Pergamon

0305-0548(93)E0016-M

Computers Ops Res. Vol. 22, No. 1, pp. 25–40, 1995
Copyright © 1994 Elsevier Science Ltd
Printed in Great Britain. All rights reserved
0305-0548/95 $7.00+0.00

EVOLUTION BASED LEARNING IN A JOB SHOP SCHEDULING ENVIRONMENT

ULRICH DORNDORF[1]† and ERWIN PESCH[2]‡

[1]INFORM, Institut für Operations Research und Management GmbH, Pascalstraße 23, D-52076, Germany and [2]Faculty of Economics and Business Administration, KE, University of Limburg, P.O. Box 616, NL-6200 MD Maastricht, The Netherlands

Scope and Purpose—Machine scheduling problems arise in diverse areas such as flexible manufacturing, production planning, computer design, logistics, communication, etc. A common feature of many of these problems is that no efficient solution algorithms are known that solve each instance to optimality in a time bounded polynomially in the size of the problem. Approximation algorithms can help to overcome these difficulties at the cost of a quality loss of the solutions. Local decision rules are mostly the backbone of such approximation algorithms, i.e. the latter try to find the best sequence of local decisions by deterministic or probabilistic learning. The outcome of such an approach heavily depends on the underlying meta-strategy. In this paper we provide a probabilistic learning strategy based on principles of evolution. We shall indicate how the method can be extended to consider local decision rules of any number and complexity in order that the resulting program will do better than the deterministic learning approach.

Abstract—A class of approximation algorithms is described for solving the minimum makespan problem of job shop scheduling. A common basis of these algorithms is the underlying genetic algorithm that serves as a meta-strategy to guide an optimal design of local decision rule sequences. We consider sequences of dispatching rules for job assignment as well as sequences of one machine solutions in the sense of the shifting bottleneck procedure of Adams *et al.* Computational experiments show that our algorithm can find shorter makespans than the shifting bottleneck heuristic or a simulated annealing approach with the same running time.

INTRODUCTION—THE JOB SHOP SCHEDULING PROBLEM

A job shop consists of a set of different machines (like lathes, milling machines, drills etc.) that perform operations on jobs. Each job has a specified processing order through the machines, i.e. a job is composed of an ordered list of operations each of which is determined by the machine required and the processing time on it. There are several constraints on jobs and machines:

(i) There are no precedence constraints among operations of different jobs.
(ii) Operations can not be interrupted (non-preemption) and each machine can handle only one job at a time.

Consequently, each job can be performed on only one machine at a time. The operation sequences on the machines are unknown and have to be determined in order to minimize the makespan, i.e. the time required to complete all jobs. It is well known that the problem is NP-hard [1] and belongs to the most intractable problems considered [2]. A huge amount of literature has been published within the last 30 years, among others the books by Muth and Thompson [3], Conway

†Ulrich Dorndorf is a scientific consultant at the Institute for Operations Resarch and Management in Aachen. He studied at the University of Technology in Darmstadt and at the University of Illinois at Urbana Champaign and graduated in Mechanical Engineering and Business Administration. His research interests are in the area of Decision Support Systems, the integration and application of OR and AI techniques in industrial practice, and Transportation and Production Planning and Scheduling. He has published in engineering and computer science journals, including *ORSA Journal on Computing.*

‡Erwin Pesch is an Assistant Professor at the Faculty of Economics at the University of Limburg in Maastricht. He graduated in Mathematics and Computer Science, obtained his Ph.D. in Mathematics from the University of Technology in Darmstadt, and worked for some time on the problem of software quality control in a banking environment. His research fields are in Artificial Intelligence, Decision Support Systems, Production Planning and Scheduling. He has published in many international journals, including *Discrete Mathematics, Journal of Graph Theory, Journal of Combinatorial Theory, Discrete Applied Mathematics, ORSA Journal on Computing* and *European Journal of Operational Research.*

et al. [4], Baker [5] and French [6] as well as Lenstra [7] and Rinnooy Kan [8]. A variety of scheduling rules for certain types of job shops have been considered in an enormous number of publications, see Blazewicz [9].

There are different problem formulations [10, 11] and we have adopted that one presented by Adams *et al.* [12]. Let $V=\{0, 1, \ldots, n\}$ denote the set of operations where 0 and n are considered as dummy operations "start" and "finish", respectively. Let M denote the set of machines; A is the set of pairs of operations constrained by the precedence relations for each job. For each machine k, the set E_k describes the set of all pairs of operations to be performed on machine k, i.e. operations which cannot overlap [cf. (ii)]. For each operation i, its processing time d_i is fixed, and the earliest possible process start of i is t_i, a variable that has to be determined during the optimization. Hence, the job shop scheduling problem can be modeled as

$$\min t_n$$

subject to

$$t_j - t_i \geqslant d_i \qquad (i, j) \in A, \tag{1}$$

$$t_j - t_i \geqslant d_i \text{ or } t_i - t_j \geqslant d_j \qquad \{i, j\} \in E_k, k \in M, \tag{2}$$

$$t_i \geqslant 0 \qquad i \in V. \tag{3}$$

Restrictions (1) ensure that the processing sequence of operations in each job corresponds to the predetermined order. Constraints (2) demand that there is only one job on each machine at a time, and (3) requires an earliest starting time for the jobs. Any feasible solution to the constraints (1), (2) and (3) is called a schedule.

An illuminating problem representation is the disjunctive graph model due to Roy and Sussmann [13]. In the disjunctive graph there is a vertex for each operation $i \in V$ and vertices 0 and n representing the start and the end, respectively, of a schedule. For every two consecutive operations of the same job there is a directed arc; the start vertex 0 is considered to be the first operation of every job and the end vertex n is considered to be the last operation of every job. For each pair of operations $\{i, j\} \in E_k$ that require the same machine there are two arcs (i, j) and (j, i) with opposite directions. Thus, single arcs between operations represent the precedence constraints on the operations and opposite directed arcs between two operations represent the fact that each machine can handle at most one operation at the same time. An arc (i, j) is labelled by some positive weight corresponding to the processing time d_i of operation i. All arcs from 0 have a label 0. Figure 1 from Balas [14] illustrates the disjunctive graph for a problem with 5 jobs and 15 operations on 4 machines. The job shop scheduling problem is to find an order of the operations on each machine, i.e. to select one arc among all opposite directed arc pairs such that the resulting graph is acyclic (i.e. there are no precedence conflicts between operations) and that the length of the maximum weight path between the start and end vertex is mimimal. The length of a maximum weight (or longest) path determines the makespan.

The paper is organized as follows. After a brief introduction to genetic algorithms we describe two types of learning. In each case a genetic algorithm serves as a meta-strategy to control the process of learning combinations of local job shop scheduling procedures. As local scheduling procedures we consider priority scheduling rules described in Panwalkar and Iskander [15] as well as the shifting bottleneck procedure from Adams *et al.* [12]. Finally we present computational results compared to the best heuristics known up to now for the job shop scheduling problem, such as simulated annealing and the shifting bottleneck procedure.

GENETIC ALGORITHMS

As the name suggests, genetic algorithms are motivated by the theory of evolution and date back to the early work of Rechenberg [16], Holland [17] and Schwefel [18], see also Goldberg [19, 20], Liepins and Hilliard [21] and Michalewicz [22]. They have been designed as general search strategies and optimization methods.

