Available online at www.sciencedirect.com
SCIENCE DIRECT®

ELSEVIER

European Journal of Operational Research 155 (2004) 616–630

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/dsw

Discrete Optimization

An efficient genetic algorithm for job shop scheduling with tardiness objectives

Dirk C. Mattfeld [a,*], Christian Bierwirth [b]

[a] *Faculty of Business Studies and Economics, University of Bremen, FB Wirtschaftswissenschaft, Box 330440, Bremen 28334, Germany*

[b] *Faculty of Economics, Martin-Luther-University, Halle-Wittenberg, Germany*

Received 19 October 2001; accepted 18 December 2002

Abstract

We consider job shop scheduling problems with release and due-dates, as well as various tardiness objectives. To date, no efficient general-purpose heuristics have been developed for these problems. genetic algorithms can be applied almost directly, but come along with apparent weaknesses. We show that a heuristic reduction of the search space can help the algorithm to find better solutions in a shorter computation time. Two ways of reducing a search space are investigated by considering short-term decisions made at the machine level and by long-term decisions made at the shop floor level.

Keywords: Genetic algorithm; Job shop scheduling; Tardiness objectives; Tunable schedule builder; Multi-stage decomposition

1. Introduction

A job shop problem considers n jobs which arrive at the shop at certain points of time, referred to as release-dates. The point in time where a job is projected for customer delivery is called its due-date. The problem consists of scheduling the jobs on m machines with respect to technological constraints whilst pursuing a given objective. Besides flow-time dependent objectives, tardiness objectives are considered particularly in the event that due-dates are tight. We refer to problems of this type as tardiness problems.

In this paper we propose a genetic algorithm (GA) for job shop scheduling problems with release and due-dates and with various tardiness criteria as objectives. Typically, GAs applied to scheduling tend to lengthy runtimes and produce only a relatively poor solution quality. Therefore we employ the GA to search a heuristically derived subset of the original search space, expecting that the benefits of searching smaller spaces offset the potential losses of excluding a portion of the search space. This idea is investigated in two directions. First, a complexity reduction is achieved by narrowing the scope at the machine level by means of the schedule builder. Second, a problem decomposition at the shop floor level by means of

[*] Corresponding author. Tel.: +49-421-218-2011; fax: +49-421-218-4271.

E-mail addresses: dirk@logistik.uni-bremen.de (D.C. Mattfeld), bierwirth@wiwi.uni-halle.de (C. Bierwirth).

a multi-stage approach is considered. The effects of both approaches are investigated independently before they are combined.

In Section 2 we discuss previous work and related approaches. In Section 3 we introduce the job shop model and describe a benchmark set. In Section 4 we discuss priority rules suitable for tardiness problems and present results obtained by applying these rules probabilistically. In Section 5 we present a GA for this problem and compare results obtained from two different schedule builders. In Section 6 we propose a tunable schedule builder which is capable of scaling the search space. In Section 7 we describe a multi-stage decomposition to further reduce the search space. Experiments reveal a marked improvement in GA efficiency without any deterioration in solution quality. Section 8 compares the results obtained with those found in the literature yielding new best solutions in 13 of 24 cases. In Section 9 we summarize and conclude.

2. Related work

Research in job shop scheduling has predominantly concentrated on a simplified optimization model. In this model all jobs have identical release-dates such that each job can start immediately. Furthermore due-dates are assumed loose and are therefore not considered at all. The objective is to minimize the makespan which aims at reducing the completion time of the final job. Very efficient heuristics have been developed for the minimum makespan problem (Błażewicz et al., 1996). Many algorithms benefit from a schedule representation known as the "disjunctive graph formulation" (Adams et al., 1988). Particularly local-search algorithms like Simulated Annealing and Tabu Search have been applied with great success. It turned out that general purpose methods like GAs are only second best choice among the modern heuristics, particularly if applied to large makespan problems (Anderson et al., 1997).

For tardiness objectives, several local search approaches have been reported. With respect to the minimization of the total tardiness of jobs, a heuristic exchange neighborhood of asymptotic time complexity $O(n^2m^2)$ is used (He et al., 1996). This neighborhood is engaged in a Simulated Annealing algorithm resulting in an effective but time consuming search. Also Tabu Search has been applied to tardiness problems (Valls et al., 1998). In order to avoid computationally expensive neighborhood evaluations the authors propose a makespan minimization by treating the due-dates as constraints. The task for Tabu Search is to obtain a feasible schedule or a good compromise schedule in the event that the violation of due-dates cannot be circumvented. Tabu Search is also applied for directly pursuing the mean absolute lateness as objective (James and Buchanan, 1998). Efficient neighborhood definitions based on job sequences and binary schedule representations are compared. Unfortunately the approach is limited to single machine problems. Recently, a neighborhood definition originated for makespan problems has been used for minimizing the weighted sum of tardiness. A random perturbation of schedules relying on this neighborhood leads to encouraging results for the particular objective pursued (Kreipl, 2000).

Despite some progress gained by local-search algorithms so far, it is still unknown how to derive efficient neighborhood definitions for tardiness problems. The approaches sketched above lack efficiency because the neighborhoods are either too large or they are reasonably sized but the neighborhood definition is not goal-oriented. This motivates to consider general purpose procedures for tardiness problems. Unlike local-search algorithms, GAs can be easily modified to handle release-dates, due-dates and alternate performance measures (Fang et al., 1996; Lin et al., 1997; Norman and Bean, 1999). Besides of static environments a GA has also been applied to problems where jobs have release-dates that are not known in advance (Bierwirth and Mattfeld, 1999). The tunable schedule builder, which is also used in this paper, leads to satisfying results for minimizing the mean flow time of jobs.

3. Tardiness problems

In industrial production, jobs are released with respect to capacity constraints, material supplies and customer demands. A job release-date denotes its earliest possible starting time, while its due-date indicates the point in time at which the job should be completed to adhere to the projected date of delivery. Furthermore, priorities express the effort to be spent on completing a job on time with respect to the importance of a certain customer. In this context, makespan minimization is no economically relevant measure of schedule performance. A flow-time oriented approach can be used instead, maintaining a moderate work-in-process. Due-date oriented criteria are considered to improve customer service-levels.

