UNIVERSITY OF LEIPZIG

Institute of Production Management
and Industrial Information Management

Marschnerstr. 31, 04109 Leipzig, Germany
Phone: [49] / (0)341 / 4941-182, Fax: -125

Report No. 16

An Efficient Scheduling Algorithm Based upon Threshold Accepting

Jukka Siedentopf

⟨siedentopf@wifa.uni-leipzig.de⟩

- to appear -

March 1995

Index

Summary

A mutation-selection approach for the general job shop scheduling problem is presented. The underlying algorithm uses *threshold accepting* as an iterative search technique generating alternative operation sequences for the machines of the job shop. Search is combined with a simple heuristic algorithm transforming operation sequences into a schedule. Effectiveness of the approach is demonstrated within a comparison with some well-known heuristic and exact algorithms, which in part are clearly outperformed.

Remark: This report is a shortened and slightly changed version of report No. 4 of the Institute of Production Management and Industrial Information Management.

1 Introduction

Facing combinatorial optimization problems, approximation approaches are proposed as alternatives to somewhat more conventional approaches of mathematical programming. Some of these approaches make use of stochastic processes to prevent premature termination in local optima. A common characteristic of the proposed approaches is the principle of generating new solution candidates, so-called *configurations*, by performing (stochastic) perturbations, so-called *mutations*, of already available configurations. Another characteristic is that the selection of configurations permitted to remain in the solution process depends not only on the quality of the solution represented by the considered configuration but also on external parameters. Furthermore, the selection can be made in an either deterministic or probabilistic way.

The number of involved configurations can serve as a classification criterion for mutation-selection procedures: In *genetic algorithms*, a multitude of configurations is managed in a *population*. Availability of several different configurations is an indispensable requirement for *recombination* of new configurations from the building blocks of already available ones. Recombination is the predominating principle for generating configurations in genetic algorithms. Mutation serves for adding new building blocks by implementing small stochastic perturbations in the available building material.

In another more simply structured class of mutation-selection approaches, only one configuration undergoes a process of mutation. A decision as to selection is reduced to a decision whether a new, modified configuration will replace the old one or not. Examples for such procedures are threshold accepting, *simulated annealing* or the g*reat deluge algorithm*.

The number of involved configurations limits the number of steps in the solution space, which can be examined during one iteration of the underlying process. The processes can thus be intuitively classified as *multidirectional* and *unidirectional* processes.

Complex scheduling problems have been a main emphasis particularly of the application of genetic algorithms since about the middle of the 80ies (e.g. Forrest 1993). Despite evident similarities, unidirectional methods only occasionally have been concerned as an alternative - even systematic comparisons of the approaches are missing. The submitted contribution sketches development and test of a scheduling algorithm based upon threshold accepting. The algorithm was originally developed for the purpose of a comparison with a specific genetic algorithm and adopts several modules of that algorithm (see sections 3.2 and 3.3).

2 The Problem

The considered scheduling problem is an incarnation of the general job shop scheduling problem (e.g. French 1982):

- n jobs $\{J_1, J_2, ... , J_n\}$ are to be processed on m machines $\{M_1, M_2, ... , M_m\}$.
- Each job is to be processed exactly once on each machine; processing a job on a machine is also referred to as an *operation* or a *task*.
- Each job must run through the machines in a specific technologically conditioned order (*machine sequence*). Different machine sequences for different jobs are permitted.
- The sequences the jobs are to be processed in on distinct machines (*task* or *operation sequences*) are subject of disposal.
- The processing times of the operations are known and constant, i.e. independent of precedence relationships. Setup and transportation times are not considered, and interruption of the processing of an operation is not permitted (*non-preemptive case*).
- The mission is to determine task sequences respecting machine sequences and optimizing the value of a given objective function. In the following, the objective of minimizing the period of time for complete processing of all jobs (*makespan*) is considered.

3 The Algorithm

The presented scheduling algorithm is based on threshold accepting as presented from Dueck and Scheuer 1990 or - as a part of a multi-phase process (*mutation selection strategy with destabilization phase*) - from Ablay 1987. Threshold accepting supplies the base structure of a unidirectional iterative search process for generating and selecting configurations (section 3.1). A configuration represents a suggestion for (all) the machine sequences. The sequences are modified during the solution process (section 3.2), and for the purpose of evaluation a configuration is transformed into a schedule by means of a simple heuristic scheduling algorithm eliminating occasionally occuring inconsistencies (section 3.3).

3.1 The Iterative Search Process

The considered algorithm (**T**hreshold **A**ccepting for **M**akespan **M**inimization - **TAMM**) is presented in Fig. 1. The algorithm is initialized by a configuration (*candidate*), a start value for the parameter *threshold*, and an instruction determining the reduction of *threshold* in the lapse of time. This instruction includes the quantity of the respective reduction (*threshold_lower_step*) as well as the number of iterations after which a reduction is enforced (*change_threshold_all*). Besides, a termination criterion is defined. The criterion indicates the number of unsuccessful iterations (iterations without any increase of solution quality) leading to a termination of the process (*max_unsuccessful_trials*).

In step (1) the quality (makespan) of the configuration is determined. This is performed by timetabling the operations according to the proposed task sequences by a module called *BUILD_SCHEDULE*. During this scheduling process it might become necessary to eliminate inconsistencies within the task sequences (section 3.3).

In step (2) mutation is used to choose a new configuration (*new_candidate*) from the neighbourhood of *candidate*. For that purpose, small (´local´) changes are installed in the task sequences represented by *candidate* (section 3.2, see e.g. Aarts and Korst 1989 for a formal description of neighbourhood structures in local search). Again the quality of *new_candidate* is determined by timetabling the operations of the represented task sequences (step (3)). The difference of the qualities ΔE of both configurations is calculated (step (4)), and an iteration counter is actualized (step (5)) which is required for the reduction of the threshold in step (9).

If the quality of *new_candidate* does not exceed that one of *candidate* ($\Delta E \leq 0$, step (6); higher quality corresponds to lower makespan and vice versa) a counter *unsuccessful_trials* is updated. Otherwise, *new_configuration* is stored as the best current configuration (*temp_optimum*) and the counter *unsuccessful_trials* is reset (step (7)).

