 4 & 1 & 4 \end{pmatrix} \qquad D = \begin{pmatrix} - & 4 & 2 \\ 2 & - & 1 \\ 5 & 6 & - \end{pmatrix}$$

For example, for the job sequence $< 1, 3, 2 >$ this leads to the schedule sketched in Figure 1.

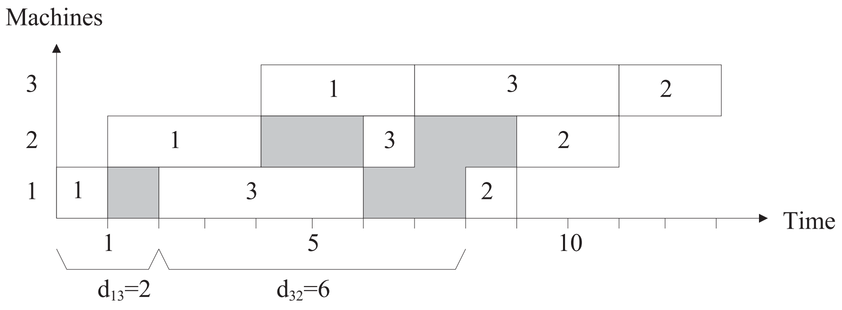

Figure 1: Job/machine schedule with continuous flow characteristic.

The objective is to construct a permutation $\Pi =< \pi_1, \ldots, \pi_n >$ of the jobs (π_i denotes the job that is positioned at the i-th position of a schedule) that minimizes some given objective function. The objective considered in this paper is to minimize the total processing time (flow-time)[2]

$$F(\Pi) = \sum_{i=2}^{n} (n + 1 - i) d_{\pi(i-1),\pi(i)} + \sum_{i=1}^{n} \sum_{j=1}^{m} t_{ij} .$$

[2]Alternatively, one may strive to minimize the makespan of the schedule, i.e., the duration from the start of the first job on the first machine until the end of the last job on the last machine. This goal leads to a problem that can be formulated as an asymmetrical traveling salesman problem (cf. Reddi and Ramamoorthy (1972) and Wismer (1972)) and, therefore, may be tackled by corresponding methods from the stock of algorithms for traveling salesman problems. Contrary to flow-time oriented problems, there has been a substantial body of research for continuous flow-shop scheduling problems with makespan objective; see, e.g., Bonney and Gundry (1976), King and Spachis (1980), Gangadharan and Rajendran (1993) and Rajendran (1994).

In this formula, besides the total sum of processing times, the delays incurred on the first machine are added depending on the number of jobs that follow. For example, the delay between the start of the job positioned first and the succeeding job affects all but the first job and accordingly must be included in the sum with a multiplier of $n-1$. Alternatively, one could compute the total processing time as the sum over all completion times of the jobs. For the example given above the total processing time is computed as $F(<1,3,2>) = 2 \cdot 2 + 1 \cdot 6 + 21 = 7 + 13 + 11 = 31$.

The continuous flow-shop scheduling problem with the objective to minimize total flow-time was posed by van Deman and Baker (1974). Rajendran and Chaudhuri (1990) study the continuous flow-shop scheduling problem with flow-time objective and present a simple construction heuristic with two different priority rules and respective computational results. Chen et al. (1996) present a genetic algorithm for these problems including computational experience. Theoretical aspects of continuous flow-shop scheduling problems are investigated by Gupta (1976), Papadimitriou and Kanellakis (1980), Szwarc (1981), Adiri and Pohoryles (1982) and van der Veen and van Dal (1991).

The continuous flow-shop scheduling problem with flow-time objective is equivalent to the one-machine scheduling problem with sequence-dependent setup times while minimizing the sum of completion times. Furthermore, in the same spirit as the problem of minimizing the makespan leads to an asymmetrical traveling salesman problem, the flow-time oriented problem corresponds to a variation of the traveling salesman problem with cumulative costs, which is also called delivery man problem; cf. Bianco et al. (1993), Fischetti et al. (1993) and Lucena (1990). There the objective is to find a Hamiltonian tour which minimizes the average arrival time at the vertices. This problem, which was proven to be $\mathcal{NP}$-hard by Sahni and Gonzales (1976), is a special case of the time-dependent traveling salesman problem, where the transition costs between vertices generally depend on the position in the sequence. Gouveia and Voß (1995) classify different mathematical formulations for the time-dependent traveling salesman problem and show that the LP relaxation of the 3-index formulation of Picard and Queyranne (1978) provides, in comparison with other formulations from the literature, a relatively strong lower bound for the optimal objective function value. In Section 6, we assess our heuristic results in comparison with corresponding lower bounds.

3 Construction methods

Construction methods build a feasible solution, i.e. a sequence of n jobs, by successively completing a (partial) solution according to some rule. When the solution quality obtained is satisfactory, such methods may be attractive from a practical point of view, since most basic construction methods, first, are rather easy to understand and allow a straightforward implementation, second, usually do not need much computation time, and, third, do not require calibration.

Priority rules are the most simple construction methods which successively build a complete sequence by adding jobs according to some preference relation. In particular, the nearest neighbor heuristic (NN) appends at each step a not yet included job with a minimal inevitable delay to the last job of the yet incomplete sequence. This strategy seems reasonable for the problem at hand, since the jobs positioned first have a greater impact on the objective function.

The cheapest insertion heuristic (Chins) considers all possible insertions of all not yet included jobs while successively building a complete sequence. That is, starting with some inital job, at each step $k, k = 2, \dots, n$, a best combination (with respect to the flow-time objective function under consideration) of the remaining $n - k + 1$ jobs and all k insertion positions is selected. This leads to a time complexity of $O(n^3)$. To illustrate the cheapest insertion heuristic we use the example problem instance from Section 2. Starting with the initial sequence $< 1 >$, in step $k = 2$ we have four alternatives to select from: $< 2, 1 >_{F=14}$, $< 1, 2 >_{F=16}$, $< 3, 1 >_{F=21}$ and $< 1, 3 >_{F=18}$. So we insert job 2 before job 1 which leads to a minimal total processing time of the resulting partial sequence. In step $k = 3$, we have three alternatives: $< 3, 2, 1 >_{F=35}$, $< 2, 3, 1 >_{F=28}$ and $< 2, 1, 3 >_{F=27}$. Accordingly, Chins provides as final solution the sequence $< 2, 1, 3 >$ with a total processing time of 27.

In general, the effectiveness of construction heuristics such as NN and Chins depends on the initial job. So we consider repeating the respective construction heuristic for all possible jobs used as initial job and eventually selecting the best final sequence. For example, we may repeat Chins for all jobs used as inital partial sequence.