3.1. Problem definition and objectives

The general job shop model considers n jobs to be processed on m machines. The processing of a job on a certain machine is referred to as an operation. The operation processing times are fixed and known in advance. Every job passes every machine exactly once in a prescribed order, resulting in a total of $n \times m$ operations. In this research we modify the above assumptions by considering

- non-zero release-dates r_i of jobs,
- prescribed job due-dates d_i, and
- priorities among jobs, expressed by weights $\omega_i > 0$.

No job can start before its release-date r_i and its processing should not exceed its due-date d_i. A job does not necessarily pass every machine and it may pass a machine more than once. The following objectives are treated:

- the minimization of the weighted mean flow-time of jobs $\overline{F}$,
- the minimization of the weighted mean tardiness of jobs $\overline{T}$,
- the minimization of the maximum tardiness of jobs $T_{\max}$, and
- the minimization of the weighted number of tardy jobs T_{n}.

A definition of the weighted mean flow-time refers to the completion times c_i of jobs, which are determined by the starting time of the final operation of a job plus the associated operation processing time. This figure is used in the well-known measure $\overline{F} = (1/n)\sum_{i=1}^{n} \omega_i(c_i - r_i)$.

The tardiness of a job indicates the time span the completion time overshoots the projected due-date. Let $T_i = \max(c_i - d_i, 0)$ denote the tardiness of a job. Based on this measure, the weighted mean tardiness is defined as $\overline{T} = (1/n)\sum_{i=1}^{n} \omega_i T_i$. Similarly, the worst violation of due-dates is expressed by $T_{\max} = \max\{T_i | 1 \leqslant i \leqslant n\}$.

Finally, let $\mathrm{tardy}(i)$ indicate whether job i is tardy or not, i.e. $\mathrm{tardy}(i) = 1$ means job i is tardy ($T_i > 0$) while $\mathrm{tardy}(i) = 0$ means it is completed in time. The weighted number of tardy jobs is simply given by $T_{\mathrm{n}} = \sum_{i=1}^{n} \omega_i \,\mathrm{tardy}(i)$. This measure is frequently met in practise because it has a direct relation to the β-service-level expressing the percentage of on-time deliveries.

3.2. Benchmark set

For a computational study we rely on 12 scheduling scenarios provided with the "Parsifal" software package in Morton and Pentico (1993). For each scenario, referred to as an instance in the following, the four above defined objectives are considered. This leads to a total of 48 scheduling problems. It can be taken from Table 1 that instances 1–6 consist of 30 jobs each, to be scheduled on three and six machines respectively. The larger instances 7–12 consist of 50 jobs and five or eight machines. These instances contain about three times the operations of the smaller ones. The total number of operations ops is approximately $n \times m$. A smaller value of ops (e.g. instance 1) indicates that not every job passes every machine, while a larger value of ops (e.g. instance 3) indicates that some jobs pass a machine at least twice. Additional statistics are presented to demonstrate the heterogeneity and challenging nature of the benchmark set under consideration.

The time span between the release-date and the due-date of a job is called its allowance. Let P_i denote the total processing time of job i, i.e. the sum of its operation processing times. The allowance can be given in percent of the total processing time

Table 1
A set of heterogeneous scheduling instances

No.	n	m	ops	A	U	σ_u	σ_w
1	30	3	67	1.4	0.7	0.1	0.5
2	30	3	60	1.5	0.9	0.4	0.6
3	30	3	126	1.1	0.7	0.2	0.5
4	30	6	216	1.6	1.0	0.3	0.5
5	30	6	292	0.8	0.2	0.1	0.5
6	30	6	196	1.0	0.4	0.2	0.5
7	50	5	193	1.5	0.8	0.2	0.3
8	50	5	466	1.0	0.5	0.1	0.3
9	50	5	211	1.5	1.0	0.3	0.5
10	50	8	250	1.5	0.7	0.1	0.6
11	50	8	318	0.0	0.4	0.2	0.3
12	50	8	230	1.5	0.9	0.3	0.4

$$A_i = \frac{d_i - r_i}{P_i}, \qquad (1)$$

expressing the tightness of a job's due-date. Obviously, if $A_i < 1$ a job will always be tardy. However, completing a job on time is not very likely, even if A_i is close to 1. An allowance up to 1.5 indicates a rather tight due-date. Considering due-dates alone does not sufficiently reflect a tardiness problem. Additionally, the workload influences the problem difficulty. If many jobs await processing at a certain point in time, the consecutive scheduling decisions are not independent of each other. The workload is caused by two independent forces, the arrival process of jobs and the scheduling process itself. The former force is described by the mean inter-arrival time of jobs $\lambda = (r_n - r_1)/(n-1)$, where r_1 and r_n denote the earliest and the latest release-date respectively. Due to the a priori unknown quality of scheduling, the machine utilization rate is used to approximate a certain workload situation. Let $\overline{P}$ denote the average total processing time of all jobs, i.e. $\overline{P} = \sum_i P_i/n$. Since atmost m machines are busy in parallel, the utilization rate is given by

$$U = \frac{\overline{P}}{m\lambda}. \qquad (2)$$

A value $U > 1$ leads to a continuous increase of the workload over time. For a utilization ranging from 0.7 to 1.0, in general challenging problems can be expected.

Also the distribution of operation processing times influences the difficulty of a problem. The operations do not occupy the machine capacities uniformly which is shown by the standard deviation σ_u of machine utilization. An above average utilized machine will probably turn out as a bottleneck during the scheduling process. The last column of the table refers to the deviation of job-weights σ_w indicating a different distribution of job priorities.

4. Probabilistic scheduling procedures

4.1. Priority rules

Scheduling systems typically rely on priority rules, which have therefore become the subject of intense study (Haupt, 1989). Each time a machine becomes idle, a job is selected among those awaiting processing. This decision is taken according to a priority rule. Among the large number of rules proposed, we concentrate on three rules which are known to reduce job flow-times and job tardiness effectively. The FCFS rule is considered for reasons of comparison.

FCFS. The first-come, first-served rule attempts to implement an unbiased conflict solver because it neglects properties of jobs and the state of machines.