In the following iteration of TAMM n*ew_candidate* will serve as a working basis if its solution quality is better or not more than *threshold* worse than that one of *candidate* (step (8)). Hence, in the next iteration a new configuration will be generated in the neighbourhood of *new_candidate* instead of *candidate*. The *threshold* is modified in step (9) if necessary. Subsequently, the termination criterion is examined (step (10)) and, according to the result, the process either is interrupted or - beginning with the mutation in step (2) - is repeated. In the case of discontinuance, the best configuration found (*temp_optimum*) is made available to any further processing as an approximization of the aspired optimum (*solution*, step (11)).

The underlying approximization process corresponds largely to the better-known simulated annealing (see e.g. Aarts and Korst 1989). A significant difference is due to the acceptance of new configurations: In contrast to simulated annealing, threshold accepting does without *probabilities* of acceptance. Hence, the formulation of a *cooling* or *annealing schedule*, a very critical task in applying simulated annealing (cf. Aarsts and Korst 1989, pp. 57, or Dueck and Scheuer 1990, p. 162), is replaced by a (more simple) instruction of a *threshold decrease*.

3.2 Representation and Modification of Configurations

Originally, TAMM has been developed to be compared with an algorithm of Nakano and Yamada based on a ´classical´ genetic algorithm (Nakano and Yamada 1991). Nakano and Yamada code configurations as sequences of binary values (*bitstrings*) and produce new configurations applying the standard recombination operators of *mutation* and *crossover* (Holland 1975). The announced comparison should explore whether the impact of a multidirectional

search and the recombination in the genetic algorithm can compensate for the disadvantages of a binary representation. On one hand, the rejection of recombination within threshold accepting enables the introduction of a symbolic representation of configurations in TAMM. On the other hand, the modification and evaluation components in the algorithm of Nakano and Yamada are left unchanged in functionality - adapted only to the new representation.

```
THRESHOLD ACCEPTING FOR MAKESPAN MINIMIZATION
START
    initialize  candidate
                threshold
                threshold_lower_step
                change_threshold_all
                max_unsuccessful_trials
                trials ← 0
                unsuccessful_trials ← 0
                temp_optimum ← candidate
    compute makespan (candidate) within BUILD_SCHEDULE                (1)
    LOOP
        select new_candidate in the neighbourhood of candidate        (2)
        compute makespan (new_candidate) within BUILD_SCHEDULE        (3)
        ΔE ← makespan (candidate) - makespan (new_candidate)          (4)
        trials ← trials + 1                                           (5)
        IF ΔE ≤ 0 THEN                                                (6)
            unsuccessful_trials ← unsuccessful_trials + 1
        ELSE                                                          (7)
            temp_optimum ← new_candidate
            unsuccessful_trials ← 0
        IF ΔE > (-1)·threshold THEN candidate ← new_candidate         (8)
        IF trials = change_threshold_all AND threshold > 0 THEN       (9)
            trials ← 0
            threshold ← threshold - threshold_lower_step
    UNTIL unsuccessful_trials > max_unsuccessful_trials               (10)
    solution ← temp_optimum                                           (11)
END
```

Fig. 1: TAMM

In the approach of Nakano and Yamada, task sequences are represented as lists of binary precedence values. A precedence function $precedence(o_{ij}, o_{kj})$ receives value 1, if job i is processed before job k on machine j, and value 0 otherwise. The representation of a configuration comprises all precedence values among operations to be processed on the same machine. It is obvious that lists of binary precedence values does not neccessaryly represent admissible sequences. Consider, for instance, for a machine x the following precedence values:

$$precedence(o_{1x}, o_{2x}) = 1 \ \wedge \ precedence(o_{2x}, o_{3x}) = 1 \ \wedge \ precedence(o_{1x}, o_{3x}) = 0$$

The two first values denote that job 1 should be processed before job 2 and job 2 before job 3. Because of the transitivity of the precedence relationship, job 1 should have to be processed before job 3, too. This, however, contradicts the third precedence value. Thus the above-mentioned bitstring (1,1,0) does not present a permissible job permutation for machine x. For eliminating the sketched inconsistencies, Nakano and Yamada propose a *repair algorithm* called *local harmonization* which should produce a permissible job permutation for a considered machine while ´flipping´ a minimal number of bits.

Since binary coding in the approach of Nakano and Yamada serves only for the applicability of conventional crossover and mutation operators and threshold accepting works without these operators, it can dispense with a binary representation. Schedules are coded directly as task sequences, which are entered in a matrix row by row for the individual machines. In Fig. 2 an example of a configuration for the processing of 4 jobs on 4 machines is presented.

	1st position	2nd position	3rd position	4th position
M_1	J_1	J_2	J_3	J_4
M_2	J_3	J_4	J_2	J_1
M_3	J_2	J_3	J_1	J_4
M_4	J_4	J_3	J_1	J_2

Fig. 2: Representation of configurations

In the described implementation of TAMM, neighbourhood solutions are generated by exchanging two randomly chosen elements in the job sequence of a randomly chosen machine (*simple modification*) or by repeating this change n times (*n-fold modification* with $n \geq 2$). This mutation operator always generates job permutations as complete and - on machine level - permissible task sequences. Thus, local harmonization is not required in TAMM.

3.3 The Scheduling Process

The task sequences coded in configurations are scheduled by means of a simple heuristic algorithm. As a basic element of this algorithm a *schedule condition* is formulated which settles that a job i can be scheduled on a machine j if, and only if

- j is the current machine according to the machine sequence of job i and
- i is the current job according to the job sequence of machine j.

By means of this schedule condition the algorithm presented in Fig. 3 can be implemented. First, a quantity of all operations not yet scheduled (*unscheduled_operations*) is initialized with all operations to be scheduled (step (1)). An iteration of the scheduling process is introduced, if this quantity is not empty (step (2)). Within an iteration, the schedule condition is examined (step (5)) for each machine (loop (4)). Occasionally, a concerned operation is scheduled on the corresponding machine and removed from the quantity of unscheduled operations. Besides, a

flag called *at_least_one_operation_scheduled*, which has been initialized in step (3), indicates that in loop (4) an operation has been scheduled.

