Chapter 1

GENETIC ALGORITHMS FOR SHOP SCHEDULING PROBLEMS: A SURVEY

Frank Werner*

Otto-von-Guericke-Universität, Fakultät für Mathematik, 39106 Magdeburg, Germany

Key Words: Scheduling, Genetic algorithms, Flow shop, Job shop, Open shop

AMS Subject Classification: 90B35, 90C57.

*E-mail address: frank.werner@mathematik.uni-magdeburg.de or frank.werner@ovgu.de

ABSTRACT

Genetic algorithms are a very popular heuristic which have been successfully applied to many optimization problems within the last 30 years. In this chapter, we give a survey on some genetic algorithms for shop scheduling problems. In a shop scheduling problem, a set of jobs has to be processed on a set of machines such that a specific optimization criterion is satisfied. According to the restrictions on the technological routes of the jobs, we distinguish a flow shop (each job is characterized by the same technological route), a job shop (each job has a specific route) and an open shop (no technological route is imposed on the jobs). We also consider some extensions of shop scheduling problems such as hybrid or flexible shops (at each processing stage, we may have a set of parallel machines) or the inclusion of additional processing constraints such as controllable processing times, release times, setup times or the no-wait condition. After giving an introduction into basic genetic algorithms discussing briefly solution representations, the generation of the initial population, selection principles, the application of genetic operators such as crossover and mutation, and termination criteria, we discuss several genetic algorithms for the particular problem types emphasizing their common features and differences. Here we mainly focus on single-criterion problems (minimization of the makespan or of a particular sum criterion such as total completion time or total tardiness) but mention briefly also some work on multi-criteria problems. We discuss some computational results and compare them with those obtained by other heuristics. In addition, we also summarize the generation of benchmark instances for makespan problems and give a brief introduction into the use of the program package 'LiSA - A Library of Scheduling Algorithms' developed at the Otto-von-Guericke-University Magdeburg for solving shop scheduling problems, which also includes a genetic algorithm.

1 INTRODUCTION

Genetic algorithms belong to the class of *evolutionary algorithms*. These are algorithms which are based on the principles of natural evolution, and they can be divided into four major types of algorithms: genetic algorithms (GA), genetic programming, evolution strategies and evolutionary programming. All these types of algorithms are based on a population of individuals. Evolutionary algorithms have been applied to many problems in management, e.g., to location, inventory, production, scheduling, distribution or timetabling problems (for an overview on such applications see e.g. [76]).

The use of evolutionary algorithms for shop scheduling problems started around 1980. Two of the first applications to flow shop scheduling problems have been given by Werner [121, 122], and the first application to job shop scheduling problems can be found in [27]. Genetic algorithms are the most popular variant of evolutionary algorithms. The structure

and components of elementary genetic algorithms has been discussed e.g. by Goldberg [40] or Beasley et al. [10]. Evolution strategies have been originally developed for optimization problems in engineering. Here one can mention the pioneering works by Rechenberg [93] and Schwefel [102].

Although both types of evolutionary algorithms have several common features, there also exist some differences (see e.g. [51]). Evolution strategies typically work directly with real-valued vectors, genetic algorithms often use strings of bits. While in genetic algorithms recombination in the form of using crossover operators plays a dominant role, evolution strategies mainly use mutation in the form of small changes of particular real variables. Evolution strategies also use some type of recombination, often in the form of discrete recombination for generating the offspring (i.e., it is decided for each component the value of which of the two parents is used for the offspring) and intermediate recombination to determine the strategy parameters. While in genetic algorithms often the parameters for applying specific genetic operators are constant, the strategy parameters in evolution strategies typically underly an adaptation process. Genetic algorithms are particularly applied to combinatorial optimization problems so that in the following, we mainly focus on this class of evolutionary algorithms.

During the last decades, there appeared a huge number of publications dealing with genetic algorithms for shop scheduling problems. Of course, not all of them can be included into this overview. Since the goal was not to exceed approximately 50 - 60 pages, there had to be made a selection among the existing approaches to be included into this survey (which, of course, represents a bit the personal view on the subject). We describe some basic algorithms with standard components of a GA more detailed and briefly sketch the development of some more advanced works in order to stimulate further investigations in this area. The applied operators are summarized, and we give an insight into the parameters used in the tests. We mainly focus on shop scheduling problems with a single criterion, but briefly review also some papers dealing with multiple objectives. Moreover, we discuss here mainly journal and working papers but no complete books on genetic algorithms. We also skip some previous papers which have been discussed more in detail in one the papers mentioned in this review.

The rest of this chapter is organized as follows. In Section 2, we give a brief introduction into shop scheduling problems and sketch some of the extensions. In Section 3, we give some comments on the generation of test instances for the particular classes of shop scheduling problems. In particular, we also give a reference to standard benchmark instances often used nowadays for testing new algorithms for makespan minimization problems. In Section 4, we give a brief general introduction into genetic algorithms and their components, where the genetic operators are explained using the permutation representation of a solution. Then, in Section 5 - 7, we discuss some typical algorithms developed for the three major classes of shop scheduling problems, namely flow shop problems (Section 5), job shop problems (Section 6) and open shop problems (Section 7). For flow and job shop scheduling problems, we also include a subsection dealing with hybrid or flexible shop problems. Finally, we sketch in Section 8 the use of the program package LiSA for applying a standard genetic algorithm, and Section 9 gives some conclusions.

2 SHOP SCHEDULING PROBLEMS AND EXTENSIONS

Shop scheduling problems belong to the class of multi-stage scheduling problems, where each job consists of a set of operations. For describing these problems, we use the standard 3-parameter classification $\alpha \mid \beta \mid \gamma$ introduced in [42]. The parameter α indicates the machine environment, the parameter β describes job characteristics, and the parameter γ gives the optimization criterion.

In such a shop scheduling problem, a set of n jobs $J_1, J_2, \ldots, J_n$ has to be processed on a set of m machines $M_1, M_2, \ldots, M_m$. The processing of a job J_i on a particular machine M_j is denoted as an operation and abbreviated by (i, j). Each job J_i consists of a number n_i of operations. For the deterministic scheduling problems, the processing time p_{ij} of each operation (i, j) is given in advance.

Among the shop scheduling problems, there are three basic types: a *flow-shop*, a *job-shop* and an *open-shop*. In a flow shop problem ($\alpha = F$), each job has exactly m operations, and the technological route (or machine order) in which the job passes through the machines is the same for any job. Without loss of generality, we assume that the technological route for any job is given by $M_1 \to M_2 \to \ldots \to M_m$. In a job shop problem ($\alpha = J$), a specific technological route $M_{j_1} \to M_{j_2} \to \ldots M_{j_{n_i}}$ is given for each job $J_i, 1 \leq i \leq n$. Note that the number of operations per job n_i is equal to m for the classical job shop problems, but this number may be also smaller than m or larger than m (recirculation; in this case we may use the notation (i, j, k) for the k-th processing of job J_i on machine M_j). In an open shop problem ($\alpha = O$), no technological routes are imposed on the jobs. Usually, it is assumed that each job has to be processed on any machine. In addition to these three major types, there also exist generalizations such as a mixed shop or a general shop.