SPT. The shortest processing time gives priority to that operation with the shortest imminent processing time. Jobs waiting in a queue may cause their dedicated successor machine to run idle. SPT alleviates this risk by reducing the length of the queue in the fastest possible way.

S/OPN. The slack of a job is defined as the time span left within its allowance, assuming that the remaining operations are performed without any delay. Since jobs may wait in front of each machine the rule "slack per number of operations remaining" gives priority to the job with the minimum ratio of slack and the number of remaining operations.

COVERT. The cost-over-time rule combines ideas of SPT and S/OPN. It prioritizes jobs according to the largest ratio of the expected job tardiness and the required operation processing time. In this way it retains SPT performance but seeks to respect the due-dates if jobs are late (Russell et al., 1987).

Since the problems considered in this research contain job weights, the priorities are calculated on a weighted basis as described by Vepsalaine and Morton (1987).

A schedule builder produces a single schedule on the basis of a certain priority rule. A more balanced assessment of a rule results from applying it in a probabilistic fashion (Baker, 1974). First, priorities are assigned to all schedulable operations with respect to a certain priority rule. Then, the operation to be dispatched is drawn probabilistically in proportion to its assigned priority. In this way several schedules can be produced from one rule by retaining its original character.

4.2. Computational investigation

The above described priority rules are applied probabilistically to every problem of the benchmark set 1000 times. In order to report aggregate measures, the best result obtained for a problem is set in relation to the outcome of the deterministic version of the FCFS rule. These values, averaged over the instances of the benchmark set, are shown in Table 2 with respect to the four objectives under consideration.

The results shown for FCFS verify that a probabilistic schedule builder can produce considerable improvements against a deterministic one. We further confirm that SPT is the dominating rule if the minimization of $\overline{F}$ is pursued.

Table 2
Average performance of probabilistic scheduling

Rule	$\overline{F}$	$\overline{T}$	T_n	T_{max}
FCFS	0.000	0.000	0.000	0.000
FCFS$_p$	0.046	0.099	0.097	0.135
S/OPN$_p$	0.023	0.170	0.013	0.327
SPT$_p$	0.083	0.198	0.205	0.075
COVERT$_p$	0.051	0.217	0.129	0.269

The relative improvements achieved against deterministic FCFS are shown.

This rule also works well for reducing T_n. However, for the other criteria SPT performs rather weak, because it tends to badly delay those jobs which require comparably large processing times. S/OPN works favorable for minimizing T_{max}. Here SPT fails completely and is even outperformed by FCFS. In order to minimize $\overline{T}$ the COVERT rule is most suitable. In summary we state that a general purpose rule is not available.

5. Adaptation of priority assignment

Literature reports many GA approaches to scheduling (Ponnambalam et al., 2001). Since job shop scheduling is actually a constrained multi-sequencing problem, a lot of research concentrates on the problem representation by encoding operation sequences for the machines (Cheng et al., 1999).

In this paper we concentrate on the decision process involved in schedule building. A decision has to be made whenever multiple jobs compete for being processed on one machine. In Section 3.2 such conflicts have been solved by means of priority rules. In the following we first sketch a way of encoding decisions in a GA before going on to a brief description of the genetic operators. Then the resulting algorithm is applied to the benchmark set and its performance is compared with the outcome of probabilistic schedule building.

5.1. The genetic algorithm

5.1.1. Schedule encoding and decoding

To build a schedule, all conflicts among competing operations have to be solved. We encode these decisions by defining priorities between any two operations involved in a problem. These priorities are stored in a permutation consisting of all operations. Whenever two or more operations compete for a machine, the one is given priority which occurs leftmost in the permutation. In this way a permutation is used like a complex priority rule.

In priority-rule based scheduling, the set of conflicting operations is chosen such that no machine is kept idle when it could start processing some operation. The resulting schedules are referred to as non-delay schedules. Unfortunately, there are not necessarily optimal schedules (regarding $\overline{F}$, $\overline{T}$, $T_{\max}$ and T_{n}) in the set of non-delay schedules, because the objectives are regular measures of schedule performance. But optimal schedules regarding our objectives can always be found in the set of active schedules. In such schedules no operation can be started earlier without delaying some other operation or violating the machine routings. Active schedules are achieved by a slightly modified construction of the set of conflicting operations in the schedule builder (Giffler and Thompson, 1960). Here machines are allowed to remain idle for a while which leads to a strong increase of the search space.

The decoding of an active schedule is illustrated in Fig. 1. Assume that three operations *a*, *b* and *c* have to be processed by the same machine and let permutation (a, b, c) control the scheduling process. In the left diagram the operations are depicted at their earliest possible starting time. At time t_1 operations *b* and *c* form the conflicting set shown in grey shade. Operation *a* does not belong to this set, because *b* can be completed without delaying *a*. Here *b* is given preference with respect to the permutation.

At time t_2 operations *a* and *c* form the conflicting set. Now *a* is given preference which leads to a short idle-time of the machine. In case of non-delay scheduling idle-times are omitted and therefore merely *c* could have been scheduled regardless of the permutation ordering. Finally, at t_3 operation *c* starts processing, thus completing the machine schedule.

5.1.2. Crossover

Since sequencing as well as assignment problems allow a permutation encoding, various permutation crossover operators have been developed (Whitley, 2000). For vehicle routing, the preservation of precedence relations is assumed favorable (Blanton and Wainwright, 1993). Bierwirth et al. (1996) have incorporated this idea for scheduling problems resulting in the precedence preservative crossover operator (PPX).

A binary vector of equal length as the permutation is filled at random (Fig. 2). This vector defines the order in which the operations are successively drawn from parent 1 and parent 2. We now consider the parent and offspring permutations and the binary vector as lists. We start by initializing an empty offspring. Then the leftmost operation in one of the two parents is selected in accordance to the leftmost entry in the binary vector. After an operation is selected it is deleted in both parents and appended to the offspring. Finally the leftmost entry of the binary vector is also deleted. This procedure is repeated until the parent lists are empty and the offspring list contains all operations involved. Note that the PPX operator passes on priorities among operations given in two permutations to one offspring at the same rate, while no new priorities are introduced.

5.1.3. Mutation

We alter a permutation by first picking (and deleting) an operation before reinserting this operation at a randomly chosen position of the permutation. At the extreme, the priority relation of one operation to all other operations is affected, but typically a mutation has a much smaller effect.