If in a run of loop (4) no operation could be scheduled on any machine, the underlying configuration contains a type of inconsistency which is manifested by mutually blocking task sequences of different machines. The elimination of such an inconsisteny is carried out by another repair algorithm introduced by Nakano and Yamada, called *global harmonization* (module GLOBAL_HARMONIZATION in step (6)).

```
BUILD_SCHEDULE
START
    initialize unscheduled_operations                                   (1)
    WHILE unscheduled_operations ≠ {} LOOP                              (2)
        at_least_one_operation_scheduled ← FALSE                        (3)
        FOR all machines LOOP                                           (4)
            IF schedule_condition (operation) THEN                      (5)
                schedule (operation)
                unscheduled_operations ← unscheduled_operations \ {operation}
                at_least_one_operation_scheduled ← TRUE
        IF NOT at_least_one_operation_scheduled THEN                    (6)
            call GLOBAL_HARMONIZATION
END
```

Fig. 3: Scheduling algorithm

Fig. 4 illustrates the problem of blocking using a simple example of processing two jobs on two machines. The rows of the machine sequence matrix indicate the sequence for each job (J_1 and J_2), the machines (M_1 and M_2) must be run through. The rows of the job sequence matrix (configuration) indicate the proposed processing order of both jobs for both machines.

Machine sequence:

	1st position	2nd position
J_1	M_1	M_2
J_2	M_2	M_1

Configuration:

	1st position	2nd position
M_1	J_2	J_1
M_2	J_1	J_2

Fig. 4: Blocking on schedule level

The interpretation of the configuration´s first row in Fig. 4 yields that J_1 has to wait in front of M_1 until processing of J_2 is finished. According to its machine sequence, J_2 is to be processed on machine M_2 first - and has to wait in front of that machine until processing of J_1 is finished (second row of the configuration matrix). However, this is not permissible, since according to

the second row of the machine sequence matrix J_1 is to be processed on M_1 before - and is still waiting there for J_2 to be finished.

Because the sketched type of inconsistencies on schedule level is not recognized before scheduling is performed, the mentioned repair algorithm is embedded into the scheduling process. Unlike local harmonization, global harmonization does not work on the original binary representation anymore, but on a derived symbolic representation like that presented in Fig. 4. Thus, the originally proposed repair algorithm could be integrated in TAMM.

In the case of a blocking, the task sequence of a single machine is changed during global harmonization in a way that a job is given priority which fulfils the schedule condition for the considered machine. For a job J_i which is still to be processed on a machine M_j the function

distance (J_i, M_j),

indicates the number of jobs preceding J_i according to the task sequence of M_j. By means of the function *distance*, the structure of the global harmonization can be depicted as follows (Fig. 5):

```
GLOBAL_HARMONIZATION
START
    FOR all J_i ∈ unscheduled_jobs LOOP                          (1)
        select next machine M_j from machine sequence of J_i
        compute distance (J_i, M_j)
    D_min ← min_i,j (distance (J_i, M_j))                        (2)
    select M_j* with distance (J_i, M_j*) = D_min for any J_i    (3)
    remove J_i from the job sequence of M_j*                     (4)
    shift first D_min jobs in the job sequence of M_j* for one position   (5)
    insert J_i in the first position of the job sequence of M_j*          (6)
END
```

Fig. 5: Global harmonization

First, for each element of a set of not completely scheduled jobs (*unscheduled_jobs*) the next machine, according to its machine sequence, is selected, and the value of the function *distance* is calculated for the selected machine (step (1)). The minimum D_{min} of all calculated *distance* values is determined (step (2)), and the machine holding that minimum is selected (step (3)). Because Nakano and Yamada do not indicate any tie breaking rule for the case that the minimal distance is determined for more than one job, in the considered implementation of TAMM such conflicts are solved by choosing the first-found minimum. The corresponding job holding that minimum is moved into the first position of the task sequence of the selected machine (steps (4) and (6)). Thereby, all the jobs to be processed before the considered job acoording

to the current task sequence of the machine are shifted by one position (step (5)). Subsequently, the scheduling algorithm resumes with step (2) in Fig. 3 using the modified task sequence.

Performing the described scheduling algorithm always results in a feasible schedule, in which starting and completion times are assigned to each operation. The maximum of the completion times of all last operations of the task sequences determines the value of the objective function (makespan) of the considered configuration - presupposed that processing starts at the time 0.

4 Test and Results

4.1 Data

TAMM has been tested using the data sets of Fisher and Thompson which have been established in literature as often used benchmarks (Fisher and Thompson 1963, pp. 236). Thus, comparisons with a variety of other algorithms for the job shop scheduling are possible. The results presented in the following affect the problems of processing 10 jobs on 10 machines (´10x10-problem´) as well as processing 20 jobs on 5 machines (´20x5-problem´). The optimum (minimum) values of the makespan are **930** units of time for the 10x10-problem and **1165** units of time for the 20x5-problem, respectively. The complexity of the considered test problems is illustrated by the fact, that determination of an optimum schedule for the 10x10-problem (including proof of optimality) succeeded for the first time only after more than 20 years of research (Carlier and Pinson 1989).

Results are presented for six different parametrizations of TAMM as listed in Tab. 1. Parameters are the starting value of the threshold (*threshold*), the number of iterations after which the threshold is reduced (*change_threshold_all*), and the termination criterion (*max_unsuccessful_trials*). Since in the calculation of the makespan only integer values can occur, the value of *threshold* is always reduced for exactly one unit (parameter *threshold_lower_step* in Fig. 1). The last row in Tab. 1 indicates the number of repetitions (*runs*) of TAMM with the respective parameter setting.

	set 1	set 2	set 3	set 4	set 5	set 6
threshold	15	100	30	10	5	30
change_threshold_all	10,000	1,000	1,000	10,000	1,000	10,000
max_unsuccessful_trials	30,000	10,000	3,000	20,000	3,000	100,000
runs	1,000	200	200	200	200	100

Tab. 1: Parameter settings

4.2 Solution Qualities and Runtimes

Tables 4 and 5 display the results of the tests for both problems with regard to the achieved solution qualities and required runtimes. The results refer to a Pascal-written implementation of TAMM on a PC 80486 (66 MHz). As to solution qualities, the following values are cited:

- the best solution of all runs (*best solution*, optimum values are marked by bold types),
- the mean of the solutions (*mean solution*),
- the average deviation of the solutions from the optimum in percent (*average dev. from opt. [%]*),
- the standard deviation of the solutions (*standard deviation*),
- the mean variation of the solutions (*mean* va*riation*),

and concerning runtimes, respectively:

- the mean runtime in seconds (*mean runtime$_{total}$ [sec]*),
- the mean runtime for detecting the best solution in seconds (*mean runtime$_{best}$ [sec]*).