Most scheduling problems considered are deterministic ones, where all data are known and given in advance. For any job J_i, there might be given a release date $r_i \geq 0$, a due date $d_i \geq 0$ and/or a weight $w_i > 0$. The occurrence of release dates is indicated in the parameter β as $r_i \geq 0$, and the occurrence of due dates is indicated as $d_i \geq 0$. In addition, there can be many other processing constraints. For instance, waiting times between the processing of the operations of a job may be forbidden ($\beta = no - wait$). Another extension is the explicit representation of sequence-independent or sequence-dependent set-up times (or costs) between the processing of consecutive operations of jobs on the same machine. Consideration of setup times may also happen in the case of batch processing. An overview on scheduling with setup times and costs can be found e.g. in [4]. A *re-entrant* shop means that a job must be processed repeatedly according to the given technological route, where the level L denotes the number of times the technological routes have to be performed.

The parameter γ indicates the optimality criterion. Some typical optimality criteria are the minimization of the makespan C_{max}, the minimization of the sum of the weighted completion times (or total weighted completion time) $\sum w_i C_i$, where C_i is the completion time of job J_i, the minimization of total weighted tardiness $\sum w_i T_i$, where $T_j = \max\{0, C_j - d_j\}$ or the corresponding unweighted problems ($w_i = 1$ for all jobs).

Another type of problems are scheduling problems with uncertain data, e.g. the processing times can be given as interval times or as fuzzy times.

For describing feasible solutions, one has to specify the job orders on the machines.

This can be done by giving all job orders explicitly as a *job sequence* (i.e., a permutation of the job indices), or by combining the technological routes and the job orders into the *rank matrix* R, where r_{ij} denotes the rank of operation (i,j) in the underlying graph. Note that such a rank matrix corresponds to a specific latin rectangle with the additional sequence property: To each number $r_{i,j} = k$, there exists the number $k-1$ in row i or column j (or in both). To illustrate, consider a schedule of an open shop problem with $n = 3$ jobs, $m = 4$ machines and the rank matrix

$$R = \begin{pmatrix} 4 & 3 & 2 & 1 \\ 2 & 4 & 1 & 5 \\ 1 & 5 & 3 & 4 \end{pmatrix}.$$

From the matrix R, we can immediately obtain the machines orders of the jobs by ordering the entries of row i for job J_i. This gives

$$\begin{array}{lllllll} J_1: & M_4 & \rightarrow & M_3 & \rightarrow & M_2 & \rightarrow & M_1 \\ J_2: & M_3 & \rightarrow & M_1 & \rightarrow & M_2 & \rightarrow & M_4 \\ J_3: & M_1 & \rightarrow & M_3 & \rightarrow & M_4 & \rightarrow & M_2 \end{array}$$

Similarly, we can obtain the job order on machine $M_j, 1 \leq j \leq m$, by sorting the entries of column j from the smallest to the largest one. This yields

$$\begin{array}{llllll} M_1: & J_3 & \rightarrow & J_2 & \rightarrow & J_1 \\ M_2: & J_1 & \rightarrow & J_2 & \rightarrow & J_3 \\ M_3: & J_2 & \rightarrow & J_1 & \rightarrow & J_3 \\ M_4: & J_1 & \rightarrow & J_3 & \rightarrow & J_2 \end{array}$$

For an overview on results and algorithms for flow shop problems with the makespan criterion, we refer the reader to [96,97]. A review on flow shop problems particularly for the minimization of total tardiness has been given in [115], where 40 heuristic algorithm have been compared in detail (including a GA from [81]). A recent overview on optimization algorithms for the flow shop scheduling problem with several objectives can be found in [108]. A detailed survey on job shop problems can be found e.g. in [48]. A recent survey of milestones in fifty years of scheduling can be found in [88].

In addition to the classical shop problems problems, there have been considered several extensions. One of these generalizations are *hybrid or flexible* shops. This is a combination of a shop scheduling problem and a parallel machine scheduling problem. It is also a multi-stage scheduling problem, in which some or all of the k stages consist of several parallel (i.e., either identical, uniform or unrelated) machines. Most of the flexible shop problems considered in the literature are of the flow shop (FFS) or the job shop (FJS) type. A review on recent developments, algorithms and results for FFS problems can be found in [100]. For the open shop environment, there are practically no results on flexible shops available. Note that we do not distinguish between hybrid and flexible shops while in some papers, flexible shops are related to the case of identical parallel machines and hybrid shops are related to the general case of non-identical parallel machines.

3 GENERATION OF TEST INSTANCES

While the first papers on shop scheduling problems generated test instances often based on uniformly distributed integers (e.g. for the processing times or other data), this often does not generate hard instances. An interesting discussion from an overall point of view how experimental data should be generated can be found e.g. in [44].

During the last two decades, a few sets of benchmark instances have been generated which are now often used in computational tests. Next, we briefly summarize some of these sets of test instances for the particular types of problems in the case of makespan minimization, where it is recommended to test algorithms to be developed in the future using these sets.

FLOW SHOP PROBLEMS

For $\gamma = C_{max}$, the following sets of instances have been mainly used:

Taillard [110]: This is a set of 120 instances including 10 instances for each format $(n,m) \in \{(20,5),(20,10),(20,20),(50,5),(50,10),$ $(50,20),(100,5),(100,10),(100,20),(200,10),(200,20),(500,20)\}$ (available e.g. from http://people.brunel.ac.uk/∼mastjjb/jeb/orlib/files/flowshop2.txt or http://www.lifl.fr/∼liefooga/benchmarks/benchmarks/index.html).

Reeves [94] This is a set of 126 instances including three types of problems each of them containing six instances for each of the formats $(n,m) \in \{(20,5),(20,10),(20,15),(30,10),(30,15),(50,10)(75,20)\}$ (available e.g. from http://people.brunel.ac.uk/∼mastjjb/jeb/orlib/files/flowshop1.txt).

Demirkol et al. [29] : This set contains 80 instances including 10 instances for each of the formats $(n,m) \in \{(20,15),(20,20),(30,15),(30,20),(40,15),(40,20),(50,15),(50,20)\}$. Note that there are also given 320 instances for the flow shop problem with the L_{max} criterion.

We only mention that e.g. for $\gamma = \sum T_i$, the following set of instances can be recommended for further tests:

Vallada et al. [115] : This is a set of 540 problems ranging from 50 to 350 jobs and from 10 to 30 machines (available from http://soa.iti.es).