In contrast to local search algorithms such as simulated annealing [23] and tabu search [24, 25], which are based on manipulating one feasible solution, a genetic algorithm considers a population

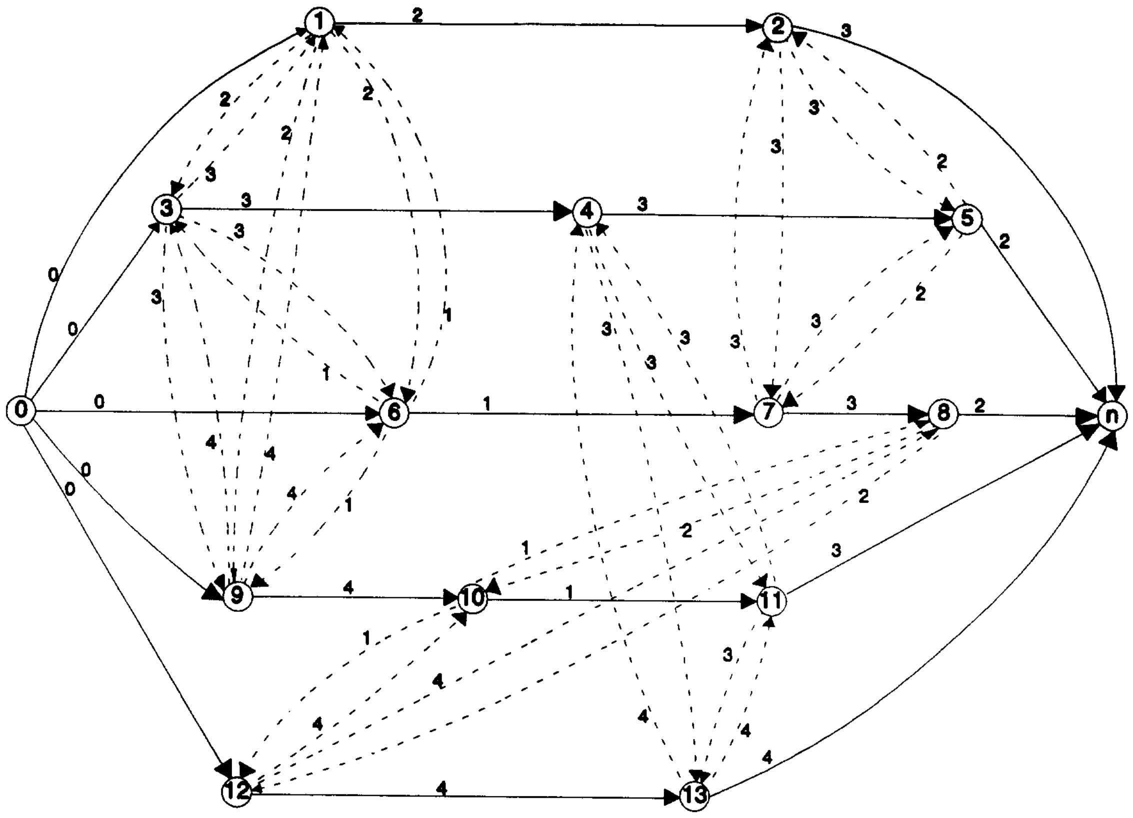

Fig. 1

of feasible solutions. Working with populations permits us to identify and explore properties which good solutions have in common.

Roughly speaking, a genetic algorithm aims at producing near-optimal solutions by letting a set of random solutions undergo a sequence of unary and binary transformations governed by a selection scheme biased towards high-quality solutions.

These transformations constitute the recombination step of a genetic algorithm and are performed on the population by three simple operators. The effect of operators is that implicitly good properties are identified and combined into a new population, which hopefully has the property that the best solution and the average value of the solutions are better than in previous populations. The process is then repeated until some stopping criteria are met. It can be shown that the process converges to an optimal solution with probability one (cf. Eiben *et al.* [26]). The three basic operators of a genetic algorithm are reproduction, crossover and mutation, defined below.

Problems from combinatorial optimization are well within the scope of genetic algorithms and early attempts closely followed the scheme of what Goldberg [19] calls a simple genetic algorithm. Compared to standard heuristics, for instance for the travelling salesman problem, "genetic algorithms are not well suited for fine-tuning structures which are very close to optimal solutions" [27]. Therefore it is essential if a competitive genetic algorithm is desired, to incorporate (local search) improvement operators, see Jog *et al.* [28] and Suh and Van Gucht [29]. The resulting algorithm has then been called genetic local search heuristic; in the case of the travelling salesman we refer to Ulder *et al.* [30], Johnson [31], and Kolen and Pesch [32]. Putting things into a more general framework, a solution of a combinatorial optimization problem may be considered as a sequence of local decisions. For instance, in case of the travelling salesman problem a local decision might result in a choice of a city visited next. A local decision for the job shop scheduling problem might be the choice of an operation to be scheduled next. In what follows we will consider more general decision rules. In an enumeration tree of all possible decision sequences a solution of the problem is represented as a path corresponding to the different decisions from the root of the tree to some leaf. A branch and bound algorithm learns to find those decisions leading to an optimal

```
**Initialization**: Construct an initial population of individuals each of which is a string of local decision rules.
**Assessment/Improvement**: Assess each individual in the current population introducing problem specific
    knowledge by special purpose heuristics (such as local search) which are guided by the sequence of local decisions.
    *if* special purpose heuristics lead to a new string of local decision rules *then*
        replace each individual by the new one, for instance a locally optimal one.
*repeat*
    **Recombination**: Extend the current population by adding individuals obtained by unary and binary
        transformations (crossover, mutation) on one or two individuals in the current population.
    **Assessment/Improvement**: Assess each individual in the current population introducing problem specific
        knowledge by special purpose heuristics (such as local search) which are guided by the sequence of local
        decisions.
        *if* special purpose heuristics lead to a new string of local decision rules *then* replace each individual by the
        new one, for instance a locally optimal one.
    **Selection**: Reduce the extended population to its original size according to the selection rules.
*until* some stopping criterion is met.
```

Fig. 2. Genetic enumeration.

solution with respect to the space of all decision sequences. Genetics can guide a search process in order to learn to find the most promising decisions within a reasonable amount of time. The scheme of a so called genetic enumeration algorithm is described in Fig. 2, it requires further refinements in order to design a successful algorithm.

During the recombination step for reproduction the fitness value of a solution has to be defined. Usually the fitness value is the value of the objective function or some scaled version of it when all local decision rules and the improvement step have been applied. Next a new population is constructed. Therefore a new temporary population is generated where each member is a replica of a member of the old population. A copy of an old string is produced with probability proportional to its fitness value, i.e. better strings probably get more copies. The generation of new solutions is handled by the crossover operator.

In order to apply the crossover operator the population is randomly partitioned into pairs. Next, for each pair, the crossover operator is applied with a certain probability by choosing a position randomly in the string and exchanging the tails (defined as the substring starting at the chosen position) of the two strings.

The mutation operator which makes random changes to single elements of the string only plays a secondary role in genetic algorithms. Mutation serves to maintain diversity in the population.

Besides the unary and binary recombination operator one may also introduce operators of higher arities such as consensus operators, that fix arc directions common to most schedules of a current population, cf. Aarts *et al.* [33] and Nakano and Yamada [34]. Selection can be realized in a number of ways: one could adopt the scenario of Goldberg [19] or use deterministic ranking. Further it matters whether the newly recombined offspring compete with the parent solutions or simply replace them.

The traditional genetic algorithm based on a binary string representation of a solution is often unsuitable for combinatorial optimization problems because it is very difficult to represent a solution such that substrings have a meaningful interpretation. Choosing a more natural representation of solutions, however, involves more intricate recombination operators, in particular crossover operators in order to get feasible offspring; for the job shop scheduling problem see Yamada and Nakano [35]; for the travelling salesman problem see Mühlenbein *et al.* [36, 37], Gorges-Schleuter [38], or Kolen and Pesch [32]. (The travelling salesman problem was also the driving force behind earlier approaches on genetic based scheduling, inter alia motivated by the close relationship of the travelling salesman and the flow shop problem, see Ablay [39] and Stöppler and Bierwirth [40].) To overcome these difficulties and apply the simple crossover operator one can use an interpretation of an individual solution as a sequence of decision rules. Each individual of a population is considered to be a subset of feasible schedules from the set of all feasible schedules, i.e. a solution can also be considered as a set of solutions of the problem instance.

HEURISTICS FOR THE JOB SHOP SCHEDULING PROBLEM

Priority rules are probably the most frequently applied heuristics for solving (job shop) scheduling problems in practice because of their ease of implementation and their low time complexity. The algorithm of Giffler and Thompson [41] can be considered as a common basis of all priority rule based heuristics. Let $Q(t)$ be the set of all unscheduled operations at time t. Let r_i and c_i denote the earliest possible start and the earliest possible completion time, respectively, of operation i. The algorithm of Giffler and Thompson assigns available operations to machines, i.e. operations which can start being processed. Conflicts, i.e. operations competing for the same machine, are solved randomly. A brief outline of the algorithm is given in Fig. 3.