	set 1	set 2	set 3	set 4	set 5	set 6
best solution	**930**	951	979	**930**	1015	951
mean solution	1000.87	1045.58	1046.48	1006.54	1120.86	998.26
average dev. from opt. [%]	7.62	12.43	12.52	8.23	20.52	7.34
standard deviation	24.43	26.03	23.63	23.85	45.22	22.59
mean variation	596.65	677.80	558.17	568.88	2044.53	510.19
mean runtime$_{total}$ [sec]	273.95	192.43	54.11	193.57	13.26	646.13
mean runtime$_{best}$ [sec]	180.39	173.99	47.68	112.53	8.84	438.19

Tab. 2: Results for the 10x10-problem

	set 1	set 2	set 3	set 4	set 5	set 6
best solution	**1165**	1198	1192	**1165**	1264	1173
mean solution	1205.48	1296.34	1289.59	1212.74	1380.76	1202.94
average dev. from opt. [%]	3.47	11.27	10.69	4.10	18.52	3.26
standard deviation	21.18	35.45	34.97	25.47	40.19	21.75
mean variation	448.78	1256.71	1222.88	648.67	1615.15	472.88
mean runtime$_{total}$ [sec]	390.00	268.00	76.34	264.55	20.37	914.15
mean runtime$_{best}$ [sec]	285.07	244.62	69.00	186.83	14.50	641.10

Tab. 3: Results for the 20x5-problem

The shortest runtimes measured for detecting the optimum makespan values of 930 and 1165, respectively, have been **97.22** seconds for the 10x10-problem (set 4) and **178.28** seconds for the 20x5-problem (set 4).

The cited parameter settings (Tab. 1) were not determined analytically, but more arbitrarily in a trial-and-error-procedure: First, the only requirement for any combination of parameter-values has been a maximum runtime of 10 minutes (600 seconds). Set 1 obtains best results for both problems in that case. Later, comparisons with parameter settings leading to mean runtimes of clearly over 10 minutes showed that the results of set 1 could hardly be improved within the realized implementation of TAMM. Set 6 serves as an example: An extension of the runtime ($mean\ runtime_{total}$) from approximately 4.5 to almost 11 minutes (10x10-problem) and from approximately 6.5 to over 15 minutes (20x5-problem), respectively, yields only slight improvements of mean solution quality (just about approximately 2.6 units of time) as well as of the standard deviation and mean variation values.

Comparison of set 2 and set 3 elucidates that TAMM reacts very sensitively to the choice and combination of the parameter values despite its simple parametrization. Set 3 leads to comparable results in less than 30 % of the runtime on average: For the 10x10-problem almost identical mean solution values are determined as well as an even somewhat slighter deviation; however, the best solution found is 28 units of time worse, when compared to set 2. For the 20x5-problem set 3 even dominates with regard to the presented results, since all values of the best solution found as well as of the mean solution quality as well as of standard deviation and mean variation are better than those determined with set 2.

Set 4 obtains the best results with regard to determination of optimum solution qualities, since these values are found faster than with set 1. Set 5 serves as an example for results TAMM is able to produce facing restrictive runtime-requirements.

4.3 Further Results

4.3.1 Impact of the Starting Configuration´s Quality

The results presented above have been produced by creating the starting configuration randomly, i.e. permutations of job sequences are created randomly for each machine. As a rule, TAMM starts with configurations of quite a bad quality (see Tab. 4). To explore the influence of the quality of the configuration TAMM is starting with, two alternatives were tested on the 10x10-problem:

1. The algorithm of Giffler and Thompson (Giffler and Thompson 1960) is used to produce starting configurations. The algorithm always generates an active schedule as an schedule, in which it is not possible to start the processing of any operation earlier without delaying the processing of at least one other operation. The job sequences, which are

implicitly contained in the generated schedule serve as an initial configuration for TAMM. Subsequently, TAMM is started with parameter values of set 4.

2. TAMM is running in a two-phased process: First, it is started with set 5, terminating after approximately 13 seconds on average (see Tab. 2). Subsequently, it is restarted with set 4 using the solution calculated in the first phase as an initial configuration.

The results of the tests - each repeated 200 times (200 runs) - are summed up in Tab. 4 (*set 4* (*G&T*) and *set 5 & set 4*), and are opposed to the results of pure set 4 (see Tab. 2). In addition to already introduced values, Tab. 4 indicates the mean value of the starting configuration´s quality (*mean initial solution*) as well as the simple correlation coefficient between the starting configuration´s quality and the best solution found (*correlation coefficient*).

	set 4	set 4 (G&T)	set 5 & set 4
best solution	930	936	952
mean solution	1006.54	1004.51	1007.91
average dev. from opt. [%]	8.23	8.01	8.38
standard deviation	23.85	31.92	21.99
mean variation	568.88	1018.72	483.46
mean $runtime_{total}$ [sec]	193.57	186.48	191.93
mean $runtime_{best}$ [sec]	112.53	104.02	109.91
mean initial solution	3329.55	1334.57	1112.60
correlation coefficient	- 0.002	+ 0.003	+ 0.004

Tab. 4: Results for different initial configurations (10x10-problem)

According to the values of Tab. 4, a significant correlation between the initial configuration´s quality and the best solutions found does not exist. Fig. 6 displays the convergence of TAMM as development of the solution quality in the lapse of time, for both a randomly chosen run and an optimum run, respectively (set 4, 10x10-problem). The graph provides a rather plausible explanation for the above mentioned result: Even with set 4 TAMM reaches after only a few seconds the same solution qualities it was initialized with in the above mentioned alternatives .

There is no satisfactory explanation for another phenomenon: the extraordinary high deviation values resulting when TAMM is initialized with configurations which have been generated with the algorithm by Giffler and Thompson. It can only be supposed that an initialization with configurations representing active schedules tends to hinder the algorithm: Because active schedules are somehow pre-optimized with regard to their inner structure, it might be more difficult to achieve improvements by simply exchanging two operations. Anyhow, an empirical confirmation of this hypothesis has yet to be given. Fig. 7 gives a confrontation of the convergences of a randomly chosen run with initialization by the Giffler/Thompson-algorithm (*set 4*

(*G&T*)) and without that initialization (*set 5 & set 4*), each for the first 10 seconds (within this timeslot the (*set 5 & set 4*)-run equals a pure (*set 5*)-run, because in the presented run an exchange to parameter values of set 4 is carried out not until 12.8 seconds). The graph shows, however, that convergence is actually very slow in the area of the (active) initial solution, and particularly slower than that one of the (*set 5 & set 4*)-run at about the same level of quality.