Building on these ideas, we also follow the approach of the pilot method; cf. Duin and Voß (1999). The pilot method builds on the idea to look ahead for each possible local choice (by computing a so-called "pilot"), memorizing the best result, and performing the according move. That is, the idea is to evaluate the advantageousness of a local move by overcoming the usual myopic (shortsighted) move selection rule. Here we apply this strategy by performing a cheapest insertion heuristic for all possible local steps.[3] That is, in each iteration we perform the cheapest insertion

[3]As the pilot method is generic with respect to the type of problem and defines an iterative master process that guides and modifies the operations of some subordinate heuristic, it may be regarded as a metaheuristic.

heuristic for all incomplete solutions (partial sequences) resulting from adding some not yet included job at some position to the current incomplete solution.

Using the example given above, the pilot method would proceed as follows: Starting from an empty initial solution, we have to evaluate three alternatives: adding the jobs 1, 2 or 3, respectively, to the partial (empty) sequence. Then, each of these partial solutions $< 1 >$, $< 2 >$ and $< 3 >$, respectively, is completed by the usual cheapest insertion heuristic. This ends up in three corresponding final solutions with respective total processing times. While in this example all resulting sequences are the same ($< 2, 1, 3 >_{F=27}$), in general one selects that partial solution which leads to a final solution with the best objective function value. Then, in the next iteration of the pilot process, all possibilities to insert some not yet included job at some position to this partial solution would lead to four alternatives that are evaluated by the cheapest insertion heuristic, and the resulting objective function values are again used to select one of these alternatives.

As the described pilot method leads to a time complexity of $O(n^6)$, it may be reasonable to restrict the pilot process to a given *evaluation depth*. That is, the pilot method is performed until an incomplete solution with a given number of jobs is reached (and, hopefully, the most significant construction decisions have been done); this solution is completed by continuing with a conventional (myopic) cheapest insertion heuristic. Note that an evaluation depth of 1 corresponds to repeating Chins for all possible initial jobs.

4 Metaheuristics

We focus on metaheuristics that are based on the local search paradigm. That is, given some initial feasible solution which can be constructed by the methods discussed in the previous section, solutions are successively changed by performing moves which alter solutions "locally". In Section 4.1, we describe different kinds of such move definitions which lead to corresponding neighborhoods. Moves must be evaluated by some measure to guide the search. In our case, we use the implied change of the objective function value, which provides some reasonable information about the (local) advantageousness of moves.

Following a greedy strategy, steepest descent (SD) corresponds to selecting and performing in each iteration the best move; the search stops at a local optimum solution with no better neighboring solution. As the solution quality of such local optima may be unsatisfactory, we consider the application of metaheuristics with the aim to guide the search to overcome local optimality. A simple strategy is to iterate/restart the local search process (e.g., iterated steepest descent (ISD)) after a local optimum has been obtained, which requires some perturbation scheme to generate a new initial solution (e.g., performing some random moves). In Sections 4.2 and 4.3, we briefly discuss simulated annealing and various tabu search approaches, which are based on a more structured way to overcome local optimality.

4.1 Neighborhoods

We consider two alternative neighborhoods of size $O(n^2)$, which both lead to a connected search space. In each case, we use one of these neighborhoods exclusively. The neighborhood descriptions given below assume the permutation Π as a chain of jobs, each connected by an edge, with two dummy jobs that represent a virtual fixed first (0) and last (n+1) job. With this, moves are composed of attributes defined by the edges deleted and inserted. This information is exploited by the tabu search methods described below.

The idea for a swap (interchange) move is to exchange a pair of jobs π_{p_1} and π_{p_2}, $1 \leq p_1 < p_2 \leq n$, while the remaining schedule remains identical; cf. Figure 2. Thus, a swap move corresponds to the deletion of the edges (π_{p_1-1}, π_{p_1}), (π_{p_1}, π_{p_1+1}), (π_{p_2-1}, π_{p_2}) and (π_{p_2}, π_{p_2+1}) and the inclusion of the edges (π_{p_1-1}, π_{p_2}), (π_{p_2}, π_{p_1+1}), (π_{p_2-1}, π_{p_1}) and (π_{p_1}, π_{p_2+1}).

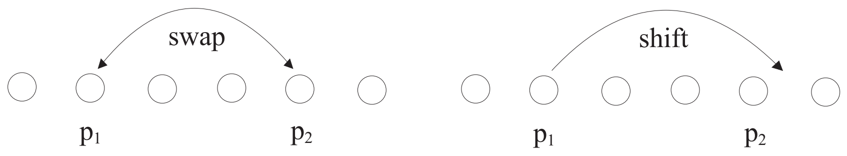

Figure 2: Swap and shift moves.

Likewise, we may define a shift (insertion) move as repositioning some job behind some other job. A shift move for the job at current position p_1 behind the job currently at position p_2 with $1 \leq p_1 \leq n$, $0 \leq p_2 \leq n$, $p_1 \neq p_2$, $p_1 - 1 \neq p_2$, may thus be defined as the deletion of the edges (π_{p_1-1}, π_{p_1}), (π_{p_1}, π_{p_1+1}), and (π_{p_2}, π_{p_2+1}) and the inclusion of the edges $(\pi_{p_1-1}, \pi_{p_1+1})$, (π_{p_2}, π_{p_1}) and (π_{p_1}, π_{p_2+1}). A restricted form of shift moves is defined by using a parameter that restricts the maximum shift distance of a job in the sequence. Setting this parameter to the value 1 leads to a neighborhood of linear size, where only swaps of jobs next to each other are considered.

A 2-exchange move, which is defined for a pair (p_1, p_2) with $0 \leq p_1$ and $p_1 + 2 \leq p_2 \leq n$ as the deletion of the edges (π_{p_1}, π_{p_1+1}) and (π_{p_2}, π_{p_2+1}) and the inclusion of the edges (π_{p_1}, π_{p_2}) and $(\pi_{p_1+1}, \pi_{p_2+1})$, corresponds to the inversion of the partial sequence between position $p_1 + 1$ and position p_2. This neighborhood is not meaningful for the continuous flow-shop scheduling problem which has an asymmetric distance function. This has been confirmed by computational experiments. Furthermore, one may use a more general 3-exchange neighborhood, which corresponds to the shift of a partial sequence to another position. However, this leads to a neighborhood size of $O(n^3)$.

4.2 Simulated annealing

Simulated annealing extends basic local search by allowing moves to inferior solutions; cf. Kirkpatrick et al. (1983) and Cerny (1985). The basic algorithm of simulated annealing may be described as follows: Successively, a candidate move is randomly selected; this move is accepted if it leads to a solution with a better objective function value than the current solution. Otherwise the move is accepted with a probability that depends on the deterioration δ of the objective function value. The probability of acceptance is usually computed as $e^{-\delta/T}$, using a temperature T as control parameter. This temperature T is gradually reduced according to some cooling schedule, so that the probability of accepting deteriorating moves decreases in the course of the annealing process.