The Giffler/Thompson algorithm can generate all active schedules and hence also all optimal schedules. As the conflict set consists only of operations, i.e. jobs, competing for the same machine, the random choice of an operation or job from the conflict set may be considered as the simplest version of a priority rule where the priority assigned to each operation or job in the conflict set corresponds to a certain probability. Many other priority rules can be considered, for instance the total processing time of all subsequent job operations of operation i may be a criterion. We apply only those rules presented in Table 1; for an extended summary and discussion see Panwalkar and Iskander [15], Haupt [42], as well as Blackstone *et al.* [43]. The first column of Table 1 contains an abbreviation and name of the rule while the last column describes which of the operations or jobs in the conflict set gets highest priority.

The **Shifting Bottleneck Heuristic** from Adams *et al.* [12] (and its recent improvements by Dauzere-Peres and Lasserre [44] and by Balas *et al.* [45]) is probably the most powerful procedure known up to now among all heuristics for the job shop scheduling problem. The idea is to solve for each machine a one machine scheduling problem to optimality under the assumption that a lot of arc directions in the optimal one machine schedules coincide with an optimal job shop schedule. Consider all operations of a job shop scheduling instance that have to be scheduled on machine m. In the (disjunctive) graph including a partial selection of opposite directed arcs (corresponding to a partial schedule) there exists a longest path of length h_i from dummy operation 0 to each operation i scheduled on machine m. Processing of operation i cannot start before time (also called head) h_i. There is also a longest path of length q_i (called tail) from i to the dummy operation n. Obviously, when i is finished it will take at least q_i time units to finish the whole schedule. The one machine scheduling problem with heads and tails is also NP-complete [46], however, there is a powerful branch and bound method proposed by Carlier [47] which dynamically changes heads and tails in order to improve the operations' sequence.

The shifting bottleneck heuristic consists of two subroutines. The first one (SB1) repeatedly solves one machines scheduling problems while the second one (SB2) builds a partial enumeration tree where each path from the root to a leaf is similar to an application of SB1. In order to understand the genetic approach, we have to consider the heuristic in more detail. As the very name suggests, the shifting bottleneck heuristic always schedules bottleneck machines first. As a measure of the bottleneck quality of machine m, the value of an optimal solution of a certain one machine scheduling problem on machine m is used. The one machine scheduling problems in consideration are those which arise from the disjunctive graph model when certain machines are already sequenced. The operation orders on sequenced machines are fully determined. Hence sequencing an additional machine probably results in a change of heads and tails of those operations whose machine order is still open. For all machines not sequenced, the maximum makespan of the corresponding optimal one machine schedules, where the arc directions of the already sequenced machines are fixed, determines the bottleneck machine. In order to minimize the makespan of the job shop scheduling

```
$t := 0$; $Q(t) := \{1, \ldots, n-1\}$;
*repeat*
    Among all unscheduled operations in $Q(t)$ let $j^*$ be that one with smallest completion time, i.e. $c_{j^*} = \min\{c_j \mid j \in Q(t)\}$.
    Let $m^*$ denote the machine $j^*$ has to be processed on.
    Randomly choose an operation $i$ from the conflict set $\{j \in Q(t) \mid j$ has to be processed on machine $m^*$ and $r_j < c_{j^*}\}$.
    $Q(t) := Q(t) \setminus \{i\}$; Modify $c_j$ for all operations $j \in Q(t)$. Set $t$ to the next possible operation to machine assignment.
*until* $Q(t)$ is empty.
```

Fig. 3. The algorithm of Giffler and Thompson.

Table 1. Priority rules

Rule	Description
1. SOT-rule (shortest operation time)	An operation with shortest processing time on the considered machine
2. LOT-rule (longest operation time)	An operation with longest processing time on the machine considered
3. LRPT-rule (longest remaining processing time)	An operation with longest remaining job processing time
4. SRPT-rule (shortest remaining processing time)	An operation with shortest remaining job processing time
5. LORPT-rule (longest operation remaining processing time)	An operation with highest sum of tail and operation processing time
6. Random	The operation for the considered machine is randomly chosen
7. FCFS-rule (first come first serve)	The first operation in the queue of jobs waiting for the same machine
8. SPT-rule (shortest processing time)	A job with smallest total processing time
9. LPT-rule (longest processing time)	A job with longest total processing time
10. LOS-rule (longest operation successor)	An operation with longest subsequent operation processing time
11. SNRO-rule (smallest number of remaining operations)	An operation with smallest number of subsequent job operations
12. LNRO-rule (largest number of remaining operations)	An operation with largest number of subsequent job operations

```
Let $M$ be the set of all machines and let $M' := \{\ \}$ be the set of all sequenced machines;
repeat
    for $m \in M \backslash M'$ do
    begin
        Compute head and tail for each operation $i$ that has to be scheduled on machine $m$. Solve the one machine
        scheduling problem to optimality for machine $m$; let $v(m)$ be the resulting makespan for this machine.
    end;
    Let $m^*$ be the bottleneck machine, i.e. $v(m^*) \geqslant v(m)$ for all $m \in M \backslash M'$.
    $M' := M' \cup \{m^*\}$;

    /* local reoptimization */
    for $m \in M'$ in the order of its inclusion do
    begin
        Delete all arcs between operations on $m$ while all arc directions between operations on machines from $M' \backslash \{m\}$
        are fixed.
        Compute heads and tails of all operations on machine $m$ and solve the one machine scheduling problem.
    end;
until $M = M'$.
```

Fig. 4. Outline of the SB1-heuristic.

problem the bottleneck machine should be sequenced first. A brief statement of the shifting bottleneck procedure is given in Fig. 4.

The one machine scheduling problems are solved using the algorithm of Carlier [47]. During the local reoptimization part of the SB1-heuristic, for each machine, the operation sequence is redetermined keeping the sequences of all other already scheduled machines untouched. As suggested by Adams *et al.* [12], we go through at most three local reoptimization iterations for each set M'. Only in the last step, when $M = M'$, we continue until there is no improvement for a full iteration. For details see Adams *et al.* [12].

The quality of the schedules obtained by the SB1-heuristic heavily depends on the sequence in which the one machine problems are solved and these machines included in the set M'. Sequence changes may yield substantial improvements. This is the idea behind the second version of the shifting bottleneck procedure, i.e. the SB2-heuristic, as well as behind our second genetic algorithm approach. The SB2-heuristic applies a slightly modified SB1-heuristic to the nodes of a partial enumeration tree. A node corresponds to a set M' of machines that have been sequenced in a particular way. The root of the search tree corresponds to $M' = \{\ \}$. A branch corresponds to the inclusion of a machine m into M', thus the branch leads to a node representing an extended set $M' \cup \{m\}$. At each node of the search tree a single step of the SB1-heuristic is applied, i.e. machine m is included possibly followed by a local reoptimization. Each node in the search tree corresponds to a particular sequence of inclusion of the machines into the set M'. Thus, the bottleneck criterion no longer determines the inclusion into M'. Obviously a complete enumeration of the search tree is not acceptable. Therefore a breadth-first search up to depth l is followed by a depth-first search. In the former case, for a search node corresponding to set M' all possible branches are considered which result from inclusion of a machine $m \notin M'$. Hence the successor nodes of node M' correspond to machine sets $M' \cup \{m\}$ for all $m \in M \backslash M'$. Beyond the depth l an extended bottleneck criterion is applied, i.e. instead of $|M \backslash M'|$ successor nodes there are several successor nodes generated corresponding to the inclusion of the bottleneck machine as well as several other machines m to M'. In our implementation of the SB2-heuristic we kept $l = 3$, and beyond this depth applied the bottleneck criterion, i.e. only one successor node corresponding to the inclusion of the bottleneck machine was generated.

LEARNING BY POPULATION GENETICS

In this section we demonstrate how special purpose heuristics can be incorporated into a genetic framework such that population genetics serve as a strategy to determine how high quality solutions can be constructed. We consider two strategies, one is based on priority rules while the other uses ideas of the SB2-heuristic.