5.2. Computational investigation

In order to assess the solution quality obtained, the above described GA is compared to probabilistic scheduling. The schedule builder is run in two

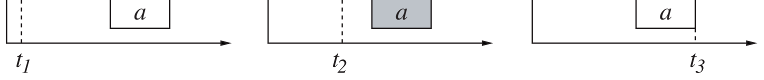

Fig. 1. Active schedule building controlled by permutation (a, b, c).

parent permutation 1	a	b	c	d	e	f
parent permutation 2	c	a	b	f	d	e
select parent no. (1/2)	1	2	1	1	2	2
offspring permutation	a	c	b	d	f	e

Fig. 2. PPX for a problem consisting of operations $a, b, \ldots, f$.

variants, producing either non-delay or active schedules. For both variants, the GA is parameterized identically. Fitness proportional selection is used, crossover is applied with probability 0.6, and mutation is applied with probability 0.01. A fixed population size of 100 individuals is used. This standard setting has been verified as producing satisfying results (Bierwirth and Mattfeld, 1999).

After a schedule has been decoded, the used permutation is rewritten with the sequence of operation starting times projected. In the example of Fig. 1 permutation (a, b, c) is rewritten with (b, a, c). In this way the redundancy of encoding schedules is reduced which leads to a faster convergence. In order to take advantage of a fast convergence, a flexible termination criterion is used. The algorithm terminates after T generations have been carried out without gaining further improvements. We confine T to one-half of the number of operations contained in a problem. In this way the algorithm is given a reasonably longer time to converge for larger problems.

The computational results of 50 runs each are aggregated as already described in Section 3.2. Table 3 shows the mean quality, deviation, and runtime of these computations (measured on a Pentium/200 MHz). The mean quality is shown together with the best results obtained by the best performing rule.

Both GAs outperform the most suitable priority rule for all objectives with the only exception of $T_{\max}$. Improvements range up to 7.5% for $\overline{T}$ and to 9% for T_{n}. However, the improvement for $\overline{F}$ is not remarkable, since already SPT performs excellently for this objective. Regarding $T_{\max}$ neither the non-delay GA nor the active GA is able to produce a mean schedule quality comparably as good as S/OPN.

Comparing both GA variants, the active GA performs superior for three criteria. For minimizing $\overline{F}$ the non-delay GA is clearly advantageous. This finding is surprising because non-delay schedules form a true subset of the active schedules. There may exist active schedules of better quality than the best non-delay schedules. Although within the scope of search, solutions found by the active GA for $\overline{F}$ are worse than those found by the non-delay GA. Obviously, the GA fails to explore the larger space of active schedules.

Table 3
Performance of probabilistic scheduling vs. GA performances using different schedule builders

	$\overline{F}$	$\overline{T}$	T_{n}	$T_{\max}$
Schedule quality				
Best probabilistic rule	0.083	0.217	0.205	0.327
Mean GA (non-delay)	0.095	0.279	0.259	0.314
Mean GA (active)	0.089	0.294	0.295	0.322
Standard deviation				
Best probabilistic rule	0.02	0.07	0.06	0.11
Mean GA (non-delay)	0.01	0.02	0.02	0.03
Mean GA (active)	0.02	0.04	0.04	0.05
Average runtime (seconds)				
GA (non-delay)	14	15	7	9
GA (active)	30	30	9	14

The standard deviation of the GA results is encouragingly small if compared with probabilistic scheduling. A small deviation is desired because a GA, unlike a priority rule, will be applied only once under realistic conditions. We take this handicap into account by reporting the mean schedule quality achieved. The observed standard deviation consistently duplicates when switching from non-delay to active schedule building.

The active GA is more time consuming on average than its non-delay counterpart. This is explained by the flexible termination criterion used, and the larger space explored by an active schedule builder. The increase of computation time corresponds to the increase of the standard deviation observed. The computation time of probabilistic scheduling is of course much shorter. It solely depends on the number of iterations carried out.

By increasing this number we can approximate arbitrarily large GA computation times, but we will hardly be likely to produce any further significant improvements.

In comparison with priority-rule based scheduling it can be doubted whether the GA results justify the effort spent so far. Improving the GA capabilities while keeping the flexibility and broad applicability of the algorithm is the subject of the reminder of this article.

6. Tunable schedule building

The common way to improve GA performance focuses on tuning its parameters, i.e. to enlarge the population size, or to alter the selection pressure. If all the parameters are already within useful bounds, the expected improvement usually does not justify the costs of finding an even more appropriate fine tuning (Thierens, 1999). More substantial improvements require an adjustment of the scope of search.

6.1. Non-delay versus active scheduling

It has already been established that neither non-delay scheduling nor active scheduling performs best. Therefore Della Croce et al. (1995) extend the search space of non-delay schedules towards active ones by using a look-ahead term. Similarly Norman and Bean (1997) introduce a delay factor, which is adapted together with the schedule. These ideas of flexible decoding motivate to vary the scope of search systematically by means of a tunable schedule builder as proposed by Storer et al. (1992).

In order to control the schedule builder, a look-ahead parameter $\delta \in [0, 1]$ is introduced which defines a bound on the time span a machine is allowed to remain idle. At the extremes $\delta = 0$ produces a non-delay schedule, while $\delta = 1$ produces an active schedule. The critical set of operations refers to those which can start at time $t \in [t', t' + \delta(c' - t')]$ as illustrated in Fig. 3. Here t' and c' denote the earliest possible starting time and the earliest possible completion time of an operation on machine M'.

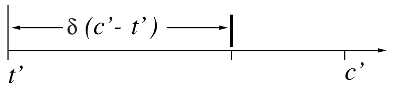

Fig. 3. Look-ahead interval with $\delta = 2/3$.

6.2. Computational investigation

In order to evaluate the tunable schedule builder the parameter δ is varied in [0, 1] by steps of 0.2. The rest of the experimental setting is used as described in Section 4.2. The corresponding results are shown in Table 4. Notice that the first and the last line (non-delay schedules with $\delta = 0.0$ and active schedules with $\delta = 1.0$) already appear in Table 3.