From a theoretical point of view, a simulated annealing process may provide convergence to an optimal solution if some conditions are met (e.g., with respect to an approriate cooling schedule and a neighborhood which leads to a connected solution space); cf. van Laarhoven and Aarts (1987) and Aarts et al. (1997) which give surveys on simulated annealing with theoretical results as one main topic. As convergence rates are usually too slow, in practice one typically applies some faster cooling schedule (giving up the theoretical convergence property).

We follow the robust parameterization of the general simulated annealing procedure as described by Johnson et al. (1989). T is initially high, which allows many inferior moves to be accepted, and is gradually reduced through multiplication by a parameter $\alpha < 1$ according to a geometric cooling schedule.[4] At each temperature *sizeFactor* $\times |N|$ move candidates are tested ($|N|$ denotes the current neighborhood size), before T is reduced to $\alpha \times T$. The starting temperature is determined as follows: Given a parameter *initialAcceptanceFraction* and based on an abbreviated trial run, the starting temperature is set so that the fraction of accepted moves is approximately *initialAcceptanceFraction*. A further parameter, *frozenAcceptanceFraction* is used to decide whether the annealing process is *frozen* and should be terminated. Every time a temperature is completed with less than *frozenAcceptanceFraction* of

[4]For some other cooling schedules see, e.g., Hajek (1988), Huang et al. (1986) and Lundy and Mees (1986). Connolly (1992) and Osman (1993) propose non-monotonous temperature variations.

the candidate moves accepted, a counter is increased by one. This counter is reset every time a new best solution is found. The procedure is terminated when the counter reaches 5. Then, it is possible to reheat the temperature to continue the search by performing another annealing process. We set the parameters to the values recommended by Johnson et al. (1989): $\alpha = 0.95$, $\mathit{initialAcceptanceFraction} = 0.4$, $\mathit{frozenAcceptanceFraction} = 0.02$, $\mathit{sizeFactor} = 16$.

4.3 Tabu search

The basic paradigm of tabu search (cf. Glover and Laguna (1997)) is to use information about the search history to guide local search approaches to overcome local optimality. In its basic form, this is done by dynamically prohibiting certain moves during the neighbor selection to diversify the search. The various tabu search strategies differ especially in the way in which the tabu criteria are defined, taking into consideration the information about the search history (performed moves, traversed solutions).

In general, tabu search proceeds by selecting at each iteration the best admissible move. A neighbor, respectively a corresponding move, is called *admissible*, if it is not tabu or if an aspiration criterion is fulfilled. The aspiration criterion may override a possibly inappropriate tabu status. The aspiration criterion used here was to allow all moves that lead to a neighbor with a better objective function value than encountered so far.

Strict tabu search embodies the idea to prevent cycling to formerly traversed solutions. That is, the basic goal of strict tabu search is to provide necessity and sufficiency with respect to the idea of not revisiting any previously visited solution. Accordingly, a move is classified as tabu if and only if it leads to a neighbor that has already been visited during the previous part of the search. To accomplish this criterion, we store (approximate) information about all solutions visited so far by using a hash function which defines a non-injective transformation from the set of solutions to integer numbers; cf. Woodruff and Zemel (1993). That is, we check for every neighbor whether the respective hash code is included in the trajectory data. Given a vector $(r_1, \ldots, r_n)$ of pseudo-random integers, we use hash codes $\sum_{i=1}^{n} r_i \pi_i$. As the hash code of two different solutions may be the same whenever a so-called collision occurs, moves might be unnecessarily set tabu in some cases. However, as our own experiments have shown, this random effect usually does not affect the search negatively. While strict tabu search is easy to apply since it does not need any calibration of parameters, its tabu criterion is often too weak to provide a sufficient diversification of the search process.

The most commonly used tabu search method is to apply a *recency-based* memory that stores moves, more exactly move attributes, of the recent past. The basic idea of such static tabu search approaches is to prohibit an appropriately defined inversion of performed moves for a given period. Considering a performed move (p_1, p_2), we store, depending on the neighborhood used, move attributes that represent the inserted edges in a static tabu list of fixed length l. To obtain the current tabu status of a move, we must check whether the edges to be deleted are contained in the tabu list. As we apply multi-attribute moves, there are different ways to define the tabu criterion: a move may be classified as tabu when at least one, two, or, depending on the neighborhood used, three or four of the attributes of this move are contained in the tabu list. For our purposes, all these criteria are enabled by using a parameter *tabu threshold* (tt). It defines the number of attributes of a move that have to be contained in the tabu list in order to regard the move as tabu.

The application of static tabu search is complicated by the need to set the tabu list length to a value that is appropriate for the actual problem to be solved. Reactive tabu search aims at an automatic adaptation of this parameter during the search process; cf. Battiti (1996). The basic idea is to increase the tabu list length when the tabu memory indicates that the search is revisiting formerly traversed solutions. The concrete algorithm applied here may be described as follows: We start with a tabu list length l of 1 and increase it to $\min\{\max\{l + 2, l \times 1.2\}, u\}$ every time a solution has been repeated, taking into account an appropriate upper bound u. If there has been no repetition for some iterations, we decrease it appropriately to $\max\{\min\{l - 2, l/1.2\}, 1\}$. To accomplish the detection of a repetition of a solution, we apply a trajectory based memory using hash codes as described before. As noticed by Battiti (1996), it may be appropriate to explicitly diversify the search into new regions of the search space whenever the tabu memory indicates that we may be trapped in a certain region of the search space. Trajectory information provides means to detect such situations. As a corresponding trigger mechanism, we use the combination of at least *three solutions each having been traversed three times.* The simple diversification strategy used here is to perform randomly some moves of the corresponding neighborhood.

5 Implementation

Our application of the described methods for the continuous flow-shop scheduling problem is based on HotFrame, a Heuristic OpTimization FRAMEwork implemented in C++, which provides both adaptable components that incorporate different metaheuristics and an architectural description of the collaboration among these components and problem-specific complements; cf. Fink and Voß (1999b, 2002) and Fink (2000). All typical application-specific concepts are treated as objects or classes: problems, solutions, neighbors, solution and move attributes. On the other side, metaheuristic concepts such as different methods and their building-blocks such as tabu criteria and diversification strategies are also treated as objects. HotFrame uses genericity as the primary mechanism to make these objects adaptable. That is, common behavior of metaheuristics is factored out and grouped in generic (template) classes, applying static type variation. Metaheuristics template classes are parameterized by aspects such as solution spaces and neighborhood structures.