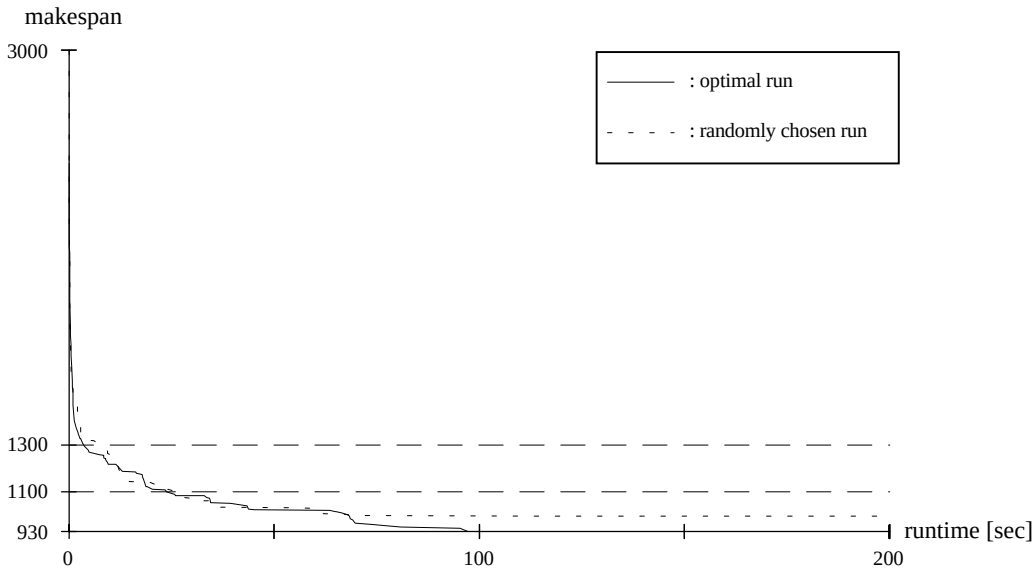

Fig. 6: Convergence of solution quality (10x10-problem, set 4)

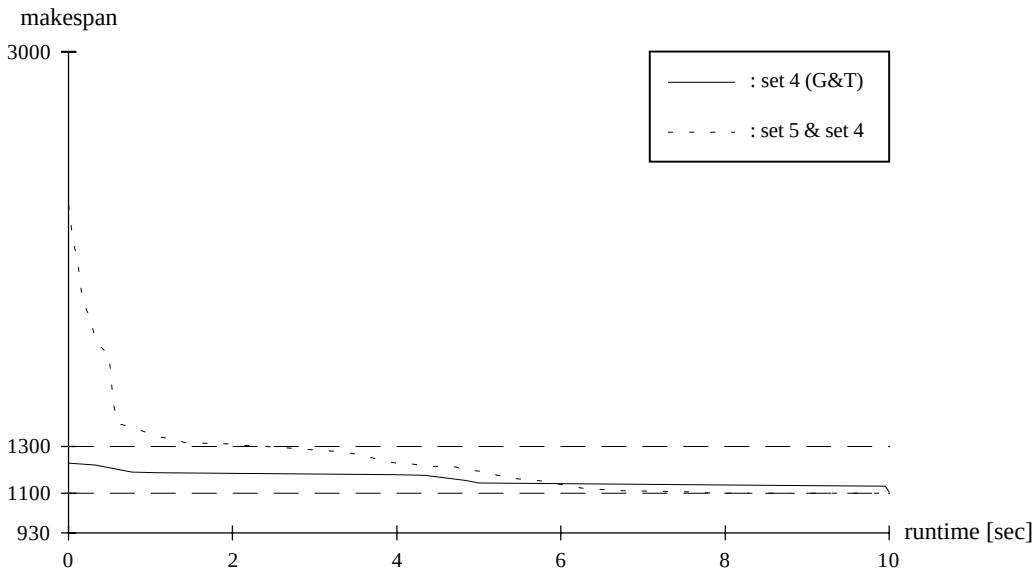

Fig. 7: Convergence of solution quality for *set 4* (*G&T*) and *set 5 & set 4*

4.3.2 Some Remarks upon the Performance of TAMM

Above all, mutation-selection approaches have achieved promising results in applications for traveling salesman problems (e.g. Dueck and Scheuer 1990 or Dueck 1993). In symmetric traveling salesman problems, modified configurations can easily be produced by breaking up

two or more connections within a tour and knotting new connections between the involved nodes such that again a valid tour is produced. A feature of that procedure is that the length of a new tour does not have to be calculated completely new. Instead, the length of the old tour is decreased by the sum of the lenghts of broken-up connections and increased by the lengths of newly knotted connections which can directly be taken from a given distance table. In contrast, to determine the quality of a new configuration for a scheduling problem, it is generally necessary to create a completely new schedule - even after slight changes of the job sequences as the proposed exchange of only two operations in the job sequence of one machine.

In the described implementation of TAMM, the number of accomplished exchanges of operations - and thereby of required complete scheduling runs - amounts partly to a few hundred thousands. Using set 6, for example, the threshold is reduced down from a value of 30 every 10,000 iterations. Thus, a run reaching a threshold of 0 requires the building of at least 300,000 complete schedules. The performance of the scheduling algorithm thus has an essential influence on the performance of TAMM.

The implemented heuristic scheduling algorithm requires 0.615 milliseconds on average to build up a schedule for the 100 operations of the 10x10-problem. This holds under the condition that the underlying configuration represents a permissible solution, so no repair-functionality is needed to create valid task sequences.

The number of configurations representing inadmissible solutions, resulting from arbitraryly exchanging operations in the task sequences, is another influential factor to the performance of the system. A test showed that simple modifications of admissible configurations produce inadmissible ones with a probabilty of approximately 34 % - forcing calls of the repair algorithm (global harmonization) in these cases. The required runtime for building up a schedule increases up to 0.93 milliseconds on average. It should be pointed out that not only the inadmissibility of a configuration is responsible for the increase of runtime requirement, but also the ´level of inadmissibility´ which is indicated by the frequency the repair algorithm is called while generating a schedule. Additionally, a (small) part of the increase from 0.615 up to 0.93 milliseconds is caused by the process of the modification itself, although this part is less than 0.02 milliseconds.