The most important result is that much better solutions, compared to those obtained from non-delay and active scheduling, are found in every case. Regardless of the particular objective pursued, a value of $\delta = 0.6$ yields the best mean performance. Even the maximal tardiness criterion where the S/OPN rule has produced the best result thus far ($T_{\max} = 3.27$), is improved clearly by 4.5%. The SPT rule, so powerful concerning the mean flow-time criterion ($\overline{F} = 0.83$), is outperformed by at least 3%.

With respect to the heterogeneity of the benchmark set, it can be conjectured that for individual instances values different from $\delta = 0.6$ work best. To investigate this issue we take a closer look at instances 7, 10 and 12 while varying δ for the mean tardiness criterion. In order to achieve comparability for these three problems, the measure is normalized. To this end the worst average performance is assigned a normalized measure of 0.0 and the best average performance is assigned a normalized measure of 1.0. The further results

Table 4
GA performance using a tunable schedule builder

δ	$\overline{F}$	$\overline{T}$	T_n	$T_{\max}$
0.0	0.095	0.279	0.259	0.314
0.2	0.104	0.319	0.276	0.345
0.4	0.111	0.335	0.291	0.356
0.6	0.113	0.349	0.314	0.372
0.8	0.106	0.334	0.308	0.353
1.0	0.089	0.294	0.295	0.322

obtained by varying δ are linearly scaled between these extrema (Fig. 4).

Starting from non-delay scheduling with $\delta = 0$, increasing the look-ahead improves the quality for all three problems. Previously excluded solutions are now included into the scope of the search while the GA still works sufficiently. Beyond a certain value of δ, a further increase of the search space leads to a deterioration of the solution quality.

The above finding surprises because increasing δ potentially introduces solutions of better quality into the search space. However, increasing δ incorporates a few solutions of superior quality at the expense of including a vast majority of solutions of inferior quality into the scope of search. At δ close to 1 the GA neither finds newly included solutions of superior quality nor it is able to identify solutions which have already been found before.

We find different look-ahead settings to be most suitable for the three problems under consideration. These critical values, where the capabilities of the GA are exhausted, give a hint on the specific difficulty of a problem. Unfortunately, it is unknown in advance.

The parameterization of the schedule builder also influences the computation time of the GA. Using $\delta = 1$ yields the largest search space which requires the longest computation time because of the dynamic termination criterion used. Accordingly, $\delta = 0$ leads to a significantly shorter computation time (Table 3).

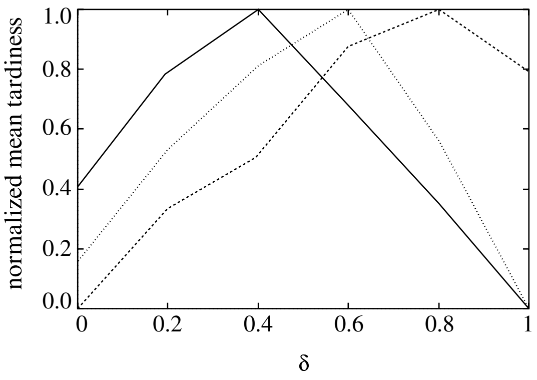

Fig. 4. GA performance, depending on δ, for instances 7, 10 and 12 and $\overline{T}$.

7. Multi-stage decomposition

The improvements presented so far are achieved by short-term planning at the level of individual machines. To further reduce the search space we propose to focus on the long-term planning at the shop-floor level. For this purpose we divide the jobs into subsets by the temporal order of their release-dates. In the first stage the GA is applied to the first subset of jobs. Those operations are fixed which are scheduled to start before the release of jobs in the second subset. The backlog of operations not fixed in the first stage is rescheduled together with the second subset of jobs in the second stage. This process of splitting a large problem into smaller problems is referred to as a multi-stage decomposition (Burke and Newall, 1999).

7.1. Generating sub-problems

In order to achieve a multi-stage decomposition for the scheduling problems at hand the jobs are sorted according to their release-dates (Raman and Talbot, 1993). The temporal order of jobs is divided into ε subsets (degree of decomposition) such that all subsets contain approximately the same number of jobs. In the first stage, the sub-problem comprises the jobs of the first subset. Subsequent sub-problems comprise the jobs of the corresponding sub-problem plus the backlog of the preceding sub-problem. The procedure of generating sub-problems in the multi-stage decomposition is shown below in detail. Here, the variable t denotes the earliest release-date of jobs in their respective subset.

(1) Remove all operations started before t from being considered.
(2) Remove a job if all of its operations have been removed already.
(3) Adjust the release-dates of jobs begun but not completed.
(4) Adjust the earliest time of availability for machines busy at t.
(5) Add all jobs released in the current period to the problem data.

The decomposition degree ε affects the size of the search space in the consecutive sub-problems. Obviously, the larger ε is, the smaller the search space gets. Using a very large decomposition degree leads to tiny sub-problems which can be solved effectively. At the same time this large decomposition degree reduces the planning horizon which can deteriorate the overall schedule quality again. Of course, even if all sub-problems are solved to optimality there is no guarantee that the resulting overall schedule is optimal.

7.2. Computational investigation

The benchmark set presented in Table 1 is stressed again. We vary the degree of decomposition by $\varepsilon \in \{1, 2, 3, 6\}$ and parameterize the schedule builder by $\delta \in \{0.0, 0.2, \ldots, 1.0\}$. The GA is run for 50 times on the 48 problems, resulting in a total of 57,600 observations (Table 5).

No significant impact of the parameter ε on the results imposed by δ (already observed in Table 4) can be recognized. Therefore we perform an analysis of variances (ANOVA). Here δ and ε are considered as independent variables and the achieved solution quality as the dependent variable. The analysis is performed for each of the four objectives separately, yielding almost identical results.

(1) As one would expect, there is a strong statistical significance concerning δ, i.e. varying this parameter affects the solution quality systematically.
(2) There is no statistical significance concerning ε on the basis of $\varepsilon \in \{1, 2, 3\}$. A small decomposition degree comes along with almost the same level of solution quality. In this range the effects of modifying ε (increasing the problem size and the potentials for planning at the same time, or vice versa) are in a balance.
(3) By incorporating $\varepsilon = 6$ we observe a weak significance, indicating that further decomposition leads to deterioration of the solution quality. The negative effect concerning reduced potentials for planning start dominating the positive effect of a smaller problem size.
(4) Since ε has shown hardly any significance, it does not surprise that the interaction of δ and ε shows no statistical significance at all. Negative affects of changing one parameter cannot be substituted by changes to the other parameter, at least within the investigated parameter range.