HotFrame provides a local search frame which can be specialized by selecting from, e.g., different neighbor selection rules resulting in methods such as steepest descent or iterated local search. In the same manner, customized simulated annealing and tabu search methods can be generated by exploiting respective components. On the other side, HotFrame provides components for often needed solution spaces such as permutations which includes corresponding neighborhood structures and attribute definitions. This simplifies applying metaheuristics, since the user, essentially, only needs to implement the objective function. Thus, in our case where one of the predefined solution spaces fits the problem at hand, the implementation efforts for applying simulated annealing or tabu search are actually less than for a simple priority rule method.[5]

The efficiency of local search methods often crucially depends on an adaptive computation of move evaluations. That is, the standard form of evaluating moves by actually performing the move to a new solution and computing the respective objective function value from scratch leaves open a great opportunity to speed up the computation. So we compute evaluations for swap and shift moves adaptively by calculating only the actual change of the objective function value. By storing for the current solution a vector with the start times of all jobs, respective computations can be performed with constant costs. The effect of such an enhancement can be illustrated by the following example: On a Pentium II with 266 MHz, for a problem with $n = 200$ jobs, the move selection, which includes the evaluation of $O(n^2)$ moves, takes about 0.9 seconds when done in the straightforward way, while the computation times could be reduced to about 0.05 seconds by an adaptive computation.

[5]Of course, in the general case, where the user may need to implement some special solution space or neighborhood structure, the application of local search methods is still not without efforts. This is in conformance with the "No free lunch theorem" which says that there is no general problem solver that is the most effective method for all types of problems; cf. Wolpert and Macready (1997) and Culberson (1998). That is, one might need to do problem-specific adaptations, which is enabled by HotFrame, to obtain high quality results.

6 Computational results

According to our general point of view discussed in Section 1, we applied the methods described in Sections 3 and 4 in a straightforward way without any manual tuning of parameters. This provides a way for a fair comparison of different heuristics by controlled and unbiased experiments, which conforms to some of the criticism discussed by Hooker (1994, 1995) and Barr et al. (1995). For example, we simply switch the neighborhood or the tabu criterion to be used without any other change.

We apply the heuristics for benchmark data sets of Taillard (1993) which have been generated for unrestricted flow-shop scheduling.[6] The data sets are available from the OR library.[7] We use problem instances with 20 (ta001–ta030), 50 (ta031–ta060), 100 (ta061–ta090), and 200 (ta091–ta110) jobs. These instances partly differ in the number of machines, which is mostly irrelevant for our purposes. So we generally present average results for problem instances of the same number of jobs. Computation times are generally given as average CPU-time in seconds. All computations were performed using a Pentium II with 266 MHz.

To assess our heuristic results we used the 3-index formulation of Picard and Queyranne (1978) to compute optimal results for the problem instances with 20 jobs and to compute lower bounds provided by the linear programming (LP) relaxation for problem instances with 50 and 100 jobs. This formulation includes n^3 binary variables and n^2 constraints. Using Cplex 6.6 on a standard PC, we were not able to solve the LP relaxation of the problem instances with 200 jobs (with about 8 million variables). So, for $n = 20$ we compare to the optimal results, for $n = 50$ and $n = 100$ to lower bounds, and for $n = 200$ to the best results obtained during all experiments. The best objective function value obtained for each problem instance is given in Table 6 in the Appendix.

Table 1 shows the results of the application of construction methods. The deviations for the identity permutations (in accordance with the numbering of the jobs), given as reference, show that there is a large potential for optimization. The results of the nearest neighbor heuristic (NN), even when iteratively performed for all jobs used as initial job (Pilot-1-NN), are much worse than those of the cheapest insertion heuristic (Chins). Repeating Chins for all jobs treated as initial job (Pilot-1-Chins) leads to clearly better results at the expense of an increased running time, which is especially relevant for the larger problem instances. The last three rows show the results of the pilot method with an evaluation depth of 10, 20, or an unbounded eval-

[6]Rajendran and Chaudhuri (1990) and Chen et al. (1996) randomly generate problem instances with a maximum job number of only 25.

[7]`http://mscmga.ms.ic.ac.uk/info.html`

uation depth, respectively. While these results are of high quality the self imposed running time limit of 10,000 seconds hindered the completion of the unbounded pilot method for the problem instances with 200 jobs. It should be noted that the majority of the small problem instances with 20 jobs are not solved to optimality by the considered construction methods (e.g., the unbounded pilot method solved 10 out of 30 instances to optimality)

	$n = 20$		$n = 50$		$n = 100$		$n = 200$	
	dev.	t	dev.	t	dev.	t	dev.	t
Identity	43.98%	-	59.61%	-	69.26%	-	74.15%	-
NN	25.81%	0.0	29.88%	0.0	30.21%	0.1	21.13%	0.3
Pilot-1-NN	19.15%	0.0	26.49%	0.2	27.90%	1.3	19.62%	11.2
Chins	3.21%	0.0	4.52%	0.1	6.18%	0.2	2.79%	1.6
Pilot-1-Chins	0.93%	0.0	2.99%	1.3	4.32%	20.1	1.62%	315.0
Pilot-10-Chins	0.27%	0.8	1.59%	37.5	3.18%	879.0	0.65%	7612.4
Pilot-20-Chins	0.25%	1.2	1.26%	107.6	2.81%	3068.4	-	-
Pilot-Chins	0.25%	1.2	1.23%	189.4	2.26%	8217.2	-	-

Table 1: Construction methods.

Figures 3 and 4 show the effect of the evaluation depth on the effectiveness of the pilot method for ten problem instances each with $n = 50$ and $n = 100$, respectively. These results confirm the expectation that the most important decisions of the construction process are made at the first steps. So it seems reasonable to use a rather small evaluation depth when running time is important.

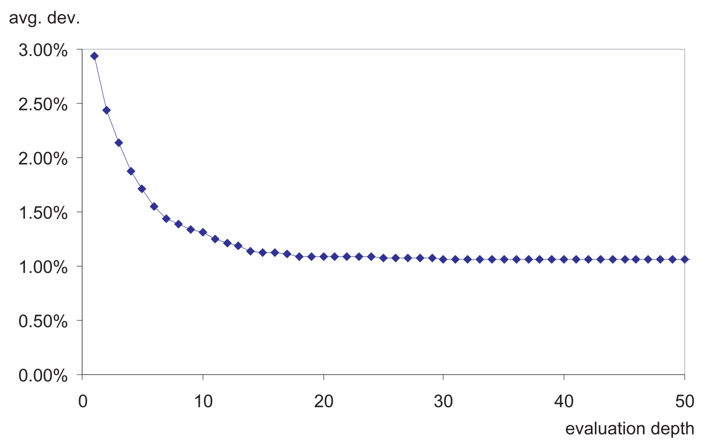

Figure 3: Average results of the pilot method with an increasing evaluation depth for ta041-050 (n=50).