Genetic algorithms do not fit best for scheduling problems in order to get near-optimal solutions if no improvement heuristic, such as local search, is incorporated (cf. Davis [48], Whitley *et al.* [49], Husbands *et al.* [50], Starkweather *et al.* [51] and Hilliard and Lipeins [52]). There are several possibilities to apply an improvement heuristic. For instance, as in the case of the travelling salesman problem, one can find an encoding such that each individual of a population in fact is a solution of the problem. Hence during the recombination phase an improvement step can be applied to all or several of the solutions in a population. Using 2-opt or the Lin–Kernighan heuristic for the travelling salesman problem provides a powerful genetic local search procedure, see Johnson [31], Ulder *et al.* [30], and Kolen and Pesch [32]. Some type of an improvement heuristic may also be incorporated into the crossover operator (see Kolen and Pesch [32]). In any case the improvement step as well as the crossover operator heavily depend on the representation of the solution. Usually a simple representation requires more sophisticated recombination operators and vice versa. To overcome these difficulties we tried a completely different encoding scheme for the job shop scheduling problem. Our solution representation enables us to use the simplest type of crossover as well as to incorporate problem specific knowledge, i.e. as Davis [48] claimed "to examine the workings of a good deterministic program in that domain", in order to be competitive with special purpose heuristics. The strategy controls a sequence of priority rules or the machine inclusion sequence for the SB1-heuristic and learns to find best combinations in both cases. Another genetic local search approach based on an arc solution representation is described in Aarts *et al.* [33] or in Nakano and Yamada [34]. Their ideas are stimulated by the encouraging results obtained for the travelling salesman problem (cf. Ulder *et al.* [30]). A different approach has been followed by Storer *et al.* [53]. They map the original data of the underlying problem instance to slightly disturbed and genetically controlled data representing new problem instances. The latter are solved heuristically and the solutions, i.e. the operations' processing orders, are considered to be solutions of the original problem.

Each individual of the **priority rule based genetic algorithm** (for short: **P-GA**) is a string of $n-1$ entries $(p_1, p_2, \ldots, p_{n-1})$ where $n-1$ is the number of operations in the underlying problem instance. An entry p_i represents one rule of the set of twelve priority rules described in Table 1. The entry in the ith position says that a conflict in the ith iteration of the Giffler and Thompson algorithm should be resolved using priority rule p_i. More precisely, an operation from the conflict set has to be selected by rule p_i; ties are broken by a random choice. Within a genetic framework a best sequence of priority rules has to be determined. The crossover operator is straightforward.

Obviously, the simple crossover, where the substrings of two cut strings are exchanged, applies and always yields feasible offspring. Heuristic information already occurs in the encoding scheme and a particular improvement step is dropped. From our experiments we recommend to use a set of priority rules which are partially complementary (e.g. SOT and LOT, SPT and LPT etc.) in order to be able (hopefully) to choose each member in a conflict set. Otherwise we could not prevent a premature convergence or exclusion of an optimal solution. The mutation operator applied with a very small probability simply switches a string position to another one, i.e. the priority rule of a randomly chosen string entry is replaced by a new rule randomly chosen among the remaining ones. We also tested an inversion operator (see Holland [17]) but the experiments did not show a clear advantage of using inversion. Besides the priority rules of Table 1 one can use other kinds of rules which fix the order of operations on the same machine. For instance, any permutation of unscheduled operations might represent a rule. However, this will result in string codes over a huge alphabet contrary to the observation that small alphabets usually accelerate a uniform convergence (see Goldberg [20]). We did some experiments in this direction where we restricted the number of rules to k out of $k!$ permutations if k is the number of operations that can be in conflict. The less encouraging results let us drop this approach.

Our approach to search a best sequence of decision rules for selecting operations is just in line with the ideas of Fisher and Thompson [54] on probabilistic learning of sequences consisting of two priorty rules, and Crowston *et al.* [55] or O'Grady and Harrison [56] on learning how to find promising linear combinations of basic priorities, see Glover [57].

In our first implementation the genetic algorithm serves as a meta-strategy to optimally control the use of priority rules, whereas the genetic algorithm controls the selection of nodes in the enumeration tree of the shifting bottleneck heuristic in our second implementation, the **shifting bottleneck based genetic algorithm** (for short: **SB-GA**). Remember that the SB2-heuristic is only a repeated application of a part of the SB1-heuristic where the sequence in which the one machine problems are solved is predetermined. Up to some depth l, a complete enumeration tree is generated and a partial tree for the remaining search levels. The SB2-heuristic tries to determine the best single machine sequence for the SB1-heuristic within a reasonable amount of time. We demonstrate that this can also be achieved by a genetic strategy, even in a more effective way. Contrary to Whitley *et al.* [49] scheduling of one machine alone does not in general provide a highly constrained solution space.

The length of a string representation of an individual in the population equals the number of machines in the problem which is equal to the depth of the enumeration tree in the SB2-heuristic. Hence, an individual is encoded over the alphabet from 1 to the number of machines and a partial string from the first to the kth entry just describes the sequence in which the single machines are considered in the SB1-heuristic. As a crossover operator we can use any travelling salesman crossover: we chose the cycle crossover as described in Goldberg [19]. Neither a mutation nor an investion operator is applied. In order to reduce the computation time we decreased the number of reoptimization cycles during the execution of the SB1-heuristic. Instead of immediately starting a reoptimization cycle when two machines are sequenced we initiated a reoptimization cycle when less than 6 machines are left over, i.e. not included in the partial solution. The value 6 has been empirically found to be best. The difference between the shifting bottleneck heuristic and our genetic approach is that the bottleneck is no longer a decision criterion for the choice of the next machine.

DETAILS OF IMPLEMENTATION AND COMPUTATIONAL RESULTS

Setting the control parameters for a genetic algorithm is an important decision. We chose the empirically best parameter configuration with respect to probably the 3 best known test problems for job shop scheduling: the 6×6, the 5×20, and the 10×10 problem from Fisher and Thompson [54].

Our implementation uses the algorithm of Baker [58] for selection, the simple (P-GA) as well as the cycle (SB-GA) crossover as described by Goldberg [19], mutation and inversion in case of the P-GA. Near-optimal individuals with high objective function values almost have equal survival probabilities during the selection process if unscaled fitness values serve as selection criterion. While this might be a desirable effect at the beginning of the genetic search process, later on slightly better individuals should survive more often. Scaling the objective function values with respect to some scaling window will meet this requirement. A scaling window of size k means that the worst objective function value f_w in the last k populations is a scaling basis, i.e. each objective function value f in the current population is replaced by $f' := f_w - f$. For window size 0 there is no scaling. We also used an elitist strategy (see Goldberg [19]), i.e. the best individual in each population always survived.

Besides the three well known test instances we generated 105 random problems in order to test our parameter configuration. For each problem type we generated five instances where each job has to be processed by each machine; the machine sequence for each job is randomly generated and the randomly chosen operation processing times are uniformly distributed within the interval [5...99].

Besides our genetic strategies each problem instance (see the tables of computational results) was solved by the SB1-heuristic (SB1), our implementation of the SB2-heuristic (SB2), each of the priority rules where only the result of the priority rule performing best is mentioned (in column "PROP"), a random choice of priority rules, i.e. whenever during the execution of the Giffler and Thompson algorithm a choice point is reached, where a next operation has to be selected, one of the 12 priority rules is chosen at random in order to select the next operation with respect to this priority rule (see column "RND"). Table 2 summarizes the set of control parameters for our genetic algorithms (consider only the first column for each of the algorithms, the second columns are used later).

Fine tuning of these control parameters is of some influence on the quality of the solutions (see Schaffer [59], De Jong and Spears [60], Grefenstette [61] who used a genetic algorithm for the control parameter setting) but changing the instance most often also changes the best parameter configuration. To evaluate different parameter configurations several performance measures may be used.

The online-performance is the average value of all objective function evaluations of the individuals in all populations, while the offline-performance is the average objective function value over all best individuals in all populations. A high online-performance is reached if a high-quality search area can be found very soon, while a high offline performance needs in each population a high-quality solution. Hence the offline performance is not affected while areas of low quality solutions are searched. Other measures of performance are the average objective function value of all members in the current population or the quality of the best solution in the current population.

The whole system was implemented as a general genetic algorithm environment for different types of problems. It was written in PASCAL and all problems were solved on a DECstation 3100 under the operating system Ultrix.