Table 5
GA performance using multistage decomposition

Criterion	ε	δ					
		0.0	0.2	0.4	0.6	0.8	1.0
$\overline{F}$	1	0.095	0.104	0.111	0.113	0.106	0.089
	2	0.096	0.106	0.113	0.115	0.109	0.093
	3	0.097	0.107	0.114	0.115	0.111	0.097
	6	0.094	0.104	0.112	0.115	0.113	0.101
$\overline{T}$	1	0.279	0.319	0.335	0.349	0.334	0.294
	2	0.285	0.320	0.333	0.348	0.326	0.288
	3	0.284	0.321	0.333	0.345	0.329	0.292
	6	0.284	0.319	0.331	0.340	0.331	0.295
T_n	1	0.259	0.276	0.291	0.314	0.308	0.295
	2	0.267	0.283	0.298	0.319	0.318	0.292
	3	0.266	0.280	0.293	0.313	0.313	0.290
	6	0.265	0.278	0.288	0.300	0.310	0.292
$T_{\max}$	1	0.314	0.345	0.356	0.372	0.353	0.322
	2	0.315	0.352	0.370	0.382	0.366	0.335
	3	0.318	0.352	0.365	0.384	0.366	0.340
	6	0.324	0.348	0.364	0.381	0.373	0.345

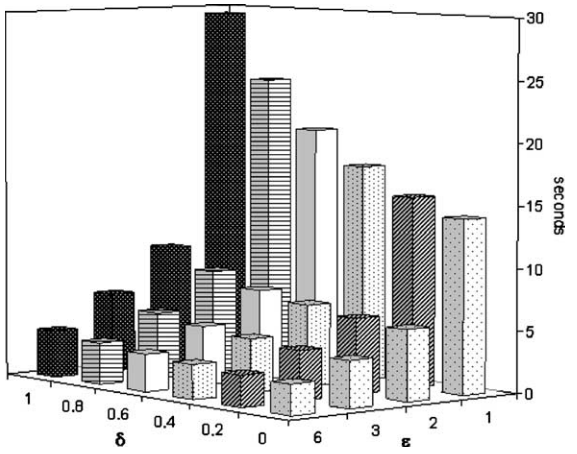

Fig. 5. Average GA computation time (seconds) needed to solve a single problem on average. Results are shown exemplary for the minimization of $\overline{T}$.

Although no significant improvement of the solution quality can be achieved, increasing ε strongly decreases the computation time needed. This trend observed for $\overline{T}$ is shown in Fig. 5 and can be confirmed for the other three objectives. The plot shows the average computation time of the GA regarding all combinations of ε and δ. The computation takes about 30 seconds for ε and δ both set to one (compare Table 3). A decrease of δ as well as an increase of ε shorten the computation time to a fraction of the original amount. Cutting a problem into two sub-problems already effects a gain larger than the one obtained by changing from an active schedule builder to a non-delay one.

An argument frequently raised against GAs is the one of comparably high computation times. Hence, it is convenient that a multi-stage decomposition limits the computational effort without deteriorating the scheduling quality significantly.

8. Comparison with other approaches

In two recent papers 24 problems of the benchmark set have been investigated (based on instances 1, 2, 7, 9, 10, 12, see Table 1). In order to compare these approaches with the GA proposed in this paper, it remains open to determine a suitable setting of δ and ε. First, we determine a promising subset of parameter combinations to limit the computational effort. Then, we compare the results of all three approaches with respect to the number of schedule evaluations performed.

8.1. Experimental setup

Since δ is an internal parameter of the GA a self-adaptive tuning is conceivable (Norman and Bean, 1997). However, co-evolving the parameter δ along with machine schedules is a difficult task in itself (John, 2002). The set of active schedules is much larger than the set of non-delay schedules, while the average performance of active schedules is inferior compared to non-delay ones. Therefore, the schedule builder will return a much better fitness in the vast majority of cases if it is called with a small δ (Mattfeld, 1999). As a consequence, selection tends to favor non-delay schedules because of their seemingly better potentials.

Fortunately, δ has turned out robust. Settings close to the extrema 0.0 and 1.0 have led to inferior results in all cases and are therefore excluded from being considered. Nevertheless, δ cannot be set suitably in advance. Therefore we consider $\delta \in \{0.2, 0.4, 0.6, 0.8\}$ in the experiment.

The degree of multi-stage decomposition is an external GA parameter. It is known from ANOVA that the setting of ε does not influence the solution quality. Therefore we confine the study to $\varepsilon = 3$ which yields reasonable runtime savings.

To achieve comparability between different algorithms, one can either measure computation times or count the number of schedule evaluations performed. We assess the latter method to be fair, because all investigated algorithms incorporate similar schedule builders.

Unfortunately, the computation effort spent on evaluating schedules in our approach cannot be determined in advance. This is because a flexible termination criterion and a multi-stage decomposition is used. In a decomposed problem ($\varepsilon > 1$) the size of the sub-problems differ. Since only a subset of the operations of the original problem is involved, we count the number of actually scheduled operations. In this way operations are counted twice (or even more often) if they are involved in two (or more) consecutive sub-problems. This leads to a generation comparative value

$$\text{gcv} = \frac{1}{\text{ops}} \sum_{i=1}^{\varepsilon} \text{gen}_i \cdot \text{ops}_i. \qquad (3)$$

As introduced above, ops represents the number of operations involved in a problem. In the ith sub-problem consisting of ops_i operations, gen_i denotes the number of generations carried out with respect to the flexible termination criterion. Table 6 reports the gcv values observed while parameterizing the GA with the above specified δ, ε setting.

In order to determine the total number of schedule evaluations carried out for each single problem, we multiply the sum of gcv's (for all δ) with the population size (100 individuals) and the number of repetitive trials (50 runs). This product is shown in the rightmost column of Table 6. The largest computational effort is spent on instance 12 in combination with $\overline{T}$, leading to $(128.3+126.0+143.2+162.1)\times 100\times 50\approx 2.8$ million evaluations.