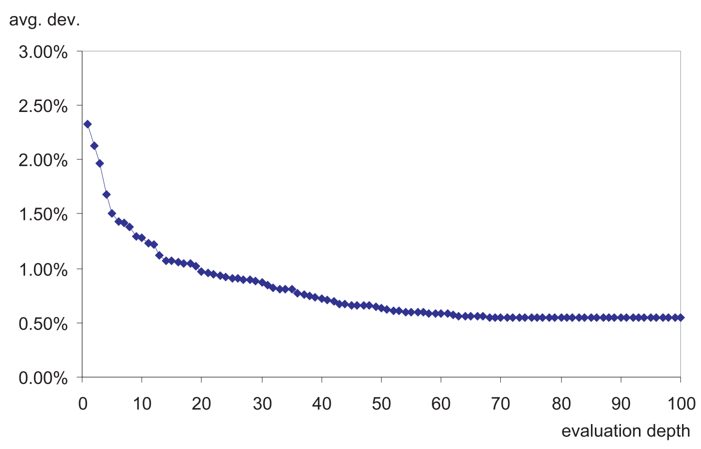

Figure 4: Average results of the pilot method with an increasing evaluation depth for ta071-080 (n=100).

With respect to the effectiveness of the different neighborhood structures, Table 2 compares the results obtained in connection with the shift neighborhood with the results obtained in connection with the swap neighborhood. We applied steepest descent (initial solutions: identity permutation vs. Chins) and iterated steepest descent (initial solution provided by Chins) for 100 seconds. The shift neighborhood performed significantly better than the swap neighborhood. This finding has been confirmed by experiments for simulated annealing and tabu search, so we restrict in the following to presenting results for the shift neighborhood. Steepest descent profits from starting from a good initial solution (both with respect to solution quality and running time).[8] Experiments with implying a maximum shift distance (e.g., 10) for the shift neighborhood led to significantly worse results (e.g., for iterated steepest descent a deviation of 3.60% in comparison to 1.29% for $n = 50$). With respect to the diversification performed after each local optimum, a lower perturbation extent ($n/5$ random moves instead of $n/2$ random moves), which keeps some structure of the local optimum, led to the better results. It is notable, that iterated steepest descent, even when restricted to a running time of 10 seconds, solves all problem instances with 20 jobs to optimality.

Table 3 shows the results of applying simulated annealing, in connection with the

[8]The running times for steepest descent, which are not shown in the table, are below one second for all but the largest problem instances. The running times for the instances with 200 jobs are about 10 seconds when starting from the identity permutation vs. 2–3 seconds when starting from the initial solution provided by Chins.

	$n = 20$		$n = 50$		$n = 100$		$n = 200$	
	shift	swap	shift	swap	shift	swap	shift	swap
SD (Identity)	2.63%	5.19%	5.22%	9.50%	7.01%	11.81%	4.83%	9.62%
SD (Chins)	2.06%	2.72%	3.34%	3.94%	5.05%	5.63%	1.82%	2.53%
ISD (div.: n/2 r. moves)	0.00%	0.06%	1.69%	3.32%	4.35%	5.50%	1.82%	2.53%
ISD (div.: n/5 r. moves)	0.00%	0.06%	1.29%	2.77%	3.84%	5.11%	1.71%	2.53%

Table 2: Steepest descent and iterated steepest descent applied for 100 seconds with different neighborhoods and different diversification extent.

shift neighborhood, in the scheme described in Section 4.2. We have performed for each problem instance and each algorithm configuration five runs with different seed values for the pseudo-random number generator; the results are averaged. When not marked otherwise, we used the parameter setting recommended by Johnson et al. (1989). The first column marks the initial solution used. As is well-known, simulated annealing usually does not profit from good initial solutions as it perturbs the initial solution intensively. This has been confirmed by our experiments, using the default parameter setting of *initialAcceptanceFraction* (i.a.f.) to 0.4. We also examined lowering this parameter to 0.1, which reduces the running time by almost 50% (due to the lower initial temperature induced). Trying to slow down the cooling process by setting the cooling schedule parameter α to 0.98 (instead of 0.95) did not lead to better results. On the other hand, performing a reheating twice, i.e., continuing the search process from a higher temperature when the search process is considered as frozen, led to a similar increase in running time while the solution quality became significantly better. We followed two different strategies: In the first case, reheating sets the temperature to half of the value of the initial temperature of the actual annealing process. In the second case, we remember the temperature when the best solution was found and reheat to the average of this value and the initial temperature of the actual annealing process. The latter strategy, which performed slightly better than the former one, was also tested for a fourfold reheating.

	$n = 20$		$n = 50$		$n = 100$		$n = 200$	
	dev.	t	dev.	t	dev.	t	dev.	t
Identity (Id.)	0.05%	4.5	1.37%	28.8	3.07%	124.4	1.26%	582.0
Chins	0.04%	4.5	1.36%	30.0	3.08%	125.5	1.30%	582.2
Id., i.a.f.=0.1	0.04%	2.3	1.43%	16.3	3.05%	69.1	1.25%	314.5
Id., $\alpha = 0.98$	0.01%	11.2	1.27%	74.8	3.07%	321.0	1.66%	1450.3
Id., 2 reheatings (half)	0.01%	9.2	1.12%	61.7	2.74%	255.5	1.07%	1174.0
Id., 2 reheatings (avg.)	0.01%	5.1	1.06%	34.0	2.74%	151.4	1.05%	741.1
Id., 4 reheatings (avg.)	0.00%	7.0	0.98%	44.0	2.64%	196.4	1.00%	982.1

Table 3: Simulated annealing in connection with shift neighborhood.

Table 4 shows the results of the application of tabu search in connection with the shift neighborhood. Strict tabu search obtained unsatisfactory results. Concerning static tabu search, in accordance with our general point of view discussed in Section 1, we have not tried to find some "best" parameter setting which generally depends on problem size and running time. We have tried a few different combinations of values for the *tabu threshold* (tt) and the tabu list length (l). When comparing the best of these combinations with the results for reactive tabu search, one may conclude that the selected values for the tabu list length seem appropriate for the larger problem instances while the results for the smaller instances are unsatisfactory. On the other hand, by the application of reactive tabu search we obtained high quality results without having to think about parameter settings. Results of applying reactive tabu search ($tt = 2$) for 1000 seconds in connection with the shift neighborhood on the basis of initial solutions provided by Chins and Pilot-10 are shown in Table 5.

		$n = 20$	$n = 50$	$n = 100$	$n = 200$
strict		0.02%	1.89%	4.46%	1.56%
static	$tt = 1, l = \sqrt{n}$	1.38%	2.99%	4.76%	1.69%
	$tt = 1, l = 2\sqrt{n}$	1.43%	2.90%	4.66%	1.55%
	$tt = 2, l = 2\sqrt{n}$	1.11%	2.35%	4.06%	1.25%
	$tt = 2, l = 3\sqrt{n}$	0.67%	2.00%	3.42%	1.06%
reactive	$tt = 1$	0.00%	1.20%	3.34%	1.38%
	$tt = 2$	0.00%	1.36%	3.09%	1.33%

Table 4: Tabu search with different tabu criteria applied for 100 seconds in connection with shift neighborhood and initial solutions provided by Chins.

		$n = 20$	$n = 50$	$n = 100$	$n = 200$
Reactive tabu search	Chins	0.00%	0.88%	2.20%	1.19%
	Pilot-10	0.00%	0.74%	1.88%	0.08%

Table 5: Reactive tabu search applied for 1000 seconds in connection with shift neighborhood on the basis of initial solutions provided by Chins and Pilot-10.