JOB SHOP PROBLEMS

For job shop problems, there exist a large number of benchmark instances for the case of minimizing the makespan. In particular, the following sets of instances are often used in computer experiments for problems with $\gamma = C_{max}$:

Fisher and Thompson [35] : This is a set of three instances of the formats $(6,6),(10,10),(20,5)$. In particular, the famous $(10,10)$ instance has often been considered, and the optimal makespan value of 930 was only confirmed 25 years after publishing the data of this instance.

Lawrence [61] : This is a set of 40 instances including 5 instances of the formats $(n,m) \in \{(10,5),(15,5),(20,5),(10,10),(15,10),(20,10),(30,10),(15,15)\}$.

Applegate and Cook [8] : This is a set of 10 instances of the format $(n,m) = (10,10)$.

Storer et al. [107] : This is a set of 80 instances including 20 instances for each of the formats $(n,m) \in \{(20,10),(20,15),(50,10),(50,10)\}$.

Yamada and Nakano [128] : This set contains 4 instances of the format $(n,m) = (20,20)$.

Taillard [110] : This set contains 80 instances including 10 instances for each of the combinations $(n,m) \in \{(15,15),(20,15),(20,20),(30,15),(30,30),(50,15),(50,20),(100,20)\}$.

Demirkol et al. [29] : This is a set of 80 instances including 10 instances for each of the formats $(n,m) \in \{(20,15),(20,20),(30,15),(30,20),(40,15),(40,20),(50,15),(50,20)\}$. We note that also a set of 320 job shop instances for the L_{max} criterion has been given in [29].

The instances by Taillard [110] can be obtained e.g. from http://people.brunel.ac.uk/∼mastjjb/jeb/orlib/files/jobshop2.txt.
Accordingly, the instances from [8, 35, 61, 107, 128] are available from http://people.brunel.ac.uk/∼mastjjb/jeb/orlib/files/jobshop1.txt.

OPEN SHOP PROBLEMS

In the case of minimizing C_{max}, it turned out that square problems with $n = m$ appear to be the hardest problems. Therefore, in practically all papers dealing with makespan problems such instances are generated. Now, mostly the following types of open shop problems are used for an experimental comparison of algorithms:

Taillard [110] : This set of 60 open shop instances includes 10 instances for each of the formats $n = m \in \{4,5,7,10,15,20\}$.

Brucker et al. [14] : This is a set of originally 18 instances including six instances for each $n = m \in \{5,6,7\}$, later extended to nine instances for each of the values $n = m \in \{5,6,\ldots,9\}$.

Gueret & Prins [43] : This is a set of 160 hard instances including 20 instances for each size $n = m \in \{3,4,\ldots,10\}$.

The instances from [110] have often been used but they are easy in the sense that for many instances, the optimal objective function value is equal to the trivial lower bound:

$$LB = \max\left\{\max_{1\le i\le n}\sum_{j=1}^{m} p_{ij},\ \max_{1\le j\le m}\sum_{i=1}^{n} p_{ij}\right\}.$$

The idea of Brucker et al. [14] is to generate processing times such that the total processing time of each job and the machine loads are equal or close to this lower bound. It has been observed that instances with such a property often have an optimal makespan value

which is different from the above lower bound. Gueret and Prins [43] refined this strategy and generated even harder instances. They reported that there exist open shop instances with $n = m = 7$ which could not be solved to optimality on a SUN Sparc-5 workstation by existing branch and bound algorithm within 50 hours.

We only note that in the case of minimizing sum functions, it turned out that problems with $n >> m$ appear to be the hardest ones, while problems with $n < m$ and also with $n = m$ appear to be easier. A broad set of instances for the problem $O || \sum C_i$ was used in the following paper:

Bräsel et al. [13] : This is a set of instances for the formats (n,m) with $n \neq m$ and $n,m \in \{10,20,30,40,50\}$ and in addition with $n = m \in \{10,15,20,25,30,35,40\}$. For each format, this set includes 50 instances of type L (processing times from the interval [1,100]) and 50 instances of type S (processing times from the interval [1,20]). This yields a total of 2700 instances. These instances are based on the random number generator by Taillard [110]. A subset of these instances has also been used in [6], and the data of these instances have been extended to weighted and unweighted total tardiness problems in [7].

The random seeds for generating these instances can also be obtained from the website of the LiSA program package (http://lisa.uni-magdeburg.de, see also Section 8).

4 GENETIC ALGORITHMS

Genetic algorithms have been originally developed by Holland [47]. They work with a population composed of individuals. The genetic structure of a biological individual is composed of several chromosomes. Each of these chromosomes is composed of several genes each of which consists of a number of alleles. In combinatorial optimization, an individual is usually identified with a chromosome. A chromosome is further split into a number of genes (in some papers a gene is also identified with an allele). Before applying a genetic algorithm to scheduling problems, an appropriate encoding (or representation) of the solution must be introduced.

Representation of a solution

In many genetic algorithms, often a binary encoding is used, i.e., each gene of an individual contains either the number 0 or the number 1. Of course, integers can be converted to such a representation but for permutation problems, this is often not favorable. For flow shop problems with the permutation condition ($prmu \in \beta$), a standard way of representing a feasible solution is the *permutation code*, i.e., an individual consists of a string of length n, and the i-th gene contains the index of the job at position i, so an individual describes the *job sequence* chosen on all machines. In the case of a regular criterion, a permutation is decoded into a feasible solution by construction the resulting *semi-active* schedule.

For job shop scheduling problems, the situation is a bit different. Here several encoding strategies exist, and it is not clear in advance which is the best. Many authors distinguish between a *direct* and an *indirect* representation. In the first case, a solution is directly encoded into the *genotype* (i.e., the schedule itself is encoded), but in the second case not (this means that an indirect representation encodes the instructions to a schedule builder). Recently, Abdelmaguid [2] presented an alternative classification, which distinguishes be-

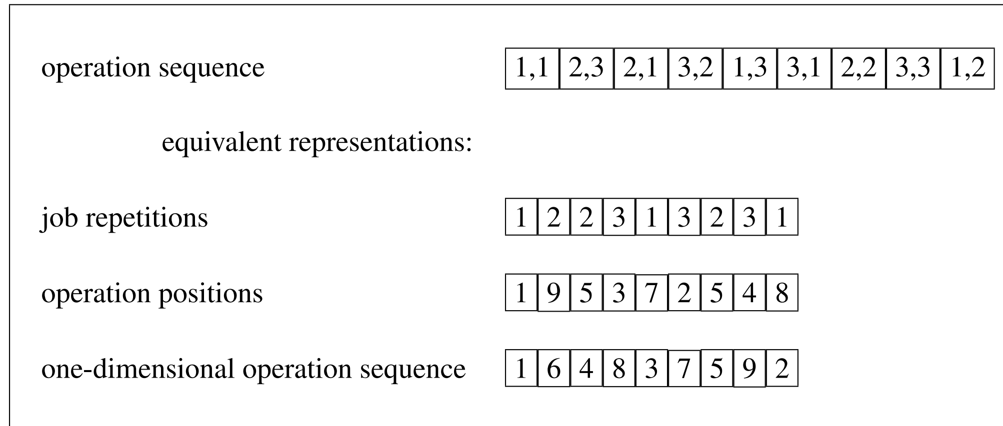

Figure 1. Operation-based representation

tween *model-based* and *algorithm-based* representations.