Using 2-fold modifications the quota of inadmissible configurations increases to approximately 44.5 % and the resulting runtime required for building up a schedule to 1.14 milliseconds on average. Although allowed, multi-fold modifications were not considered in further tests, since they showed worse convergence as opposed to the simple modification in some early tests.

Building up a schedule with the algorithm of Giffler and Thompson requires 1.63 milliseconds on average. Anyway, the algorithm is less suitable as a scheduling algorithm within TAMM, since as input it receives only the (given) machine sequences and the (given) processing times

as well as an operator which rules the selection of an operation to be scheduled on a specific machine in cases of conflict (e.g. a priority rule). A *conflict-set* comprises all operations which could be processed currently at a specific machine and whose earliest possible processing times overlap with the processing time of a *reference operation*. A conflict-set is always formed for that machine on which the processing of any operation could be finished earliest. The corresponding operation is the reference operation. Thus, variations of resulting schedules can only be generated by varying the selection operator (that inculdes the possibility of solving each conflict individually). However, a changed choice of a specific operation from a conflict-set generally implies changes of the compositions of all the conflict-sets to be formed afterwards. As a result, the control of the search process is lost, because it is no longer possible to generate ´similar´ configurations in a given neighbourhood.

By contrast, within TAMM, a (limited) similarity of configurations can be supposed in accordance with neighbourhood definitions for continuous functions: From this follows that configurations which are similar with regard to their representation must have similar solution qualities, too. Similarity of representation could be measured, for example, as the number of operations holding different positions in the task sequences of two configurations. This ´similarity supposition´ is slightly supported by the approximately continuous convergence of TAMM (see Fig. 6). Supposed that no correspondence between representation and solution quality can be ascertained (like argued for a combination of TAMM and the Giffler/Thompson-algorithm above), any search algorithm might behave as a random search process.

4.3.3 The Topology of the Solution Space

In view of missing techniques for the visualization of multi-dimensional spaces illustrations of solution spaces of complex optimization problems often resort to the idea of a mountain range. Hikers in the mountain range symbolize algorithms in search of the highest peak (maximization problems) or the deepest valley (minimization problems). Even if this kind of illustration might be quite a vivid one, it seems not to be very suitable, since it cannot always explain the success (or even the failure) of the algorithms.

Regarding the 10x10-problem and set 4, the hiker could be e.g. a parachutist landing on a plateau at an altitude of about 3,300 meters (*mean initial solution* in Tab. 4: 3,329.55) and proceeding in search of the deepest valley, which can be found at an altitude of exactly 930 meters. Thus, he has to overcome a difference in altitude of approximately 2400 meters. Considering e.g. that set 4 does not allow any step leading more than 10 meters (*threshold* in Tab. 1) in height and that this limitation gets the more restrictive the lower the hiker comes, it appears impossible to leave even smaller valleys once they have been reached. Despite this TAMM almost always reaches valleys not more than 120 m above the deepest valley.

Two possible explanations of this phenomenon could be:

- The idea of a hiker is not appropriate: On one hand, no continuous surface to move on exists. On the other hand, due to the imperfect neighbourhood definition generating new configurations does not neccessarily correspond to little steps in the ´mountain range of quality´. The hiker rather jumps than walks - turning back to the starting point, if the calculated landing point is located higher than the current location plus the current threshold.

- Many more deep valleys seem to be located in the ´mountain range of quality´ than one might suppose from geography. This hypothesis was confirmed through a simple test: For each of the both problems one optimal configuration (with a makespan value of 930 and 1165, respectively) was taken as an initial configuration. Following, neighbourhood solutions were generated using simple modifications until another solution with optimum makespan was found. This procedure was repeated 500 times for both problems. Result: for the 10x10-problem, 403 different solutions with a makespan of 930 were obtained and for the 20x5-problem even 485 different solutions with a makespan of 1165.

5 Comparison with Other Approaches

Concluding, the presented results will be compared to the results of some other authors using the same ´problem artefacts´ of Fisher and Thompson presented in section 4. These approaches are:

- Both variants of the shifting-bottleneck-procedure of Adams, Balas and Zawack (Adams, Balas, and Zawack 1988) (ABZ1 and ABZ2)

  The first variant of the shifting-bottleneck-procedure (ABZ1) is based on an iterative process optimizing the task sequence of a current bottleneck machine locally, in the sense of a one-machine-problem. In a following step, the task sequences of the other machines, even if determined beforehand, are (re-)optimized while leaving the task sequence of the current bottleneck machine unchanged. The second variant of the shifting-bottleneck-procedure (ABZ2) uses the first variant to determine an optimum path in a partial search-tree through the sequences the non-bottleneck-machines will be (re-)optimized in.

- The algorithm of Fang, Ross and Corn (Fang, Ross, and Corne 1993) (FRC)

  The algorithm FRC is a genetic algorithm using a symbolic representation (linear lists of operations) as well as some recombination operators which have been developed especially for sequencing problems. As sketched for TAMM, generation of schedules is done by a simple heuristic scheduling algorithm.

- Two algorithms of Nakano and Yamada (NY and YN)

  The two approaches of Nakano and Yamada are genetic algorithms, too. The basic ideas of the algorithm NY (Nakano and Yamada 1991) have been introduced in section 3. The approach of the algorithm YN (Yamada and Nakano 1992) is based on a symbolic representation as well as on a distinct crossover operator, using the algorithm of Giffler and Thompson to recombinate new configurations.

- Two algorithms of Dorndorf and Pesch (Dorndorf and Pesch 1992) (DP1 and DP2)

  The approaches of Dorndorf and Pesch are (hybrid) genetic algorithms, too. The first one (DP1) uses the algorithm of Giffler and Thompson to generate schedules as well as the mentioned before algorithm YN. During the evolutionary process of the genetic algorithm, sequences of priority rules are produced which can be used to solve conflicts within the scheduling process of the Giffler/Thompson-algorithm. The second algorithm (DP2) is based on the second variant of the shifting-bottleneck-algorithm. Genetic search aims at an optimum sequence of (re-)optimizing the task sequences of individual machines.