Table 3(a) and 3(b) contain the results for the well known test problems from Fisher and Thompson [54]. The makespan of the best solution found by each of the heuristics and the makespan of an optimal solution for the corresponding problem instance are shown in column "f" and column "OPT", respectively. Column "t (s)" refers to the computation time needed for the corresponding algorithm. Besides "PRIO", "SB1", and "SB2" which present the results of the best priority rule

Table 2. Genetic algorithm control parameters

Parameter	Value for P-GA		Value for SB-GA	
No. of objective function evaluations	60,000	70,000	300	400
Population size	200	300	40	40
Crossover rate	0.65	0.65	0.75	0.65
Mutation rate	0.001	0.0	0.0	0.0
Inversion rate	0.7	0.0	0.0	0.0
Window size for scaling	4	3	4	3
Elitist strategy	Yes	Yes	Yes	Yes

Table 3. The 3 Fisher/Thompson standard problems

	PRIO		RND		P-GA		SB1		SB2		SB-GA		
Problem	f	t (s)	f	t (s)	f	t (s)	f	t (s)	f	t (s)	f	t (s)	OPT
6 × 6	58	0.0	58	12.1	55	11.4	57	0.1	55	3.6	55	19.7	55
5 × 20	1489	0.2	1374	1992.5	1249	1609.6	1274	0.4	1240	10.8	1178	95.7	1165
10 × 10	1191	0.1	1088	1498.1	960	932.6	1031	0.5	951	186.5	938	106.7	930

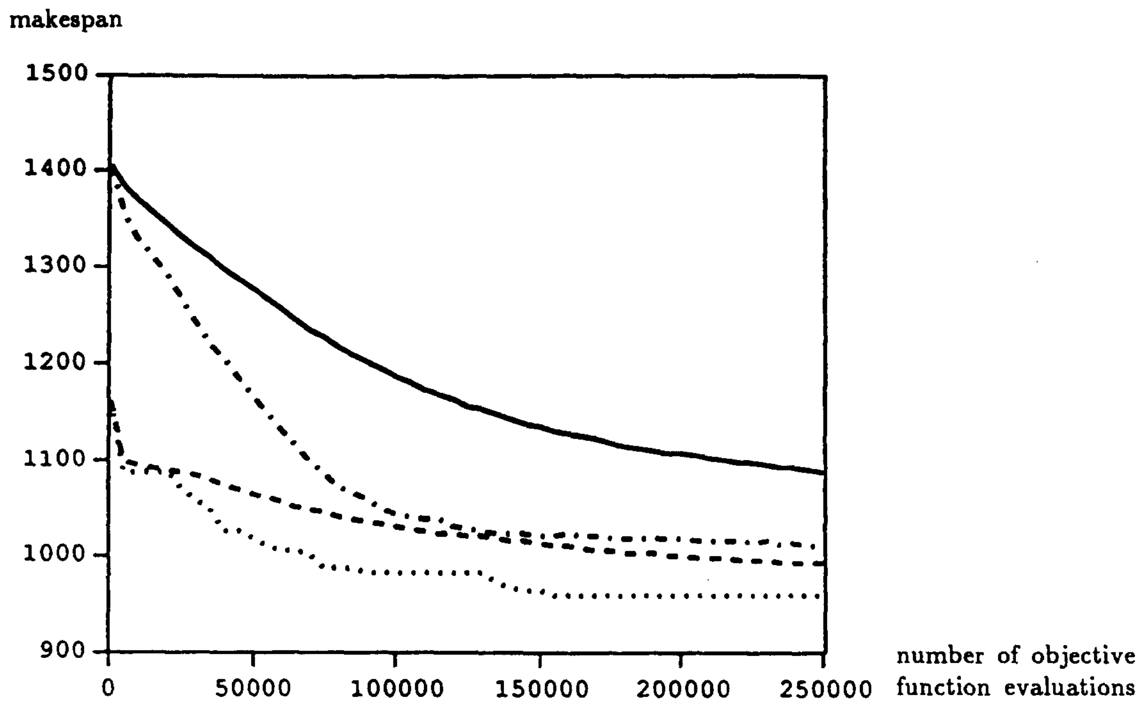

Fig. 5. Performance measures of the P-GA for the 10 × 10 problem. —— Online performance; -·-·- average performance; --- offline performance; ······ best fitness.

and the two shifting bottleneck versions where the enumeration depth *l* was fixed to 4, the columns "P-GA" and "SB-GA" contain the results obtained by the two genetic learning strategies. The results for the randomly chosen priority rules in column "RND" are based on the same number of objective function evaluations (fitness evaluations) as in the P-GA algorithm. The time difference is an effect of the random number generation. Differences of the solution values obtained by the shifting bottleneck heuristics compared to the results from Adams *et al.* [12] probably stem from slightly different implementations of the reoptimization cycle part and different implementations of SB2, respectively (see also the worse results obtained by Applegate and Cook [62] from their shifting bottleneck implementation).

In particular for the famous 10 × 10 problem we wished to have more information about the behaviour of both genetic algorithm strategies. Therefore we increased the number of fitness evaluations substantially. Figures 5 and 6 show the performance of the P-GA over 250,000 fitness evaluations, and the SB-GA over 800 fitness evaluations, respectively. In both figures the vertical axis shows the quality of the performance measure, i.e. the online, the offline, the average performance, and the best objective function value obtained in each population. In both cases the best solution converges to a suboptimum where it remains stable. In the SB-GA the speed of convergence is conspicuous due to the fact that the incorporated heuristic—the shifting bottleneck procedure—is much more powerful than priority rule based heuristics. In both figures the average and the offline performance become almost the same in later steps of the genetic algorithms.

Besides the standard test problems we considered 21 problem types, each consisting of 5 randomly generated problem instances. We applied the same set of heuristics as described for Table 3 to each of these 105 problem instances, the only difference being that because of memory restrictions we chose an enumeration depth $l = 3$ in the SB2 heuristic. Carlier's [47] one machine scheduling branch and bound algorithm provides a lower bound on the makespans for each instance. Table 4 contains for each problem type (column "type", machines × jobs) the average relative deviation over 5 instances from the lower bound in percent. The computation times on average over 5 instances on a DECstation 3100 can be found in Table 5. A dash entry in the SB2 column indicates that the breadth-first search of the SB2 heuristic caused a memory problem (excessive paging).

The worst solutions are those obtained by the priority rules (column "PRIO"). Among all 105

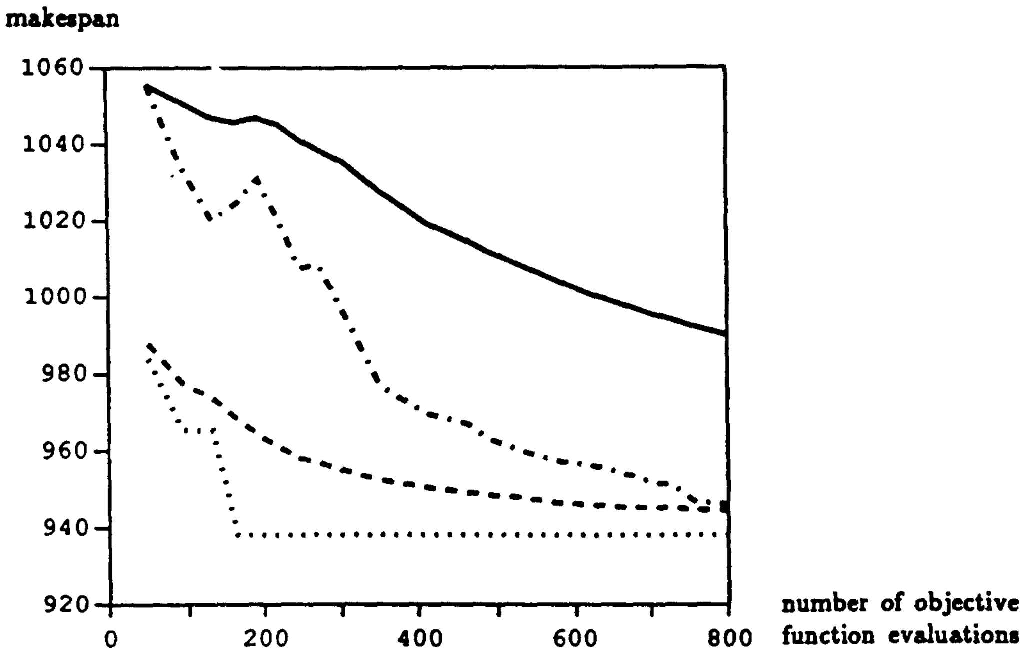

Fig. 6. Performance measures of the SB-GA for the 10 × 10 problem. —— Outline performance; –·–·– average performance; --- offline performance; ······ best fitness.