For the first type, the structure of the genotype is based on the decision variables introduced for a specific mathematical model and from the genotype, either a feasible or infeasible schedule can be directly obtained (in the latter case, an algorithm is required to transform an infeasible schedule into a feasible one). One possibility is to use the binary decision variables from the *disjunctive graph model*: all undirected arcs between operations on the same machine must be oriented in one of two possible ways which can be described by genes containing one of the numbers 0 or 1, where for the classical job shop problem, the length of the chromosome is $mn(n-1)/2$ and thus rather large. Moreover, there are several *processing sequence* representations. A straightforward generalization for job shop problems is to use the *operation code*. If each job consists of m operations, we have a chromosome composed of nm genes (i.e., each gene represents an operation (i, j)). An *operation-based* representation is illustrated in Fig. 1, where several variants are used. In the first variant, the chromosome is a sequence of the operations (i, j). From such a chromosome, we immediately get the job and machine orders according to the sequence of operations belonging to the same job or to the same machine. In the second variant using jobs with repetitions, the gene containing a particular job index i for the k-th time represents the k-th operation of job J_i according to the given technological route. Alternatively, the third and fourth variants refer to the subsequent numbering of the operations in the form $(1,1),(1,2),(1,3),(2,1),(2,2),(2,3),(3,1),(3,2),(3,3)$. In the representation describing the positions of the operations, the k-th gene contains the position of the k-th operation, e.g., number 5 in gene 3 means that the third operation (1,3) is at position 5. The one-dimensional operation sequence is similar to the first description, e.g., number 6 in gene 2 means that the 6-th operation $(2,3)$ is sequenced on position 2.

An alternative representation, which is very similar to an operation code, is the use of *random keys*, where a gene is filled with a randomly generated number between 0 and 1. By sorting these numbers, one obtains the corresponding sequence of operations. Another representation is the *preference list-based representation*, where a string of operations is used for any machine. This representation is illustrated in Fig. 2, where the chromosome characterizes the same job order on each machine as in the operation-based representation

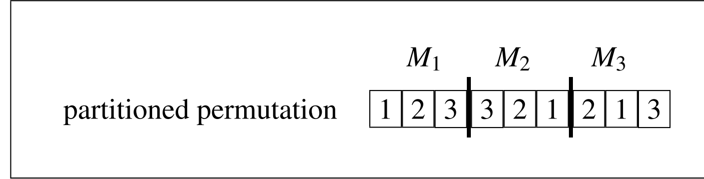

Figure 2. Preference list-based representation

in Fig. 1. However, this may lead to violations of one or several technological routes, which requires a *repairing algorithm* for transforming the current solution into a feasible one. A fourth processing sequence based representation is to describe the set of feasible job sequences on the machines by the *rank matrix* of the underlying graph. This representation excludes the redundancy contained in the operation code, namely that several strings may represent the same solution. There are several variants of decoding an individual into a feasible schedule. One possibility is the construction of the resulting *semi-active schedule* from a chromosome. However, often a decoding algorithm is used that transforms a chromosome into an *active schedule* (Giffer and Thompson or G & T algorithm [39]) or into a non-delay schedule (in this case, it is possible that an optimal schedule cannot be encoded since there may not exist an optimal non-delay schedule) schedule. For some encoding procedures a repairing method is necessary since e.g. not every chromosome can be encoded into an active schedule. In the *integer representation*, the genes are filled b y integers which describe the completion (or equivalently, the starting) times of all operations (which might cause many infeasible schedules). Note that for all regular optimization criteria (i.e., the minimization of a non-decreasing function in the completion times of the jobs) from the sequence of all jobs on the machines the corresponding semi-active schedule can be constructed in $O(n^2)$ time.

In *algorithm-based representations*, information is stored in the genes which can be used by a particular algorithm. Abdelmaguid [2] distinguished three variants of such a representation. In a *priority rule-based representation*, the string of a chromosome contains a number of priority dispatching rules which are used to sequence the operations e.g. by the Giffler and Thompson (G & T) algorithm [39] for generating active schedules. In this case, the length of the string is equal to the number of operations. In a *machine-based representation*, the chromosome is a string of the machines. This representation is applied when an algorithm treats the machine in a particular order (e.g. when incorporating the shifting bottleneck heuristic into a GA). In a *job-based representation*, the chromosome is a string of jobs of length n and using this order, an algorithm is used to schedule the operations of this job such that the technological constraints are satisfied. In algorithm-based representations, there is usually no guarantee that an optimal solution may be obtained. We only notice that e.g. operation-based, job-based, completion time based or random key representations belong to the direct representations while preference or priority list-based or disjunctive graph-based and machine-based representations belong to indirect representations.

For open shop problems, the described representations can also be used since the situation becomes easier in the sense that no imposed technological routes of the jobs have to be respected. In this way, one can immediately apply all the representations discussed for permutation or job shop problems also to open shop problems. As for the job shop problem,

the disadvantage of an operation-based sequence is the redundancy, i.e., different sequences of the operations may represent the same set of job and machine orders. We remind that a non-redundant coding by rank matrices was used e.g. in [6] and will be described more in detail in Section 7.

Next, we describe the major components of a GA (see e.g. [10] for an introductory overview of genetic algorithms or [10, 40, 77] for a description of elementary genetic algorithms). For describing the genetic operators, we give only a few comments for binary representations and focus then on the *permutation code*, which is used for many scheduling problems.

Initialization of the population

According to the chosen parameter *population size PS*, an initial population containing *PS* individuals is generated. Often these individuals are randomly generated, or a part of the individuals is generated by specific constructive heuristics for the problem under consideration.

Evaluation of the population

The evaluation of the current populations determines a *fitness value* for each individual. This fitness value is related to the objective function value of an individual. Since a maximal fitness is preferred, a maximization problem is typically assumed. Since most shop scheduling problems are minimization problems, the objective function value is usually transformed, e.g., the fitness $FIT(i)$ of an individual i can be determined as

$$FIT(i) = \begin{cases} \overline{F} - F_i(S_i), & \text{if } F_i(S_i) < \overline{F} \\ 0, & \text{otherwise} \end{cases}$$

where $F_i(S_i)$ denotes the objective function value of the schedule S_i resulting from the individual i and $\overline{F}$ denotes the objective function value of some heuristic solution.

Since the function values considered are usually positive, sometimes also

$$FIT(i) = \frac{1}{F_i(S_i)}$$

is used as the fitness measure. However, some papers use directly the function value for characterizing the solution quality of an individual so that the individual with the smallest function value has the best fitness.