- The algorithm of Carlier and Pinson (Carlier and Pinson 1989) (CP)

  The algorithm of Carlier and Pinson is a ´classical´ branch&bound approach. A schedule is generated during the branching-procedure, determining exactly one of the two possibilities of processing two operations on a machine in each step. The bounding within the algorithm is based on the solution of one-machine-problems.

- The algorithm of Barker and McMahon (Barker and McMahon 1985) (BM)

  The algorithm of Barker and McMahon also represents a ´classical´ branch&bound approach. Contrary to Carlier and Pinson, a node of the search-tree always represents a complete schedule. New schedules are produced from given ones by modifying the sequence of jobs within a distinct *critical block* of the schedule. A critical block is formed by a set of operations which are successively processed on a machine. The last operation of the critical block is the first operation of the schedule which is finished in time with the current minimum of the makespan or later. As with the approach of Carlier and Pinson, bounding is rested on the solution of one-machine problems.

Tables 7 and 8 display the results of the mentioned approaches for the 10x10- and the 20x5-problem, respectively, comparing them with the results achieved by TAMM with set 4. Runtimes values are rounded off, if necessary, and found optimum values as well as corresponding runtime values are marked by bold types. For TAMM the shortest runtime for detecting optimum values is occasionally given in brackets. Exact runtime values for TAMM are 184.5

(97.22) seconds and 277.31 (178.28) seconds, respectively. The term *n.v.* states that no value is given in the original literature.

	FRC	NY	YN	DP1	DP2	CP	BM	ABZ1	ABZ2	TAMM
best solution	949	965	**930**	960	938	**930**	960	1015	**930**	**930**
runtime [sec]	<1500	n.v.	600	932	106	3305	193	10	851	**184 (97)**

Tab. 5: Comparison of solution qualities for the 10x10-problem

	FRC	NY	YN	DP1	DP2	CP	BM	ABZ1	ABZ2	TAMM
best solution	1189	1215	1184	1249	1178	**1165**	1303	1290	1178	**1165**
runtime [sec]	<1800	n.v.	n.v.	1609	95	1234	132	3	80	**277 (178)**

Tab. 6: Comparison of solution qualities for the 20x5-problem

Remarks:

- Due to different computer systems serving as bases for the implementation and the test of the different approaches, no direct comparability of the stated runtime values is given. The following computer systems are indicated by the authors:

  - FRC: SUN-4
  - YN: SUN SPARCstation 2
  - DP1, DP2: DECstation 3100
  - CP: PRIME 2655
  - BM: Cyber 171
  - ABZ1, ABZ2: VAX 780/11

  For the algorithm NY no details about the used hardware are given. Assuming that with regard to their computational power all listed systems dominate that one used for TAMM (PC 80486, 66 MHz), the runtime values listed above underline the quality of the results achieved with TAMM.

- The branch&bound algorithm CP proves the optimality of the calculated solutions. Consideration of these proofs leads to runtimes of 17,985 seconds (!) (10x10-problem) and 1,448 seconds (20x5-problem), respectively.

- The good results of both variants of the shifting-bottleneck-procedure (ABZ1 and ABZ2) could not always be confirmed in reimplementations of other authors (see e.g. Dorndorf and Pesch 1992). Within their reimplementation, Dorndorf and Pesch achieved combinations of solution qualities and runtimes for ABZ1 and ABZ2 as follows: 1031/0.5 seconds and 951/186 seconds (10x10-problem) as well as 1274/0.4 seconds and 1240/10 seconds (20x5-problem).

- Details given about the algorithm NY are limited to the achieved solution qualities. The algorithm has been reimplemented by Rohmann on a SUN SPARCstation 10 (Rohmann 1993). Rohmann runs tests over sections of the 10x10-problem (so-called scenarios, each containing a subset of the 10 jobs) and stated average runtime values of 51 minutes for scenarios with 50 operations (or 5 of the 10 jobs, respectively). For a single run using the entire 10x10-problem, a runtime of 3 hours and 20 minutes is stated. Within this single run, the parameters proposed by Nakano and Yamada, i.e. a population size of 1000 and 150 generations, have been used, while tests over the mentioned scenarios were accomplished with a population size of 500 over 300 generations. Nakano and Yamada give no details about further parameters, e.g. about crossover and mutation rates, selection mechanisms etc.

  Since TAMM surpassed these results of NY considerably, the initially announced extensive comparison of both algorithms has been omitted.

- Although the algorithm of Barker and McMahon (BM) essentially is an optimizing algorithm, optimum values are not found since runtime limitations were introduced when calculating lower boundaries for the makespan.

6 Conclusion and Outlook

The results achieved with TAMM demonstrate that within the application for complex optimization problems good approximizations of optimum solutions can be achieved by means of simple mutation-selection approaches. On one hand, regarding the results for the well-known benchmark-problems of Fisher and Thompson, TAMM takes its place amongst the best known algorithms for job shop scheduling. Particularly, none of the compared algorithms found the optimum values for the given problems in similarly short times - although TAMM has been implemented on the comparatively weak hardware basis of a PC 80486. On the other hand, the high deviation of solution qualities indicates a significant weakness of the approach.

Up to now no parameter tuning has been introduced. More, arbitrarily chosen parameter values were tried in single runs and the most encouraging combinations of parameter values were selected afterwards. An adjustment of these values resulted only from the effort to admit termination of the algorithm not until a threshold level of 0 is reached. Thus, systematic parameter tuning will be an objection to further development of TAMM. Particularly, possibilities of self-adaption of the parameters should be explored. This appears to be a promising approach, especially because of the simple parametrization of TAMM compared to genetic algorithms or to simulated annealing.

Mutation of configurations offers another starting point for improvements: In the presented version of TAMM the operations affected by a mutation are selected by sheer luck. For com-

parison the implementation of a bottleneck-sensitive version is planned that introduces a modification operator preferring exchanges of operations lying on a critical path in a graph-oriented description of the underlying schedule.

With regard to the practical applicability of TAMM, the adaptibility of the algorithm must be proved. This is particularly valid, because a proof of effectiveness using artificial problem-data can at most be a necessary, not however a sufficient condition for applicability in operational systems. For instance, from the point of view of efficiency employment of TAMM as a short-time scheduling module within an interactive *leitstand system* is conceivable. In contrast, a proof of adaptability to the specific requirements in a real-world planning environment is still to be produced.

References

Aarts, E. and J. Korst (1989): Simulated Annealing and Boltzmann Machines, A Stochastic Approach to Combinatorial Optimization and Neural Computing, Chichester et al.