Table 4. Performance comparison of 6 algorithms: average relative deviation from the lower bound in percent

Type	PRIO	RND	P-GA	SB1	SB2	SB-GA
5 × 10	18.7	13.9	4.5	4.1	3.8	3.2
5 × 15	16.2	6.8	0.6	1.4	0.2	0.2
5 × 20	15.7	8.0	1.0	0.1	0.0	0.0
10 × 5	15.2	3.3	3.2	9.6	5.3	4.6
15 × 5	7.8	3.1	3.4	4.6	3.1	3.1
20 × 5	8.2	3.3	1.8	3.8	—	3.1
10 × 10	36.0	24.2	15.2	22.3	13.2	12.0
10 × 15	31.2	28.3	13.6	13.5	8.9	7.2
10 × 20	21.6	21.7	7.4	7.4	2.0	1.7
15 × 10	28.1	20.5	13.2	15.4	10.5	10.3
20 × 10	26.2	21.9	13.8	17.6	—	11.6
5 × 5	16.8	7.7	7.7	9.7	8.2	8.2
6 × 6	20.3	8.5	7.8	12.5	7.2	7.1
7 × 7	22.7	9.8	8.7	12.7	7.7	6.9
8 × 8	29.8	17.3	13.3	17.6	11.6	10.7
9 × 9	34.8	23.0	15.4	19.0	12.2	11.4
11 × 11	31.2	26.6	15.0	16.6	11.5	10.0
12 × 12	34.6	24.3	11.7	12.1	8.7	8.1
13 × 13	35.4	29.0	16.1	19.5	12.9	11.5
14 × 14	38.4	35.8	18.3	21.3	15.7	15.0
15 × 15	38.3	32.0	18.1	19.9	13.6	13.6

runs the LOT-rule, the LNRO-rule, the LORPT-rule, the LPT-rule, and the Random rule yielded the best solution 45, 30, 28, 1, and 1 times, respectively; the other priority rules never dominated. A random choice of a priority rule in order to detect the next operation to be scheduled turned out to yield better solutions than a fixed choice of a priority rule. However the running times are pretty high, even higher than in P-GA, due to the elaborative random number generation. In general the SB2 heuristic dominates P-GA, however, for a large number of machines the conflict sets from Giffler and Thompson's algorithm tend to be small so that the P-GA could be superior to SB2. If the performance measure is put into relation to the running time then SB1 may become the favorite algorithm. As expected, if the problem sizes turn out to be large enough SB-GA dominates all others, for both, the quality of the obtained solutions as well as the running time.

Table 5. Average running times for the results of Table 4 on a DECstation 3100

Type	PRIO	RND	P-GA	SB1	SB2	SB-GA
5 × 10	0.05	195.0	186.9	0.1	1.2	21.4
5 × 15	0.1	324.7	304.1	0.1	1.6	34.3
5 × 20	0.2	482.7	449.5	0.2	2.1	59.2
10 × 5	0.05	166.1	158.2	0.3	37.1	48.2
15 × 5	0.05	236.9	212.2	0.7	350.1	107.5
20 × 5	0.1	307.1	263.6	1.6	—	148.5
10 × 10	0.1	365.8	324.4	0.6	75.5	101.7
10 × 15	0.2	628.5	559.3	1.0	116.2	169.2
10 × 20	0.4	955.2	852.5	1.6	116.3	253.8
15 × 10	0.2	538.5	459.9	1.7	696.7	241.4
20 × 10	0.2	716.6	597.6	4.1	—	442.5
5 × 5	0.05	97.5	110.8	0.05	0.8	12.6
6 × 6	0.05	131.1	136.0	0.1	3.0	20.3
7 × 7	0.05	175.3	171.2	0.1	7.7	32.6
8 × 8	0.05	227.8	212.9	0.2	19.6	48.9
9 × 9	0.1	290.5	264.2	0.4	33.1	71.9
11 × 11	0.1	451.0	394.5	0.7	136.7	247.3
12 × 12	0.2	554.6	482.3	1.3	244.7	189.4
13 × 13	0.2	664.8	542.4	1.9	484.6	250.5
14 × 14	0.3	793.9	634.1	2.4	845.9	319.9
15 × 15	0.3	940.4	882.3	3.3	1257.8	414.5

A main aspect to consider SB-GA as a favorite is also its amount of memory in use; SB-GA uses very little memory compared to the SB2-heuristic. The latter, however, can be implemented similarly at the expense of its running time.

The encouraging results for the 105 randomly generated problems and the 3 famous instances as well as the desire to compete with alternative heuristics reported in the literature led us to consider the behaviour of our algorithms for a well known testbed of 40 problem instances. It is the same set of instances underlying the shifting bottleneck experiments of Adams *et al.* [12], the genetic local search experiments of Aarts *et al.* [33], as well as the simulated annealing experiments of van Laarhoven *et al.* [63]. The optimum solution of most of the instances is known (cf. Brucker *et al.* [64]).

We avoided spending hours of computation time on fine parameter tuning and chose our parameters with respect to the outcomes of our former experiments. Different parameter settings for the different problem sizes probably would lead to slight performance improvements in particular for the larger and the smaller problem instances, but we adopted the same overall control parameters that seemed to be reasonable for middle sized instances. The parameters correspond to the entries of the second columns in Table 2 for each of the two algorithms.

Table 6 contains our results compared to those obtained by alternative approximation algorithms. The numbering of the problems in the first column is identical to that one in Adams *et al.* [12]. We dropped the instances 31 to 35 because the SB1 implementation already solved them to optimality in a fraction of the time needed by the SB-GA. Column "SB-GA (40)" corresponds to the parameter setting as described in Table 2. Additionally, we also ran SB-GA for the larger problems from 16 to 40 on population sizes of 60 limited to 800 function evaluations [see column "SB-GA (60)"]. The results reported for SB-GA are average results over two runs while P-GA was run only once. The initial population was always randomly generated. Column "SB-ABZ" contains the makespans that Adams *et al.* [12] obtained from their shifting bottleneck implementation. Columns "*f*-GLS" and "*f*-SA" contain the best of two results obtained by two different implementations for each, a genetic local search strategy and simulated annealing, respectively. These four algorithms got the same restrictions from column "*t* (s)". The last three columns are headed by the name "ALU" indicating that their values are taken from Aarts *et al.* [33], and the part after the decimal point is truncated. Finally the last column "opt" presents the smallest makespan published in Carlier and Pinson [65], Brucker *et al.* [64]. Applegate and Cook [62], or Dell'Amico and Trubian [66], for the particular problem instance. An asterisk (*) says that the value is proved to be optimal.

The values in the "ALU" column are average results over 5 runs on a VAX 8650. The

Table 6. Computational results on a DECstation 3100 for 35 well known problem instances. Comparison to the results of the shifting bottleneck procedure from Adams *et al.* [12] ("SB-ABZ") on a VAX 780/11 and to the results from Aarts *et al.* [33] ("ALU") on a VAX 8650 for two genetic local search ("*f*-GLS") and two simulated annealing ("*f*-SA") approaches. Column "opt" contains the value of an optimal (*) or best known solution, which results from the exact algorithms of Brucker *et al.* [64] (BJS), Applegate and Cook [62] (AC), Carlier and Pinson [65] (CP), or from tabu search by Dell'Amico and Trubian [66] (DT)