Selection of individuals

The standard procedure is a *fitness-proportional selection* (or *roulette wheel selection*) of individuals to produce the offspring. The probability $P(i)$ of selecting the i-th individual is given by

$$P(i) = \frac{FIT(i)}{\sum_{k=1}^{PS} FIT(k)}.$$

This means that each individual is interpreted as a segment of the wheel such that the size of this segment is proportional to the fitness. Of course, one can also use other *ranking procedures* for selecting preferably individuals with a larger fitness.

An alternative selection method is *tournament selection*. In the case of a *binary tournament*, two individuals are randomly chosen and that with the larger fitness is chosen

as parent. This procedure is repeated as long as parents have to be chosen. One can also use larger tournaments, e.g. k individuals are randomly chosen from which that with the best fitness is selected. In several papers, individuals are completely randomly chosen as parents.

In general, by means of two chosen parents, one or two offspring can be generated.

Application of Crossover

Typically, the *crossover* is the main operator applied in genetic algorithms which is in contrast to evolution strategies. By crossover, the genetic structure of two selected individuals of the current population is mixed. The parameter P_C denotes the *crossover rate* and gives the probability of applying a crossover when generating an (or two) offspring from the selected parents. In many papers, it is recommended to use $P_C \geq 0.6$. Here we start with typical crossovers for a binary representation and describe then only the standard crossover operations for permutation problems (for some refinements of these operators, see the description of the particular papers in Sections 5 - 7).

Many genetic algorithms encode a solution as a bit string. This means that an individual is composed of a number of genes, where each gene has a 0 or 1. There exist an *n-point crossover* and a *uniform crossover*. The two most often used variants of an n-point crossover are the *one-point* crossover and the *two-point* crossover. These crossovers are illustrated in Fig. 3 and Fig. 4. In the case of a one-point crossover, one cut point is determined and for creating two offspring, the genes of the two parents after the cut point are interchanged. In the case of a two-point crossover, the genes between the first and second cut points are interchanged to generate two offspring. To describe such crossovers, one can use the notation (i, j)-crossover to indicate that the segment from the i-th to the j-th gene of the two chromosomes are interchanged (in such a way, a (i,n)-crossover characterizes a one-point crossover, if the length of the string is n). Thus, the one-point crossover in Fig. 3 is a (5,9)-crossover, and the two-point crossover in Fig. 4 is a (3,5)-crossover.

In the case of a uniform crossover, a bit mask of the numbers 0 and 1 is randomly generated, and for producing the offspring genes from the first parent are taken if the corresponding component of the mask includes 0, otherwise the gene from the second parent is used (and if a second offspring is generated, it is opposite to the first one). The uniform crossover is illustrated in Fig. 5. We note that a uniform crossover may also be a applied in a *parametrized* form, where a probability for taking the gene e.g. from the first parent is used.

Of course, one can code an integer as a bit string but for sequencing and scheduling problems often other codes are used. For *permutation problems*, where the set of solutions is characterized by the set of all (or a subset of all) permutations of n jobs often the *job* or *permutation code* is used. An individual (solution) is described by a job sequence, and the i-th gene contains the index of the job on position i.

However, when applying a classical crossover operator to individuals described in a permutation code, the offspring are often infeasible. There have been developed several operators that repair illegal offspring so that all offspring describe feasible individuals. The most popular and standard crossover operators are the PMX, the OX, the LOX, the CX operator, the OBX operator and the PBX operator. Most of these operators are two-point crossovers. We describe them in the terminology of the job code and generate one offspring

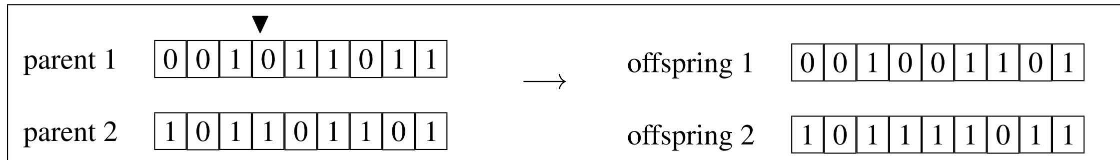

Figure 3. One-point crossover in bit strings

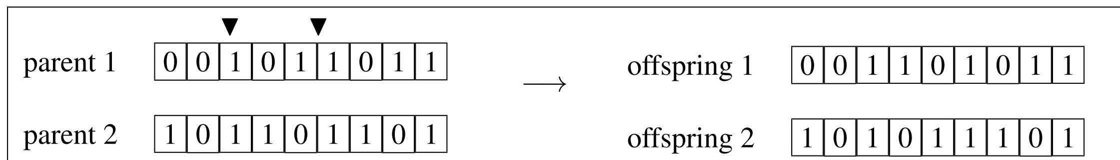

Figure 4. Two-point crossover in bit strings

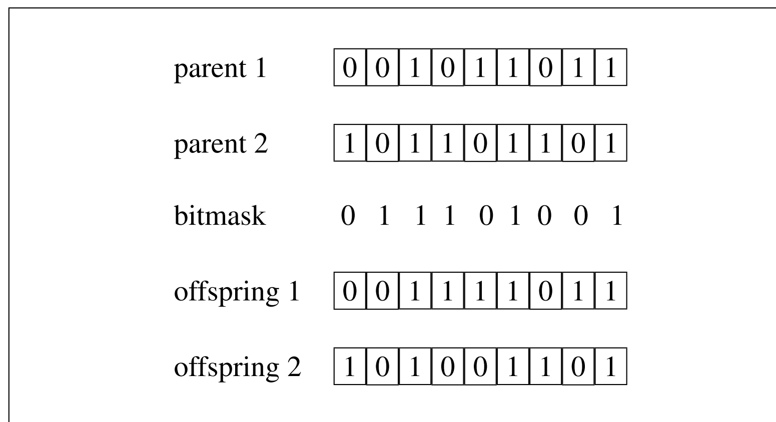

Figure 5. Uniform crossover in bit strings

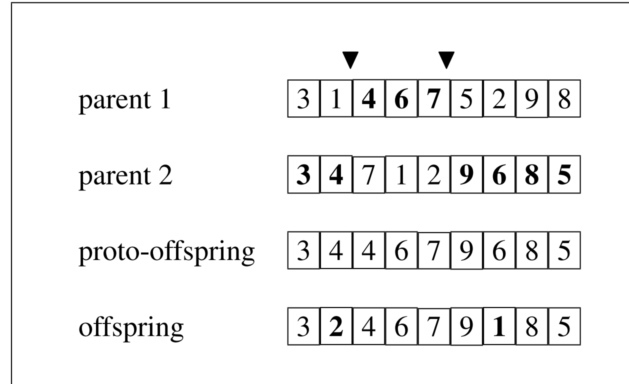

Figure 6. PMX crossover

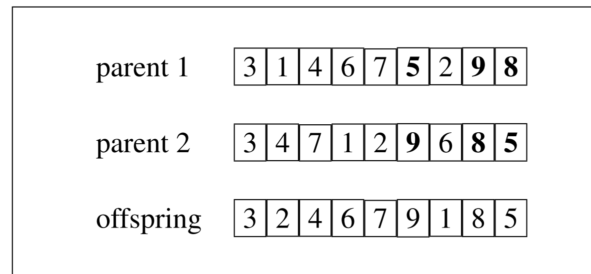

Figure 7. CX crossover

(if two offsprings are generated from the two parents, the second offspring is obtained by interchanging the role of the parents; note also that in the following we refer to the generation of the second offspring according to the previous description if not mentioned differently).