From a practical point of view it is especially important which methods are the most effective ones for some given combination of problem size, available running time and required solution quality. While such relations may in general be highly complex, we aim for some basic understanding of these concerns. The Figures 5, 6 and 7 indicate the basic trade-off between solution quality and running time (logarithmic scale) for problems with 50, 100, and 200 jobs, respectively.[9] Based on the

[9]To enable some assessment of the iteration numbers: Methods such as iterated steepest descent

experiments described above, we selected six different metaheuristics with respective configurations:

- Pilot method with increasing evaluation depth
- Iterated steepest descent (initial solution constructed by Chins, shift neighborhood, diversification by $n/5$ random moves, running times: 10s, 100s, 1000s)
- Simulated annealing (identity permutation as initial solution, default parameter setting, shift neighborhood without reheating and with two and four reheatings to the average of the initial temperature and the temperature where the best solution so far has been obtained)
- Strict tabu search (initial solution constructed by Chins, shift neighborhood, running times: 10s, 100s, 1000s)
- Static tabu search (initial solution constructed by Chins, shift neighborhood, $tt = 2$, $l = 3\sqrt{n}$, running times: 10s, 100s, 1000s)
- Reactive tabu search (initial solution constructed by Chins, shift neighborhood, $tt = 2$, running times: 10s, 100s, 1000s)

For $n = 50$, several methods show the same behaviour. Simulated Annealing provides high quality results with not much running time. Strict tabu search and static tabu search perform bad. An explanation for the latter one might be the inappropriate tabu list length (remember that we did not tune this parameter). For small size problem instances a straightforward application of iterated steepest descent might be the advice for a practical application when one wants to freely determine the running time. For $n = 100$, the situation changes. Iterated steepest descent provides average results and is outperformed by, partly, static tabu search and, definitely, by reactive tabu search. This tendency towards the more "intelligent" methods becomes even clearer for $n = 200$. Simulated Annealing, in particular in connection with some reheatings steps, also provides high quality results without the need to perform a calibration, however, with restrictions with respect to freely determining running times.

or tabu search perform about 700 iterations per second for $n = 20$, 180 iterations per second for $n = 50$, 55 iterations per second for $n = 100$, or 16 iterations per second for $n = 200$.

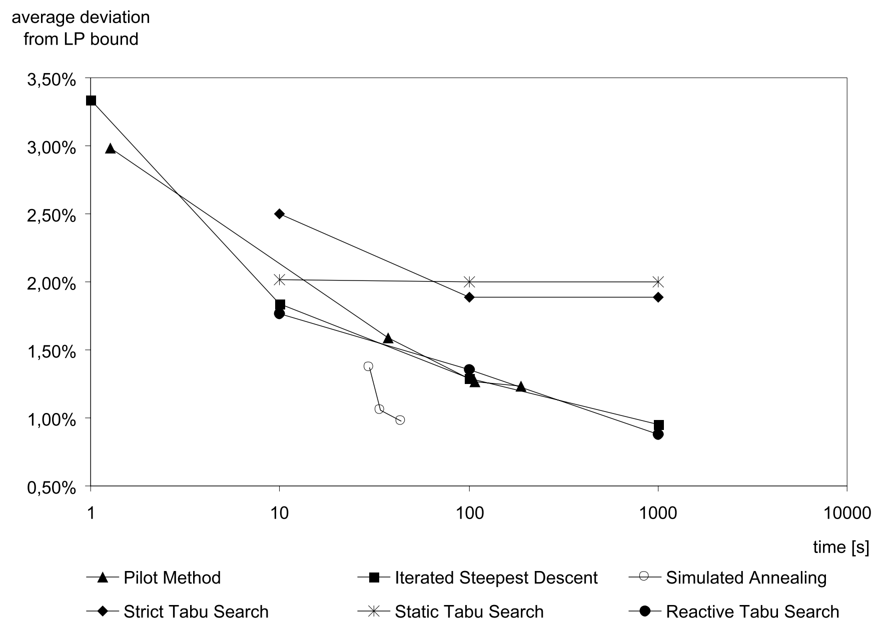

Figure 5: Solution quality vs. running time for problem instances with $n = 50$ jobs.

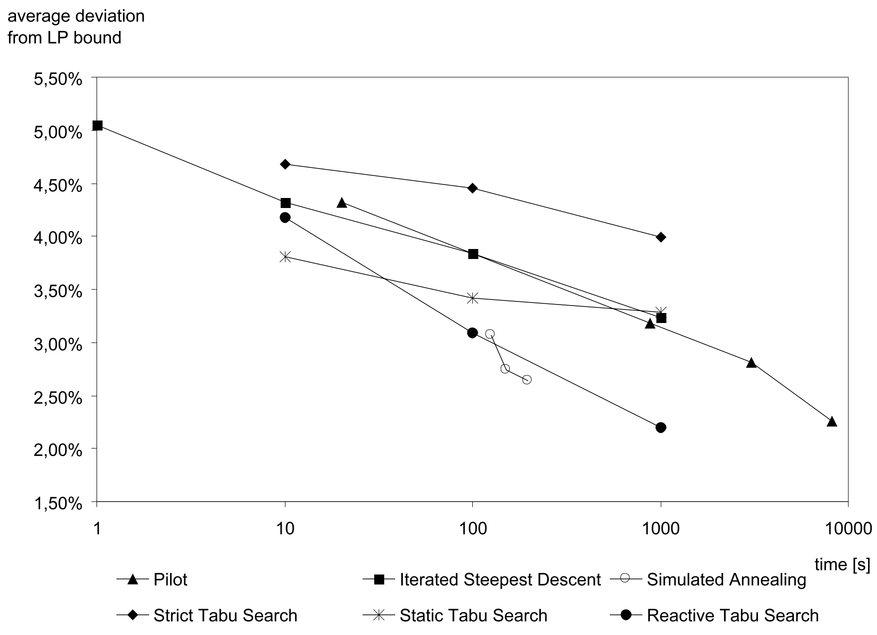

Figure 6: Solution quality vs. running time for problem instances with $n = 100$ jobs.