	P-GA		SB-GA (40)		SB-GA (60)		SB-ABZ	ALU			
Problem	*f*	*t* (s)	*f*	*t* (s)	*f*	*t* (s)	*f*	*f*-GLS	f-SA	*t* (s)	Opt
					5 machines, 10 jobs						
01	666	106	666	15.6			666	666	666	17	666*BJS
02	681	108	666	16.4			669	659	658	18	655*CP
03	620	104	604	11.3			605	609	616	21	597*BJS
04	620	106	590	11.1			593	594	590	16	590*BJS
05	593	106	593	14.8			593	593	593	13	593*BJS
					5 machines, 15 jobs						
06	926	185	926	25.2			926	926	926	25	926*CP
07	890	182	890	27.6			890	890	890	42	890*BJS
08	863	184	863	26.1			863	863	863	39	863*BJS
09	951	186	951	23.6			951	951	951	35	951*BJS
10	958	185	958	24.3			959	958	958	17	958*BJS
					5 machines, 20 jobs						
11	1222	290	1222	39.3			1222	1222	1222	59	1222*BJS
12	1039	292	1039	35.9			1039	1039	1039	50	1039*BJS
13	1150	289	1150	33.5			1150	1150	1150	50	1150*BJS
14	1292	289	1292	34.9			1292	1292	1292	21	1292*BJS
15	1237	282	1207	48.8			1207	1207	1207	76	1207*BJS
					10 machines, 10 jobs						
16	1008	191	961	76.3	961	160.3	978	976	969	88	945*CP
17	809	192	787	77.7	784	163.3	787	791	785	91	784*CP
18	916	190	848	76.2	848	165.8	859	856	856	96	848*BJS
19	880	191	863	77.4	848	161.3	860	859	854	93	842*BJS
20	928	188	911	80.4	910	158.5	914	913	911	107	902*BJS
					10 machines, 15 jobs						
21	1139	352	1074	134.8	1074	292.8	1084	1084	1078	243	1050 BJS
22	998	350	935	136.1	936	290.0	944	944	950	254	927*BJS
23	1072	353	1032	139.6	1032	295.6	1032	1032	1032	242	1032*CP
24	1014	352	960	137.3	957	289.0	976	960	960	234	935*AC
25	1014	350	1008	134.2	1007	228.9	1017	1010	1003	254	977*AC
					10 machines, 20 jobs						
26	1278	569	1219	227.3	1218	442.8	1224	1236	1221	469	1218*BJS
27	1378	565	1272	242.5	1269	446.2	1291	1300	1275	492	1242 DT
28	1327	569	1240	221.8	1241	463.5	1250	1264	1242	455	1216*DT
29	1336	570	1204	241.0	1210	453.1	1239	1260	1225	471	1182 DT
30	1411	568	1355	230.5	1355	455.6	1355	1386	1355	441	1355*CP
					15 machines, 15 jobs						
36	1373	524	1317	335.6	1317	688.1	1305	1310	1307	602	1268*CP
37	1498	520	1484	350.5	1446	665.9	1423	1449	1440	636	1402 CP
38	1296	525	1251	335.7	1241	665.9	1255	1283	1235	635	1203 DT
39	1351	525	1282	327.2	1277	687.5	1273	1279	1258	592	1233*BJS
40	1321	526	1274	348.0	1252	698.4	1269	1260	1254	596	1222*AC

neighborhood structure for the simulated annealing and the genetic local search versions are the same: (i) reversing an edge on a longest path in the graph, and (ii) reversing an edge on a longest path in the graph such that this edge is incident to a critical edge of the arc set A. For details we refer to Aarts *et al.* [33]. In the genetic local search algorithms a local search procedure is performed after the recombination step of the genetic algorithm such that each member of the population is made locally optimal with respect to the two neighbourhood structures.

Although P-GA could even find an optimal solution for several problems, and probably can do better for larger population sizes (for instance: for a population size of 500 we got the makespans 987, 807, 855, 863, 928, for the five instances from 16 to 20, respectively, within an average running time of 180 s for 70,000 function evaluations), the P-GA is never a serious competitor to other heuristics because of its running time compared to the quality of the results. However, for the largest instances the relation between running time and quality is quite good.

For the smaller problems (from 1 to 15) GLS, SA, and SB-GA are almost equal in their solution quality; the optimum could be found in nearly all runs. For these problem instances the pure shifting bottleneck implementation is the winner in both, quality of the solution and the time. For the larger instances from 16 to 30, where the running time of the shifting bottleneck implementation is still the lowest, SB-GA performs better than the pure shifting bottleneck alternative as well as the GLS and SA algorithms (even if we consider the running times to the latter as comparable). For the last five instances the performance of simulated annealing and SB-GA is quite comparable. (Remember that columns "f-GLS" and "f-SA" contain always the best of two values.)

We would like to mention that for the famous 5×20 and 10×10 problem the makespans published in Aarts *et al.* [33] are 1294 (GLS)/1216 (SA) and 978 (GLS)/969 (SA) found in 88 and 99 s, respectively. Hence, under fine parameter tuning SB-GA clearly dominates (see Table 3).

Although P-GA is less powerful than the other approaches it has its advantages: ease of implementation, also in a more general framework, as well as robustness to problem changes (SB-GA seems to be much more sensitive) can give P-GA the user's preference, even to simulated annealing.

CONCLUSIONS

We presented a special type of probabilistic learning by population genetics. Learning to find a best sequence of priority rules for solving job shop scheduling problems did not work best because of the amount of computation time needed. This is not a big suprise as we do not know if small makespan schedules can be produced by the fixed set of priority rules from Table 1. Moreover, priority rules only make little use of problem specific knowledge. In contrast, the solution of a one machine problem (with respect to some former decisions) involves a lot of problem specific knowledge, the use of which may be the main reason for the success of the genetically guided shifting bottleneck procedure. Introducing problem specific knowledge makes a genetic strategy a promising alternative. This knowledge might be incorporated by the application of a heuristic performing well on the particular problem, such as the shifting bottleneck procedure for the job shop problem and the Lin–Kernighan algorithm for the travelling salesman problem. We may also think of local search algorithms such as tabu search or simulated annealing as "special purpose" heuristics. So a simple genetic algorithm can do its work very well if an individual is considered to be a sequence of local decisions on the use of the knowledge provided by special purpose heuristics or parts of them.

REFERENCES

1. J. K. Lenstra and A. H. G. Rinnooy Kan, Computational complexity of discrete optimization problems. *Annals Discr. Math.* **4**, 121–140 (1979).
2. E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan and D. B. Shmoys, Sequencing and scheduling: algorithm and complexity. CWI-report BS-R8909, CWI Amsterdam (1989).
3. J. F. Muth and G. L. Thompson (Eds), *Industrial Scheduling.* Prentice Hall, Englewood Cliffs (1963).
4. R. N. Conway, W. L. Maxwell and L. W. Miller, *Theory of Scheduling.* Addison–Wesley, Reading (1967).
5. K. R. Baker, *Introduction to Sequencing and Scheduling.* Wiley, New York (1974).
6. S. French, *Sequencing and Scheduling: An Introduction to the Mathematics of the Job Shop.* Wiley, New York (1982).
7. J. K. Lenstra, Sequencing by Enumerative Methods. Mathematical Center Tract 69, Mathematisch Centrum, Amsterdam (1977).
8. A. H. G. Rinnooy Kan, *Machine Scheduling Problems: Classification, Complexity and Computations.* Nifhoff, The Hague (1976).
9. J. Blazewicz, Selected topics in scheduling theory. *Annals Discrete Math.* **31**, 1–60 (1987).
10. A. S. Manne, On the job shop scheduling problem. *Opns Res.* **8**, 219–223 (1960).
11. J. Blazewicz, M. Dror and J. Weglarz, Mathematical programming formulations for machine scheduling: a survey. *Eur. J. Opl Res.* **51**, 283–300 (1991).
12. J. Adams, E. Balas and D. Zawack, The shifting bottleneck procedure for job shop scheduling. *Mgmt Sci.* **34**, 391–401 (1988).
13. B. Roy and B. Sussmann, Les problèmes d'ordonnancement avec contraintes disjonctives. SEMA, Note D.S. Note D.S. No. 9, Paris (1964).
14. E. Balas, Machine sequencing via disjunctive graphs: an implicit enumeration algorithm. *Opns Res.* **17**, 941–957 (1969).
15. S. S. Panwalkar and W. Iskander, A survey of scheduling rules. *Opns Res.* **25**, 45–61 (1977).
16. I. Rechenberg, Optimieuring technischer Systeme nach Prinzipien der biologischen Evolution. *Problemata.* Frommann-Holzboog (1973).
17. J. H. Holland. *Adaptation in in Natural and Artificial Systems.* The University of Michigan Press, Ann Arbor (1975).
18. H.-P. Schwefel, *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie.* Birkhäuser, Basel (1977).
19. D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison–Wesley, Reading (1989).
20. D. E. Goldberg, Zen and the art of genetic algorithms. *Proc. 3rd Int. Conf. Genetic Algorithms* (Edited by J. D. Schaffer), pp. 80–85. Morgan Kaufmann (1989).
21. G. E. Liepins and M. R. Hilliard, Genetic algorithms: foundations and applications. *Annals Opns Res.* **21**, 31–57 (1989).

22. Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, Berlin (1992).
23. E. H. L. Aarts and J. Korst, *Simulated Annealing and Boltzmann Machines*. Wiley, Chichester (1989).
24. F. Glover, Tabu search—Part I. *ORSA J. Comput.* **1**, 190–206 (1989).
25. F. Glover, Tabu search—Part II. *ORSA J. Comput.* **2**, 4–32 (1990).
26. A. E. Eiben, E. H. L. Aarts and K. H. van Hee, Global convergence of genetic algorithms: a Markov Chain analysis. *Lecture Notes Comput. Sci.* **496**, 4–9 (1991).
27. J. J. Grefenstette, Incorporating problem specific knowledge into genetic algorithms. *Genetic Algorithms and Simulated Annealing* (Edited by L. Davis), pp. 42–60. Pitman, London (1987).
28. P. Jog, J. Y. Suh and D. Van Gucht, The effects of population size, heuristic crossover and local improvement on a genetic algorithm for the traveling salesman problem. *Proc. 3rd Int. Conf. Genetic Algorithms* (Edited by J. D. Schaffer), pp. 110–115. Morgan Kaufmann (1989).
29. J. Y. Suh and D. Van Goucht, Incorporating heuristic information into genetic search. *Proc. 2nd Int. Conf. Genetic Algorithms and Their Applications* (Edited by J. J. Grefenstette), pp. 100–107. Lawrence Erlbaum (1987).
30. N. L. J. Ulder, E. H. L. Aarts, H.-J. Bandelt, P. J. M. van Laarhoven and E. Pesch, Genetic local search algorithm for the traveling salesman problem. *Lecture Notes Comput. Sci.* **496**, 109–116 (1991).
31. D. S. Johnson, Local optimization and the traveling salesman problem. *Proc. 17th Colloq. Automata, Languages, and Programming*, pp. 446–461. Springer, New York (1990).
32. A. Kolen and E. Pesch, Genetic local search in combinatorial optimization. *Discrete Appl. Math.* **48**, 273–284 (1994).
33. E. H. L. Aarts, P. J. M. van Laarhoven and N. L. J. Ulder, Local-search-based algorithms for job shop scheduling. Working paper, University of Eindhoven (1991).
34. R. Nakano and T. Yamada, Conventional genetic algorithm for job shop problems. *Proc. 4th Int. Conf. Genetic Algorithms and their Applications*, San Diego, pp. 474–479 (1991).
35. T. Yamada and R. Nakano, A genetic algorithm applicable to large-scale job-shop problems. *Proc. 2nd Int. Workshop on Parallel Problem Solving from Nature* (Edited by R. Männer and B. Manderick), pp. 281–290 (1992).
36. H. Mühlenbein, M. Gorges-Schleuter and O. Krämer, New solutions to the mapping problem of parallel systems: the evolution approach. *Parallel Computing* **4**, 269–279 (1987).
37. H. Mühlenbein, M. Gorges-Schleuter and O. Krämer, Evolution algorithm in combinatorial optimization. *Parallel Computing* **7**, 65–85 (1988).
38. M. Gorges-Schleuter, ASPARAGOS, a parallel genetic algorithm and population genetics. *Proc. 3rd Int. Conf. Genetic Algorithms* (Edited by J. D. Schaffer), pp. 422–427. Morgan Kaufmann (1989).
39. P. Ablay, Optimieren mit Evolutionsstrategien. *Spektrum Wissenschaft* **7**, 162–173 (1987).
40. S. Stöppler and C. Bierwirth, The application of a parallel genetic algorithm to the $n/m/P/C_{\max}$ flowshop problem. Working paper, University of Bremen (1992).
41. B. Giffler and G. L. Thompson, Algorithms for solving production scheduling problems. *Opns Res.* **8**, 487–503 (1960).
42. R. Haupt, A survey of priority-rule based scheduling. *OR Spektrum* **11**, 3–16 (1989).
43. J. H. Blackstone, D. T. Phillips and G. L. Hogg, A state of the art survey of dispatching rules for manufacturing job shop operations. *Int. J. Prod. Res.* **20**, 27–45 (1982).
44. S. Dauzere-Peres and J.-B. Lasserre, A modified shifting bottleneck procedure for job-shop scheduling. *Int. J. Prod. Res.* **31**, 923–932 (1993).
45. E. Balas, J. K. Lensgra and A. Vazacopoulos, One machine scheduling with delayed precedence constraints. Management Science Research Report MSRR-589, Carnegie Mellon University, Pittsburgh (1992).
46. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco (1979).
47. J. Carlier, The one machine sequencing problem. *Eur. J. Opl Res.* **11**, 42–47 (1982).
48. L. Davis, Job shop scheduling with genetic algorithms. *Proc. Int. Conf. Genetic Algorithms and Their Applications* (Edited by J. J. Grefenstette), pp. 136–140. Lawrence Erlbaum (1985).
49. D. Whitley, T. Starkweather and D. Fuquay, Scheduling problems and traveling salesmen: the genetic edge recombination operator. *Proc. 3rd Int. Conf. Genetic Algorithms* (Edited by J. D. Schaffer), pp. 133–140. Morgan Kauffmann (1989).
50. P. Husbands, F. Mill and S. Warrington, Genetic algorithms, production plan optimisation and scheduling. *Proc. 1st Int. Workshop on Parallel Problem Solving from Nature* (Edited by H.-P. Schwefel and R. Männer). *Lecture Notes Comput. Sci.* **496**, 80–84 (1991).
51. T. Starkweather, D. Whitley, K. Mathias and S. McDaniel, Sequence scheduling with genetic algorithms. In *New Directions for Operations Research in Manufacturing* (Edited by G. Fandel, T. Gulledge and J. Jones), pp. 129–148. Springer, Berlin (1992).
52. M. R. Hilliard and G. E. Liepins, Machine learning applications to job shop scheduling. *Proc. Workshop on Production Planning and Scheduling*. AAAI-SIGMAN, St Paul, MN (1988).
53. R. H. Storer, S. D. Wu and R. Vaccari, New search spaces for sequencing problems with application to job shop scheduling. *Mgmt Sci.* **38**, 1495–1509 (1992).
54. H. Fisher and G. L. Thompson, Probabilistic learning combinations of local job-shop scheduling rules. In *Industrial Scheduling* (Edited by J. F. Muth and G. L. Thompson). Prentice Hall, Englewood Cliffs (1963).
55. W. B. Crowston, F. Glover, G. L. Thompson and J. D. Trawick, Probabilistic and parametric learning combinations of local job shop scheduling rules. ONR Research Memorandum No. 117, GSIA, Carnegie-Mellon University, Pittsburgh (1963).
56. P. J. O'Grady and C. Harrison, A general search sequencing rule for job shop sequencing. *Int. J. Prod. Res.* **23**, 951–973 (1985).
57. F. Glover, Future paths for integer programming and links to artificial intelligence. *Computers Opns Res.* **13**, 533–549 (1986).
58. J. E. Baker, Reducing bias and inefficiency in the selection algorithm. *Proc. 2nd Int. Conf. Genetic Algorithms and Their Applications* (Edited by J. J. Grefenstette), pp. 14–21. Lawrence Erlbaum, London (1987).
59. J. D. Schaffer, R. A. Caruane, L. J. Eshelman and D. Rajarshi, A study of control parameters affecting online performance of genetic algorithms for function optimization. *Proc. 3rd Int. Conf. Genetic Algorithms* (Edited by J. D. Schaffer), pp. 51–60. Morgan Kaufmann (1989).
60. K. A. De Jong and W. M. Spears, Using genetic algorithms to solve NP-complete problems. *Proc. 3rd Int. Conf. Genetic Algorithms* (Edited by J. D. Schaffer), pp. 61–60. Morgan Kaufmann (1989).

61. J. J. Grefenstette, Optimization of control parameters for genetic algorithms. *IEEE Trans. Syst. Man Cybernet.* **SMC-16**, 122–128 (1986).
62. D. Applegate and W. Cook, A computational study of the job-shop scheduling problem. *ORSA J. Comput.* **3**, 149–156 (1991).
63. P. J. M. van Laarhoven, E. H. L. Aarts and J. K. Lenstra, Job shop scheduling by simulated annealing. *Opns Res.* **40**, 113–125 (1992).
64. P. Brucker, B. Jurisch and B. Sievers, A fast branch & bound algorithm for the job-shop scheduling problem. *Discrete Appl. Math.* In press.
65. J. Carlier and E. Pinson, A practical use of Jackson's preemtive schedule for solving the job shop problem. *Annals Opns Res.* **26**, 269–287 (1990).
66. M. Dell'Amico and M. Trubian, Applying tabu-search to the job shop scheduling problem. *Annals Opns Res.* In press.