The PMX (*partially mapped crossover*) determines randomly two cut points and for generating an offspring, the segment between these cut points is taken from the first parent and it replaces this part in the second parent. The resulting proto-offspring is usually infeasible. Then the inverse replacement is applied to the job indices occurring a second time outside the chosen segment. The PMX operator tries to keep the positions of the jobs in the offspring when copying them from the parents. The number of jobs occurring at different positions is at most equal to the number of jobs between the cut points. A (3,5)-PMX crossover is illustrated in Fig. 6. The indices 4 and 6 occur twice. To repair them, we use the mappings from parent 1 to parent 2 (may be, subsequently applied): 4 is mapped to 7, and 7 is mapped to 2, so 2 replaces 4 in gene 2; 6 is mapped to 1, so 1 replaces 6 in gene 7.

The CX (*cycle crossover*) determines a subset of jobs occupying the same set of positions in both parents. These jobs are copied from the first parent into the offspring, while the remaining positions are filled with the jobs from the second parent. The CX crossover is illustrated in Fig. 7, where genes 6,8 and 9 with the indices 5, 8 and 9 have been selected. Both the PMX and the CX crossovers try to maintain the absolute positions of the jobs. The subsequent crossover operators focus on the relative order of the jobs.

The *order preserving one-point crossover* determines a cut point and selects the segment

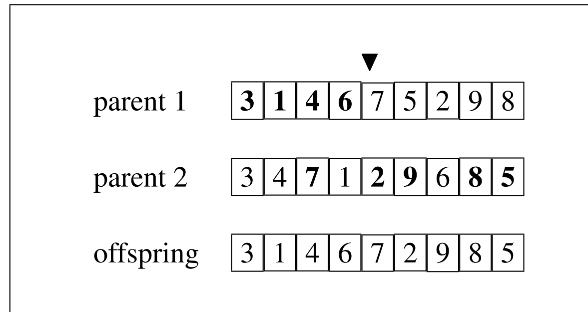

Figure 8. Order preserving one-point crossover

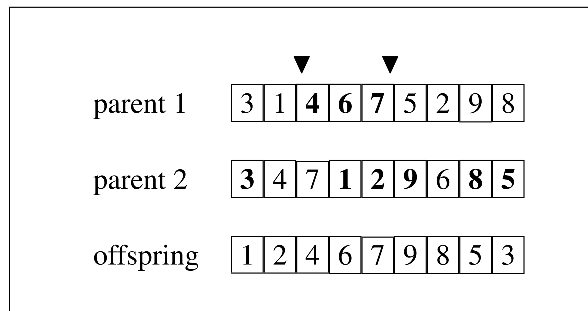

Figure 9. OX crossover

until the cut point from the first parent while the missing indices are then inserted after the cut point in the relative order of the second parent. A crossover of this type with the cut point after the fourth gene is illustrated in Fig. 8. In our previous terminology, this corresponds to the selection of the first offspring when performing a (5,9)-crossover (equivalently, this can be interpreted as the second offspring when performing a (1,4)-crossover).

The OX (*order crossover*, see [80]), originally designed for the TSP, determines randomly two cut points. The segment between these two points is taken from the first parent and copied into the offspring. Then the empty genes are filled with the missing jobs in the relative order of the second parent starting after the second cut point. A $(3,5)$-OX crossover is illustrated in Fig. 9.

The LOX (*linear order crossover*, see [121], Definition 3.6 or [33]) is a slight modification of the OX operator. The procedure is very similar with the exception that it does not consider chromosomes as cyclic. This means that after copying the segment between the two cut points from the first parent into the offspring, the empty genes in the offspring are filled by the missing jobs maintaining the relative order of the second parent scanning from left to right. A (3,5)-LOX crossover is illustrated in Fig. 10. Note also that in some papers is not clearly distinguished between the OX/LOX operators.

The OBX (*order-based crossover*, see [109]) selects a subset of jobs in the first parent. These jobs are copied in the same relative order into the offspring at the positions of these jobs in the second parent. Then the empty positions in the offspring are filled by the jobs from the second parent at these positions. The OBX crossover is illustrated in Fig. 11,

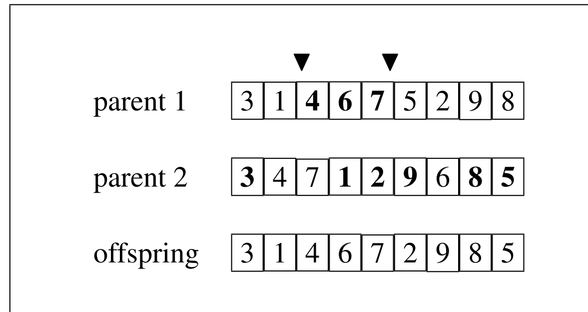

Figure 10. LOX crossover

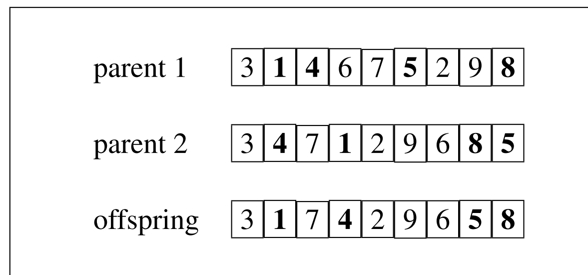

Figure 11. OBX crossover

where the indices 1, 4, 5 and 8 have been chosen.

The PBX (*position-based crossover*, see [109]) selects a subset of positions in the first parent and copies the jobs at these positions into the offspring. The remaining positions are filled with the missing jobs appearing in the same relative order as in the second parent. This crossover is illustrated in Fig. 12, the indices 1, 6, 2 and 9 from the positions 2, 4, 7 and 8 in the first parent have been chosen. The notation position-based is a bit misleading because the relative order from the second parent is chosen. Sometimes, this crossover is also denoted as *uniform order-based crossover* which characterizes better how this crossover works.

If crossover is not applied, the generation of offspring is done only by means of *mutations*.