8.2. Computational investigation

For all approaches considered the best solution quality produced for each problem is compared with respect to the computational effort spent. Therefore first the experimental setup of the competing approaches are briefly described below.

FCR. Fang et al. (1996) propose two different adaptive approaches which are compared with eight priority rules. The first adaptive method is a stochastic hill-climbing algorithm which works similarly to an Evolutionary Strategy with one parent and one offspring. As a second method, a GA previously described in Fang

Table 6
Generation comparative values gcv and total number of evaluations performed

Criterion	Instance no.	δ				Million eval.
		0.2	0.4	0.6	0.8	
$\overline{F}$	1	5.7	6.9	8.1	8.4	0.1
	2	8.0	8.3	7.9	9.1	0.2
	7	29.6	32.1	37.9	43.1	0.7
	9	89.2	88.3	94.4	99.2	1.9
	10	87.9	92.7	98.6	111.7	2.0
	12	119.4	121.5	137.9	150.6	2.6
$\overline{T}$	1	6.4	6.3	6.6	8.2	0.1
	2	7.9	8.3	8.6	9.4	0.2
	7	32.7	36.1	41.6	48.6	0.8
	9	93.1	93.8	106.8	109.8	2.0
	10	90.0	99.0	100.6	121.8	2.1
	12	128.3	126.0	143.2	162.1	2.8
T_n	1	5.7	5.6	5.9	6.2	0.1
	2	7.0	7.2	7.1	7.5	0.1
	7	20.5	22.8	22.7	25.2	0.5
	9	45.5	45.2	47.3	52.3	1.0
	10	39.9	35.6	35.8	42.3	0.8
	12	63.7	62.0	64.8	70.2	1.3
$T_{\max}$	1	5.7	5.8	5.9	6.5	0.1
	2	7.8	8.5	9.1	9.8	0.2
	7	21.7	27.4	31.7	37.4	0.6
	9	63.8	68.7	79.6	81.3	1.5
	10	49.4	61.5	61.1	70.0	1.2
	12	87.0	92.5	92.6	95.9	1.8

et al. (1993) is used. To provide a universally applicable algorithm, tailored operators are omitted, although an active schedule builder has been used for both algorithms. Both methods, as well as the eight rules are given a total of one million schedule evaluations each. The best schedule obtained from these 10 million evaluations is reported.

LPG. Lin et al. (1997) present a sophisticated GA. This algorithm takes advantage of a direct problem representation which encodes the operation starting times, and a niching population model. Next to an active schedule builder, a crossover operator tailored to combine two active schedules efficiently has been developed. A population of 500 individuals is run for 2500 generations, thus 1.125 million evaluations are carried in one run. The authors present the best result obtained from 10 runs (12.5 million evaluations).

As opposed to the above approaches, we keep the number of evaluations per GA run small while each run is repeated using different parameter settings. By comparing the figures given in the rightmost column of Table 6 with the evaluation number reported above for FCR and LPG it turns out that these approaches require a multiple of the schedule evaluations needed by our approach. We use atmost 22–28% of the computation time needed by the competing approaches (instance 12, $\overline{T}$).

For a qualitative discussion of results we distinguish between "small" and "large" problems. This is adequate, because it can be taken from Table 1 that instances 1 and 2 consist of less than one-third of the operations of the other four instances 7, 9, 10 and 12. Similarly, we consider two groups of objective functions, one consisting of the two mean-criteria $\overline{F}$ and $\overline{T}$ and the other one consisting of T_n and T_{max}. It can be taken from Table 6 that the GA produces twice the number of

Table 7
Performance of different algorithms for 24 problems

Criterion	Instance no.	RULE	FCR	LGP	MB
$\overline{F}$	1	0.029	0.099	0.099	0.99
	2	0.156	0.182	0.182	0.182
	7	0.088	0.075	0.120	0.130
	9	0.179	0.166	0.229	0.247
	10	0.095	0.084	0.107	0.135
	12	0.117	0.109	0.145	0.172
$\overline{T}$	1	0.089	0.380	0.388	0.379
	2	0.442	0.475	0.491	0.469
	7	0.451	0.404	0.670	0.687
	9	0.334	0.272	0.486	0.489
	10	0.308	0.352	0.374	0.425
	12	0.393	0.361	0.473	0.558
T_n	1	0.167	0.334	0.428	0.385
	2	0.451	0.550	0.551	0.523
	7	0.334	0.429	0.627	0.618
	9	0.413	0.591	0.688	0.725
	10	0.154	0.300	0.319	0.313
	12	0.358	0.609	0.615	0.628
T_{max}	1	0.383	0.596	0.585	0.596
	2	0.591	0.666	0.666	0.662
	7	0.478	0.500	0.646	0.732
	9	0.383	0.455	0.574	0.523
	10	0.442	0.455	0.526	0.532
	12	0.660	0.656	0.685	0.705

The best found schedules are given in terms of the improvement-rate against deterministic FCFS scheduling.

improvements for the mean-criteria indicating the particular difficulty of these criterions.

Table 7 reports the best solutions found by FCR, by LGP and finally by the approach described in this paper (MB). Additionally we list the best outcome of probabilistic scheduling (RULE).

Regarding T_n and T_{max} FCR outperforms the best performing rule for every instance. For the mean-criteria only the small problems can be improved. Here the improvements are due to the stochastic hill-climbing algorithm, the GA already fails for the small instances. For large instances probabilistic scheduling outperforms FCR in every case. This once again indicates the particular difficulty of the mean-criteria.

LGP solve small instances at almost the same level of solution quality which has been achieved already by FCR. Thus, it can be supposed that the instances 1 and 2 are solved to near-optimality by all approaches under consideration. The large instances are improved significantly by LGP in every case regardless of the objective pursued. Using advanced evolutionary techniques appears worth the higher computation times spent.

MB verify that progress is still possible. The mean-criteria are improved significantly for all large instances. For the more easy to solve objectives T_n and T_{max} only three out of four large instances are gradually improved, indicating that already LGP have solved these problems satisfactorily. Taking into account that the number of evaluations performed by LGP are at least four times larger compared to MB, it is remarkable that MB can produce superior schedules at all.