Ablay, P. (1987): Optimieren mit Evolutionsstrategien, Spektrum der Wissenschaft, 7, 104-115.

Adams, J., E. Balas, and D. Zawack (1988): The Shifting Bottleneck Procedure for Job Shop Scheduling, Management Science, 34, 3, 391-401.

Barker, J.R. and G.B. McMahon (1985): Scheduling the General Job-Shop, Management Science, 31, 5, 594-598.

Carlier, J. and E. Pinson (1989): An algorithm for solving the job-shop problem, Management Science, 35, 2, 164-176.

Dorndorf, U. and E. Pesch (1992): Evolution Based Learning in a Job Shop Scheduling Environment, Research Memorandum 92-019, University of Limburg, Limburg.

Dueck, G. (1993): New Optimization Heuristics - The Great Deluge Algorithm and the Record-to-Record Travel, Journal of Computational Physics, 104, 86-92.

Dueck, G. and T. Scheuer (1990): Threshold Accepting: A General Purpose Optimization Algorithm Superior to Simulated Annealing, Journal of Computational Physics, 90, 161-175.

Fang, H.-L., P. Ross, and D. Corne (1993): A Promising Genetic Algorithm to Job-Shop Scheduling, Rescheduling, and Open-Shop Scheduling Problems, in: Proceedings of the Fifth International Conference on Genetic Algorithms, S. Forrest (ed.), San Mateo, 375-382.

Fisher, H. and G.L. Thompson (1963): Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules, in: Industrial Scheduling, J.F. Muth and G.L. Thompson (eds.), Englewood Cliffs, 225-251.

Forrest, S. (ed.) (1993): Proceedings of the Fifth International Conference on Genetic Algorithms, San Mateo.

French, S. (1982): Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop, Chichester.

Giffler, B. and G.L. Thompson (1960): Algorithms for Solving Production-Scheduling Problems, Operations Research, 8, 487-503.

Holland, J.H. (1975): Adaptation in natural and artificial systems, Ann Arbor.

Nakano, R. and T. Yamada (1991): Conventional Genetic Algorithm for Job Shop Problems, in: Proceedings of the Fourth International Conference on Genetic Algorithms, R. Belew and L. Booker (eds.), San Mateo, 474-479.

Rohmann, T. (1993): Reihenfolgeplanung mit Genetischen Algorithmen am Beispiel der Maschinenbelegungsplanung, Dissertation, University of Münster, Institute of Business Informatics, Münster 1993.

Yamada, T. and R. Nakano (1992): A Genetic Algorithm Applicable to Large-Scale Job-Shop Problems, in: Parallel Problem Solving from Nature 2, Männer, B. and B. Manderick (eds.), Amsterdam et al., 281-290.

UNIVERSITY OF LEIPZIG

Institute of Production Management and Industrial Information Management

- Reports -

No. 1: Zelewski, Stephan: Das Konzept technologischer Theorietransformationen - eine Analyse aus produktionswirtschaftlicher Perspektive, Leipzig 1994.

No. 2: Siedentopf, Jukka: Anwendung und Beurteilung heuristischer Verbesserungsverfahren für die Maschinenbelegungsplanung - Ein exemplarischer Vergleich zwischen Neuronalen Netzwerken, Simulated Annealing und genetischen Algorithmen, Leipzig 1994.

No. 3: Zelewski, Stephan: Unternehmenskrisen und Konzepte zu ihrer Bewältigung, Leipzig 1994.

No. 4: Siedentopf, Jukka: Ein effizienter Scheduling-Algorithmus auf Basis des Threshold Accepting, Leipzig 1995.

No. 5: Zelewski, Stephan: Petrinetzbasierte Modellierung komplexer Produktionssysteme (Projekt PEMOPS), Band 1: Exposition, Leipzig 1995.

No. 6: Zelewski, Stephan: Petrinetzbasierte Modellierung komplexer Produktionssysteme (Projekt PEMOPS), Band 2: Bezugsrahmen, Leipzig 1995.

No. 7: Zelewski, Stephan: Petrinetzbasierte Modellierung komplexer Produktionssysteme (Projekt PEMOPS), Band 3: Einführung in Stelle/Transition-Netze, Leipzig 1995.

No. 8: Zelewski, Stephan: Petrinetzbasierte Modellierung komplexer Produktionssysteme (Projekt PEMOPS), Band 4: Verfeinerungen von Stelle/Transition-Netzen, Leipzig 1995.

No. 9: Zelewski, Stephan: Petrinetzbasierte Modellierung komplexer Produktionssysteme (Projekt PEMOPS), Band 5: Einführung in Synthetische Netze, Teilband 5.1: Darstellung des Kernkonzepts, Leipzig 1995.

No. 10: Zelewski, Stephan: Petrinetzbasierte Modellierung komplexer Produktionssysteme (Projekt PEMOPS), Band 5: Einführung in Synthetische Netze, Teilband 5.2: Auswertungsmöglichkeiten, Leipzig 1995.

No. 11: Zelewski, Stephan: Petrinetzbasierte Modellierung komplexer Produktionssysteme (Projekt PEMOPS), Band 6: Erweiterungen von Synthetischen Netzen, Leipzig 1995.

No. 12: Zelewski, Stephan: Petrinetzbasierte Modellierung komplexer Produktionssysteme (Projekt PEMOPS), Band 7: Fallstudie, Leipzig 1995.

No. 13: Zelewski, Stephan: Petrinetzbasierte Modellierung komplexer Produktionssysteme (Projekt PEMOPS), Band 8: Charakterisierung des Petrinetz-Konzepts, Leipzig 1995.

No. 14: Zelewski, Stephan: Petrinetzbasierte Modellierung komplexer Produktionssysteme (Projekt PEMOPS), Band 9: Beurteilung des Petrinetz-Konzepts, Leipzig 1995.

No. 15: Zelewski, Stephan: Petrinetzbasierte Modellierung komplexer Produktionssysteme (Projekt PEMOPS), Band 10: Petrinetz-Literatur, Leipzig 1995.

No. 16: Siedentopf, Jukka: An Efficient Scheduling Algorithm Based upon Threshold Accepting, Leipzig 1995.

No. 17: Siedentopf, Jukka: The Threshold Waving Algorithm for Job Shop Scheduling, Leipzig 1995.