Application of Mutation

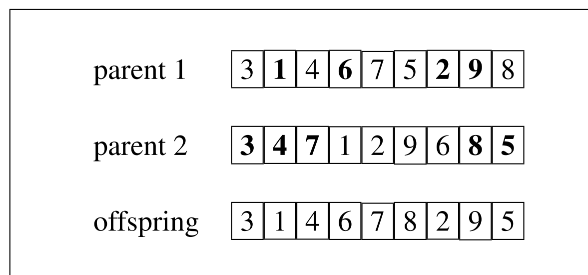

Figure 12. PBX crossover

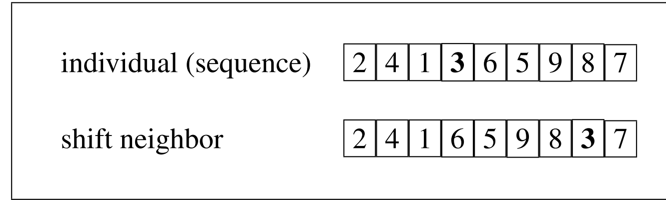

Figure 13. Shift mutation

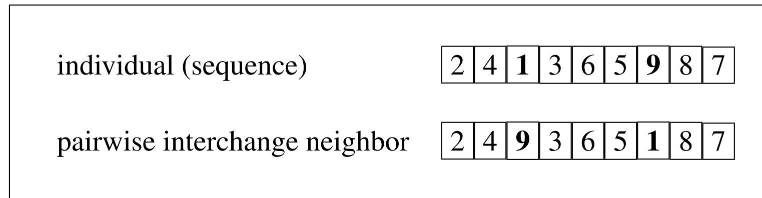

Figure 14. Pairwise Interchange mutation

In genetic algorithms mutation can be considered as a background operator. The use of a mutation is also controlled by a *probability parameter*, called the *mutation rate* (or probability) P_M. In case of a binary coding, it represents the probability of changing a particular gene from 0 to 1 or vice versa. This mutation rate is usually small (often it is recommended $0.01 \leq P_M \leq 0.1$). If a permutation coding is used, a mutation usually corresponds to the generation of a neighbor in some neighborhood. In this case, the mutation denotes the probability of applying a mutation to the current individual. Note that after the selection of two parents and possibly applying a crossover to them, one applies a mutation independently to each intermediate offspring when generating two offspring. Here the mutation rate is typically larger.

For permutation problems, where the set of feasible solutions is described by the set of $n!$ permutations of the n jobs, one can use e.g. the *shift mutation* or neighborhood (an arbitrary job in the current sequence is selected and shifted to an arbitrary but different position - see Fig. 13, i.e., there are $(n-1)^2$ neighbors of each sequence; often also referred to as *insertion neighborhood*) or the *pairwise interchange neighborhood* (two arbitrary jobs are selected and interchanged in the sequence - see Fig. 14, i.e., there are $n(n-1)/2$ neighbors of each sequence, often also referred to as *swap neighborhood*). A sub-neighborhood is the *API neighborhood*, where only adjacent jobs may be interchanged to generate a neighbor. Thus, there exist $n-1$ neighbors of a sequence. Finally, in the *inversion neighborhood*, a neighbor is generated by choosing a segment of jobs and reinserting it in the opposite sequence - see Fig. 15. These types of mutations are also denoted as shift, pairwise interchange, API or inversion operator. Some theoretical properties of these neighborhoods for such permutation problems as e.g. the diameter of the underlying graphs have been derived in [123, 124].

For problems of minimizing the makespan, the above neighborhoods are often restricted further by considering only neighbors satisfying a *necessary condition* for an objective function value improvement. This is done by considering the *critical path* of a solution in the underlying graph and forming *blocks* of such operations, each block containing at least two operations to be processed subsequently on the same machine. In the case of

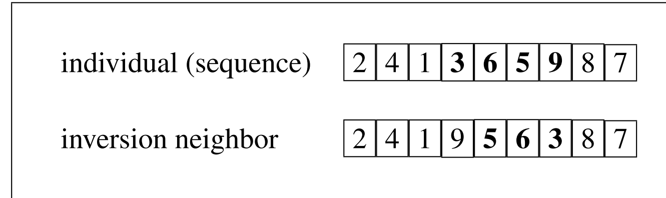

Figure 15. Inversion mutation

the API neighborhood, only the first two and the last two operations of a block need to be considered (in all other cases, for an API neighbor, there would exist a path containing the same operations as the critical path of the starting solution. A similar restriction can be applied to the other neighborhoods, e.g. in the shift neighborhood, a neighbor with a better function value may only be obtained by shifting a job with an operation in such a block to the left (i.e., at least before the first job of this block) or to the right (i.e., at least after the last job of this block). We denote these restricted neighborhoods as crit-API, crit-shift, etc.

Formation of the new population

In the first applications, the new population was always formed by offspring (e.g. those with the largest fitness). A modification is the *elitist strategy* which ensures that the individual with the best fitness (or a specific percentage of best individuals) is always included into the next generation. In several genetic algorithms, the new populations may contain both selected parents and created offspring. Such a situation occurs e.g. in the case of *2/4 selection* which means that two parents are selected and two offspring are generated, and in the new population, the two parents are replaced by the two individuals with the largest fitness, where both the parents and offspring can survive. An alternative are *steady state* algorithms, where the worst individual are systematically replaced by offspring (i.e., the offspring do not directly replace their parents).

Stopping criteria

Often a maximal number *NGEN* of generations or a maximal time limit are imposed. Alternatively, one can use a termination criterion such that the algorithm stops if for a specific number of generations, no improvements in the best objective function value have been obtained, or if the relative deviation of the best objective function value from a given lower bound does not exceed a particular value.

In particular for the job shop problem, it has been observed that GA converge rather slowly. For this reason, often *hybrid genetic algorithms* are used. This means that a GA is combined with some other heuristic algorithms, often with a local search procedure applied to the offspring generated by crossover and/or mutation. Such a procedure is often denoted as *genetic local search* (i.e., a population based local search) or *memetic search*. Another way of hybridizing a GA is the use of special heuristics for constructing the initial population or the incorporation of particular heuristics into the evaluation function for decoding chromosomes into feasible schedules. In this sense, practically all recently developed algorithms are a hybrid algorithm, but most papers concentrate on the incorporation of advanced local search techniques into a GA.

Most genetic algorithms developed in the literature work with *one* population. However, there have been also developed *parallel* genetic algorithms of different types. We discuss some of these aspects when dealing with the corresponding papers.