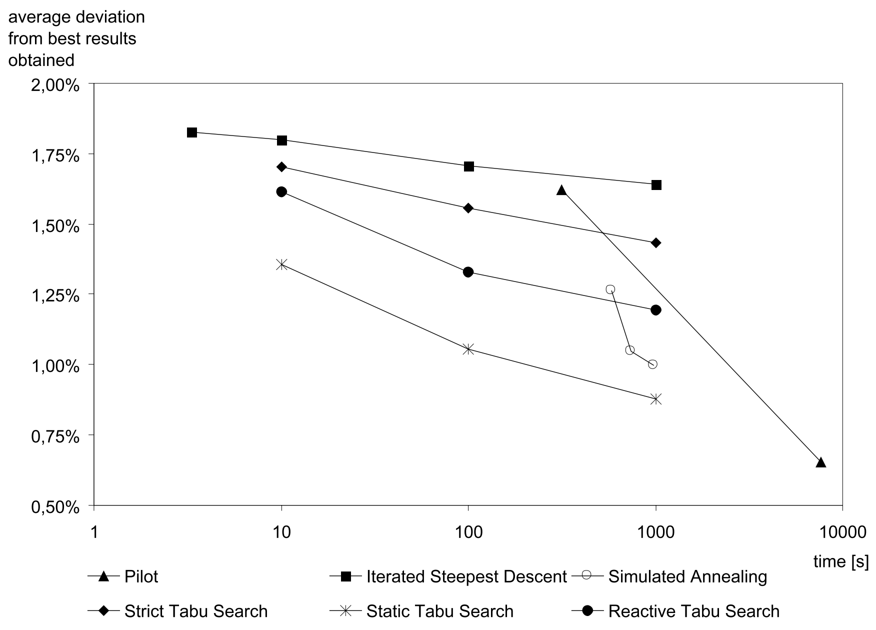

Figure 7: Solution quality vs. running time for problem instances with $n = 200$ jobs.

7 Conclusions

We examined the application of different kinds of heuristic methods to the continuous flow-shop scheduling problem. The results of computational experiments confirm that the effectiveness of (meta-)heuristics crucially depends on the constellation of problem size, wanted solution quality and available running time. That is, there is no single "best" method that dominates all other considered methods. The computational results give evidence for the superiority of the neighborhood structure determined by shift moves. For small problem instances, even simple methods such as iterated steepest descent obtained very good results with not much running time. For all but the smallest problem instances, the pilot method provided excellent results at the expense of large running times; future research should examine the application of corresponding look-ahead strategies for improvement methods. In general, reactive tabu search obtained high quality results without the need to perform some parameter tuning.

In principle, most of the considered metaheuristics such as simulated annealing and reactive tabu search can be applied without knowledge about the inner working of these algorithms: calibration is mostly not necessary and efficient implementation support is provided by HotFrame. Our implementation allows the effective treatment of considerably larger problem instances for the continuous flow-shop scheduling problem than previously reported in the literature (200 jobs vs. 25 jobs). Further work should include the treatment of other types of flow-shop problems. Furthermore, in connection with the goal set in the introduction, the next research steps should not be to improve the objective function value of the "soft" goal *minimization of total processing time* by the last percent, but to transfer the findings described in this paper to decision support systems dealing with real world problems. Respective work may confirm the generality of our research as various problem types may be subsumed under an appropriate framework.

Moreover, future research should increase the explicit treatment and examination of online problems. In practice, planning must take into account changing data (e.g., new jobs). In this respect, one often prefers relatively stable plans instead of "nervous" changes; i.e., new data should not completely invalidate the existing plan as might be the case when new schedules are computed from scratch. In this respect, local search methods naturally provide the means of adaptively changing schedules on the basis of existing plans.

References

Aarts, E.H.L., J.H.M. Korst, and P.J.M. van Laarhoven (1997). Simulated annealing. In E. Aarts and J. Lenstra (Eds.), *Local Search in Combinatorial Optimization*, pp. 91–120. Wiley, Chichester.

Adiri, I. and D. Pohoryles (1982). Flowshop/no-idle or no-wait scheduling to minimize the sum of completion times. *Naval Research Logistics Quarterly 29*, 495–504.

Barr, R.S., B.L. Golden, J.P. Kelly, M.G.C. Resende, and W.R. Stewart (1995). Designing and reporting on computational experiments with heuristic methods. *Journal of Heuristics 1*, 9–32.

Battiti, R. (1996). Reactive search: Toward self-tuning heuristics. In V. Rayward-Smith, I. Osman, C. Reeves, and G. Smith (Eds.), *Modern Heuristic Search Methods*, pp. 61–83. Wiley, Chichester.

Bianco, L., A. Mingozzi, and S. Ricciardelli (1993). The traveling salesman problem with cumulative costs. *Networks 23*, 81–91.

Bonney, M.C. and S.W. Gundry (1976). Solutions to the constrained flowshop sequencing problem. *Operations Research Quarterly 24*, 869–883.

Cerny, V. (1985). Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications 45*, 41–51.

Chen, C.-L., R.V. Neppalli, and N. Aljaber (1996). Genetic algorithms applied to the continuous flow shop problem. *Computers & Industrial Engineering 30*, 919–929.

Connolly, D. (1992). General purpose simulated annealing. *Journal of the Operational Research Society 43*, 495–505.

Culberson, J.C. (1998). On the futility of blind search: An algorithmic view of "no free lunch". *Evolutionary Computation 6*, 109–127.

Dudek, R.A., S.S. Panwalkar, and M.L. Smith (1992). The lessons of flowshop scheduling research. *Operations Research 40*, 7–13.

Duin, C.W. and S. Voß (1999). The pilot method: A strategy for heuristic repetition with application to the Steiner problem in graphs. *Networks 34*, 181–191.

Fink, A. (2000). *Software-Wiederverwendung bei der Lösung von Planungsproblemen mittels Meta-Heuristiken.* Shaker, Aachen.

Fink, A., G. Schneidereit, and S. Voß (2000). Solving general ring network design problems by meta-heuristics. In M. Laguna and J. González Velarde (Eds.), *Computing Tools for Modeling, Optimization and Simulation*, pp. 91–113. Kluwer, Boston.

Fink, A. and S. Voß (1999a). Applications of modern heuristic search methods to pattern sequencing problems. *Computers & Operations Research 26*, 17–34.

Fink, A. and S. Voß (1999b). Generic metaheuristics application to industrial engineering problems. *Computers & Industrial Engineering 37*, 281–284.

Fink, A. and S. Voß (2002). HotFrame: A heuristic optimization framework. In S. Voß and D. Woodruff (Eds.), *Optimization Software Class Libraries*, pp. 81–154. Kluwer, Boston.

Fink, A., S. Voß, and D.L. Woodruff (1999). An adoption path for intelligent heuristic search componentware. In E. Rolland and N. Umanath (Eds.), *Proceedings of the 4th INFORMS Conference on Information Systems and Technology*, pp. 153–168. INFORMS, Linthicum.

Fischetti, M., G. Laporte, and S. Martello (1993). The delivery man problem and cumulative matroids. *Operations Research 41*, 1055–1076.

Gangadharan, R. and C. Rajendran (1993). Heuristic algorithms for scheduling in the no-wait flowshop. *International Journal of Production Economics 32*, 285–290.

Glover, F. and M. Laguna (1997). *Tabu Search.* Kluwer, Dordrecht.

Gouveia, L. and S. Voß (1995). A classification of formulations for the (time-dependent) traveling salesman problem. *European Journal of Operational Research 83*, 69–82.