9. Summary

In this paper we have dealt with important aspects of realistic scheduling scenarios and with related performance criteria. The heterogeneity of data, constraints and objectives have so far at least hindered the implementation of tailored algorithmic approaches. GAs offer the opportunity to incorporate priority rule-based scheduling resulting in efficient optimization techniques.

The capabilities of GAs to conduct a proper search vanish with an increasing problem size. This inherent weakness of this generic approach can be alleviated by a tunable schedule builder which reduces the size of the search space. The multi-stage decomposition reduces the search space even further. However, the positive effects on the performance prevail. Together, new best solutions have been found for 13 of 24 problems while the computational burden is cut significantly.

References

Adams, J., Balas, E., Zawack, D., 1988. The shifting bottleneck procedure for job shop scheduling. Management Science 34, 391–401.

Anderson, E.J., Glass, C.A., Potts, C.N., 1997. Machine scheduling. In: Aarts, E.H.L., Lenstra, J.K. (Eds.), Local Search Algorithms in Combinatorial Optimization. Wiley, pp. 361–414.

Baker, K.R., 1974. Introduction to Sequencing and Scheduling. Wiley, New York.

Bierwirth, C., Mattfeld, D.C., 1999. Production scheduling and rescheduling with genetic algorithms. Evolutionary Computation 7, 1–17.

Bierwirth, C., Mattfeld, D.C., Kopfer, H., 1996. On permutation representations for scheduling problems. In: Voigt, H.-M., Ebeling, W., Rechenberg, I., Schwefel, H.-P. (Eds.), Proceedings of Parallel Problem Solving from Nature IV. Springer, Berlin.

Blanton, J.L., Wainwright, R.L., 1993. Multiple vehicle routing with time and capacity constraints using genetic algorithms. In: Forrest, S. (Ed.), Proceedings of the 5th International Conference on Genetic Algorithms. Morgan Kaufmann, San Francisco, CA.

Błażewicz, J., Domschke, W., Pesch, E., 1996. The job shop scheduling problem: Conventional and new solution techniques. European Journal of Operational Research 93, 1–30.

Burke, E.K., Newall, J.P., 1999. A multi-stage evolutionary algorithm for the timetable problem. IEEE Transactions on Evolutionary Computation 3, 63–74.

Cheng, R., Gen, M., Tsujimura, Y., 1999. A tutorial survey of job-shop scheduling problems using genetic algorithms part II: Hybrid genetic search strategies. Computers and Industrial Engineering 36, 343–364.

Della Croce, F., Tadei, R., Volta, G., 1995. A genetic algorithm for the job shop problem. Computers and Operations Research 22, 15–24.

Fang, H.-L., Ross, P., Corne, D., 1993. A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems. In: Forrest, S. (Ed.), Proceedings on Genetic Algorithms. Morgan Kaufmann, San Francisco, CA.

Fang, H.-L., Corne, D., Ross, P., 1996. A genetic algorithm for job-shop problems with various schedule quality criteria. In: Fogarty, T. (Ed.), Proceedings of AISB Workshop. Springer, Berlin.

Giffler, B., Thompson, G., 1960. Algorithms for solving production scheduling problems. Operations Research 8, 487–503.

Haupt, R., 1989. A survey of priority rule-based scheduling. OR Spektrum 11, 3–16.

He, Z., Yang, T., Tiger, A., 1996. An exchange heuristic imbedded with simulated annealing for due-dates job-shop scheduling. European Journal of Operational Research 91.

James, R.J.W., Buchanan, J.T., 1998. Performance enhancements to tabu search for the early/tardy scheduling problem. European Journal of Operational Research 106, 254–265.

John, D.J., 2002. Co-evolution with the Bierwirth–Mattfeld hybrid scheduler. In: Proceedings of the GECCO 2002 Conference, New York.

Kreipl, S., 2000. A large step random walk for minimizing total weighted tardiness in a job shop. Journal of Scheduling 3, 125–138.

Lin, S.-C., Goodman, E.D., Punch, W.F., 1997. A genetic algorithm approach to dynamic job shop scheduling problems. In: Bäck, T. (Ed.), Proceedings of the 7th International Conference on Genetic Algorithm. Morgan Kaufmann, San Francisco, CA.

Mattfeld, D.C., 1999. Scalable search spaces for scheduling problems. In: Banzaf, W., Daida, J., Eiben, A.E., Garzon, M.H., Honavar, V., Jakiela, M., Smith, R.E. (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference. Morgan Kaufmann, San Francisco, CA.

Morton, T.E., Pentico, D.W., 1993. Heuristic Scheduling Systems. Wiley, New York.

Norman, B.A., Bean, J.C., 1997. A random keys genetic algorithm for job shop scheduling problems. Engineering Design and Automation Journal 3, 145–156.

Norman, B.A., Bean, J.C., 1999. A genetic algorithm methodology for complex scheduling problems. Naval Research Logistics 46, 199–211.

Ponnambalam, S.G., Aravindan, P., Rao, P.S., 2001. Comparative evaluation of genetic algorithms for job-shop scheduling. Production Planning and Control 12, 560–574.

Raman, N., Talbot, F.B., 1993. The job shop tardiness problem: A decomposition approach. European Journal of Operational Research 69, 187–199.

Russell, R.S., Dar-El, E.M., Taylor, B.W., 1987. A comparative analysis of the COVERT job sequencing rule using various shop performance measures. International Journal of Production Research 25, 1523–1540.

Storer, R., Wu, D., Vaccari, R., 1992. New search spaces for sequencing problems with application to job shop scheduling. Management Science 38, 1495–1509.

Thierens, D., 1999. Scalability problems of simple genetic algorithms. Evolutionary Computation 7, 331–352.

Valls, V., Perez, M.A., Quintanilla, M.S., 1998. A tabu search approach to machine scheduling. European Journal of Operational Research 106, 277–300.

Vepsalaine, A.P.J., Morton, T.E., 1987. Priority rules for job shop with weighted tardiness costs. Management Science 33.

Whitley, D., 2000. Permutations. In: Bäck, T., Fogel, D., Michalewicz, T. (Eds.), Evolutionary Computation 1. IOP Press, pp. 139–150.