Gupta, J.N.D. (1976). Optimal flowshop schedules with no intermediate storage space. *Naval Research Logistics Quarterly 23*, 235–243.

Hajek, B. (1988). Cooling schedules for optimal annealing. *Mathematics of Operations Research 13*, 311–329.

Hall, N.G. and C. Sriskandarajah (1996). A survey of machine scheduling problems with blocking and no-wait in process. *Operations Research 44*, 510–525.

Hooker, J.N. (1994). Needed: An empirical science of algorithms. *Operations Research 42*, 201–212.

Hooker, J.N. (1995). Testing heuristics: We have it all wrong. *Journal of Heuristics 1*, 33–42.

Huang, M.D., F. Romeo, and A. Sangiovanni-Vincentelli (1986). An efficient general cooling schedule for simulated annealing. In *Proceedings of ICCAD-86: IEEE International Conference on Computer-Aided Design*, pp. 381–384. Santa Clara.

Johnson, D.S., C.R. Aragon, L.A. McGeoch, and C. Schevon (1989). Optimization by simulated annealing: an experimental evaluation; part 1, graph partitioning. *Operations Research 37*, 865–892.

King, J.R. and A.S. Spachis (1980). Heuristics for flow-shop scheduling. *International Journal of Production Research 18*, 343–357.

Kirkpatrick, S., C. Gelatt Jr., and M. Vecchi (1983). Optimization by simulated annealing. *Science 220*, 671–680.

Lucena, A. (1990). Time-dependent traveling salesman problem – the deliveryman case. *Networks 20*, 753–763.

Lundy, M. and A. Mees (1986). Convergence of an annealing algorithm. *Mathematical Programming 34*, 111–124.

Nievergelt, J. (1994). Complexity, algorithms, programs, systems: The shifting focus. *Journal of Symbolic Computation 17*, 297–310.

Osman, I.H. (1993). Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. In F. Glover, E. Taillard, M. Laguna, and D. de Werra (Eds.), *Tabu Search*, Annals of Operations Research 41, pp. 421–451. Baltzer, Amsterdam.

Osman, I.H. and J.P. Kelly (Eds.) (1996). *Meta-Heuristics: Theory and Applications.* Kluwer, Boston.

Papadimitriou, C.H. and P.C. Kanellakis (1980). Flowshop scheduling with limited temporary storage. *Journal of the ACM 27*, 533–549.

Picard, J.-C. and M. Queyranne (1978). The time-dependent traveling salesman problem and its application to the tardiness problem in one-machine scheduling. *Operations Research 26*, 86–110.

Rajendran, C. (1994). A no-wait flowshop scheduling heuristic to minimize makespan. *Journal of the Operational Research Society 45*, 472–478.

Rajendran, C. and D. Chaudhuri (1990). Heuristic algorithms for continuous flowshop problem. *Naval Research Logistics 37*, 695–705.

Reddi, S.S. and C.V. Ramamoorthy (1972). On the flow-shop sequencing problem with no wait in process. *Operational Research Quarterly 23*, 323–331.

Reeves, C.R. (Ed.) (1993). *Modern Heuristic Techniques for Combinatorial Problems.* Halsted, Blackwell.

Reisman, A., A. Kumar, and J. Motwani (1997). Flowshop scheduling/sequencing research: A statistical review of the literature, 1952–1994. *IEEE Transactions on Engineering Management 44*, 316–329.

Sahni, S. and T. Gonzales (1976). $\mathcal{P}$-complete approximation problems. *Journal of the Association of Computing Machinery 23*, 555–565.

Szwarc, W. (1981). A note on the flow-shop problem without interruptions in job processing. *Naval Research Logistics Quarterly 28*, 665–669.

Taillard, E. (1993). Benchmarks for basic scheduling instances. *European Journal of Operational Research 64*, 278–285.

van Deman, J.M. and K.R. Baker (1974). Minimizing mean flowtime in the flow shop with no intermediate queues. *AIIE Transactions 6*, 28–34.

van der Veen, J.A.A. and R. van Dal (1991). Solvable cases of the no-wait flow-shop scheduling problem. *Journal of the Operational Research Society 42*, 971–980.

van Laarhoven, P.J.M. and E.H.L. Aarts (1987). *Simulated Annealing: Theory and Applications.* Reidel, Dordrecht.

Voß, S., S. Martello, I.H Osman, and C. Roucairol (Eds.) (1999). *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization.* Kluwer, Boston.

Wismer, D.A. (1972). Solution of the flowshop sequencing problem with no intermediate queues. *Operations Research 20*, 695–705.

Wolpert, D.H. and W.G. Macready (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation 1*, 67–82.

Woodruff, D.L. and E. Zemel (1993). Hashing vectors for tabu search. *Annals of Operations Research 41*, 123–138.

Appendix

$n = 20$		$n = 50$		$n = 100$		$n = 200$	
ta001	15674	ta031	76016	ta061	308052	ta091	1521201
ta002	17250	ta032	83403	ta062	302386	ta092	1516009
ta003	15821	ta033	78282	ta063	295239	ta093	1515535
ta004	17970	ta034	82737	ta064	278811	ta094	1489457
ta005	15317	ta035	83901	ta065	292757	ta095	1513281
ta006	15501	ta036	80924	ta066	290819	ta096	1508331
ta007	15693	ta037	78791	ta067	300068	ta097	1541419
ta008	15955	ta038	79007	ta068	291859	ta098	1533397
ta009	16385	ta039	75842	ta069	307650	ta099	1507422
ta010	15329	ta040	83829	ta070	301942	ta100	1520800
ta011	25205	ta041	114398	ta071	412700	ta101	2012785
ta012	26342	ta042	112725	ta072	394562	ta102	2057409
ta013	22910	ta043	105433	ta073	405878	ta103	2050169
ta014	22243	ta044	113540	ta074	422301	ta104	2040946
ta015	23150	ta045	115441	ta075	400175	ta105	2027138
ta016	22011	ta046	112645	ta076	391359	ta106	2046542
ta017	21939	ta047	116560	ta077	394179	ta107	2045906
ta018	24158	ta048	115056	ta078	402025	ta108	2044218
ta019	23501	ta049	110482	ta079	416833	ta109	2037040
ta020	24597	ta050	113462	ta080	410372	ta110	2046966
ta021	38597	ta051	172845	ta081	562150		
ta022	37571	ta052	161092	ta082	563923		
ta023	38312	ta053	160213	ta083	562404		
ta024	38802	ta054	161557	ta084	562918		
ta025	39012	ta055	167640	ta085	556311		
ta026	38562	ta056	161784	ta086	562253		
ta027	39663	ta057	167233	ta087	574102		
ta028	37000	ta058	168100	ta088	578119		
ta029	39228	ta059	165292	ta089	564803		
ta030	37931	ta060	168386	ta090	572798		

Table 6: Best results obtained for data sets of Taillard when treated as continuous flow-shop problem instances with total processing time objective.