In the following three sections, we survey existing genetic algorithms for the three major types of shop scheduling problems. Here, we focus only on the most relevant work including detailed computational tests. In [100], it has been analyzed for the flexible flow shop problem, which objective functions have been considered in the literature: 60 % of the papers consider the minimization of the makespan C_{max}, while 11 % of the papers consider the minimization of total or average (weighted or unweighted) completion time. On the other side, only 2 % of the papers deal with multi-criteria problems and even only 1 % deal with earliness-tardiness criteria. Among the problem classes considered, most genetic algorithms have been proposed for the flow and job shop problems with the makespan criterion. In view of the huge number of papers on this subject, a selection had to be made which of the results are included into this survey. In the following, for flow and job shop problems, we review first the results for the makespan criterion and then for other single criterion problems as well as some extensions of the classical problems while at the end of each section, few comments on algorithms for multi-criteria problems are given. Hybrid or flexible flow and job shop scheduling problems are considered in a separate subsection. For the open shop problem, where almost all papers deal with the C_{max} criterion, we do not make this distinction.

5 FLOW SHOP PROBLEMS

First, we deal with some algorithms for the problem $F|prmu|C_{max}$. Some of the earliest applications of evolutionary algorithms in the scheduling area have been given for flow shop problems (see [1,121,122]). These algorithms contain also components of evolution strategies. We describe here the algorithms originally given by Werner in [121], which has also been published later in a journal paper (see [122]). Since the main ideas from these works have never been published in English, we describe the major components of this algorithm more in detail here.
In contrast to the genetic algorithms developed for scheduling problems later, the algorithm from [121] uses individuals which are composed of two chromosomes, namely a *dominant* and a *recessive* chromosome. Each chromosome is a permutation of the jobs (indices). For generating an offspring, two parent individuals $P^1 = (p^{1D}, p^{1R})$ and $P^2 = (p^{2D}, p^{2R})$ are chosen, where the superscript D denotes the dominant and R denotes the recessive chromosome. The dominant chromosome is the one with the better fitness (i.e., with the lower objective function value). For generating one offspring from the chosen parents, one chromosome is chosen from any parent, e.g., $P^{OFF} = (p^{1D}, p^{2R})$ (any of the four combinations is possible). The underlying idea is to prevent that, after some generations, the current population is composed of very similar individuals. The used crossover (see [121], Definition 3.6) is exactly the operator later denoted as the LOX operator. The mutations are mainly based on performing subsequently several shifts of particular jobs. This process is controlled by a parameter summing up the distances of the individual moves in the shift neighborhood. Here the distance is measured as the number of pairs of jobs which are in two permutations in opposite order. In addition, also inversion mutations and pairwise interchanges of blocks

of subsequent jobs (which are a generalization of the pairwise interchange operator) are possible. Note that according to the current probabilities for applying a specific mutation, several types of mutations can be subsequently applied to a proto-offspring. In this sense, the realization of a mutation corresponds to a restricted *variable neighborhood search*.
As in evolution strategies, Werner [121] used a strategy population controlling the generation of individuals. A *strategy individual* is also composed of two strategy chromosomes, where the concept of dominant and recessive chromosomes has also been applied. In [121], such a strategy chromosome consists of seven genes. The first gene gives the probability for selecting the dominant chromosome of an individual in the permutation population when generation an offspring. Genes 2 - 6 control the application of the different types of mutations described above. The last gene gives the probability for applying the LOX crossover. Contrary to most genetic algorithms, these probability parameters are variable and will be adapted during the search.
Werner applied several selection variants for forming the next generation. First, an offspring may replace the parent with the lower fitness if it an improvement (i.e., the function value of the dominant chromosome of the offspring is better than the function value of the dominant chromosome of the parent with lower fitness; selection of first type). In addition, the offspring may also replace the parent with lower fitness, if its fitness is lower than that of this parent. This is done in a similar way as in simulated annealing algorithms (selection of second type), which may help to increase diversity further. Based on initial tests, an individual in the strategy population can survive only for a restricted number of generations, while this does not hold for the permutation population.
This algorithm also considered several variants of a parallel GA working with different subpopulations. By means of an *isolation* parameter, it is controlled whether after a specific number of generations, an exchange of individuals in adjacent sub-populations is possible or not.
Both the permutation and strategy populations contained 12 individuals (i.e., 24 permutations and strategy variants). The parameter *NGEN* has been fixed in such a way that all tested variants approximately generate 24000 individuals. The algorithm stops when the given maximal number of generations has been constructed, a time limit is elapsed or for a given number of generations, no improvement of the fitness of the best individual has been obtained. In the computational tests, many variants have been tested and compared to other stochastic search procedures on randomly generated instances with up to $n = 105$ jobs and $m = 30$ machines.

Reeves [94] presented a GA which uses an initial population containing the job sequence obtained by the NEH heuristic [74], while the remaining individuals are randomly generated. The GA applied a simple ranking procedure to select the parents. The crossover finally used is an order-based one-point crossover. Based on initial tests, the shift neighborhood with an adaptive mutation rate was used in the mutation step. In addition to own test instances, the algorithm developed has been compared on Taillard's instances [110] with a simulated annealing algorithm and a simple neighborhood search procedure. The GA was superior to the neighborhood search procedure for most problems and obtained similar results as simulated annealing, where the GA was better for the large problems (and obtained its final solution more quickly).

Chen et al. [23] used several constructive heuristics for generating the initial popula-

tion. The first $m-1$ individuals are those generated by the heuristic given by Campbell et al. [18], the m-th individual results from the rapid access heuristic by Dannenbring [26], and the remaining individuals are generated by one pairwise exchange of two jobs in an individual already generated. Parents are selected proportional to their fitness. After initial tests, this algorithm uses the PMX crossover with $P_C = 1.0$, but it does not use mutation at all. The algorithm has been run with $PS = 60$ and $NGEN = 20$.
The algorithm has been compared with two existing heuristics on 200 own randomly generated instances for 20 combinations (n,m) with $n \in \{7,10,15,25\}$ and $m \in \{4,5,8,10,15\}$. The GA developed turned out to be superior to the other two heuristics.

Murata et al. [72] used the permutation code and tested three different selection probabilities for choosing the parents. They also tested 10 crossover operators, among them a one-point crossover and three variants of a two-point crossover were preferred. All these operators are of the order-preserving type for filling the empty positions of the offspring by the genes of the second parent. For a mutation, the shift, the pairwise interchange, the API neighborhoods as well as a generalization, where three jobs can arbitrarily change their positions, have been tested. The elitist strategy is used by passing the best individual directly to the next generation. The stopping criterion was a maximal number of fitness evaluations. The computational tests have been made on 100 randomly generated instances with $n = 20$ and $m = 10$. After detailed tests, $PS = 10$, a variant of the two-point crossover with $P_C = 1.0$ and the shift mutation with $P_M = 1.0$ have been recommended. Then the authors compared their GA with tabu seach, simulated annealing, genetic local search and genetic simulated annealing. In their tests, genetic local search and genetic simulated annealing worked best.

Reeves and Yamada [95] improved the algorithm from [94] by suggesting a new crossover operator denoted as MSXF (*multi-step crossover fusion*) and introducing path relinking. The MSXF operator (originally given for the job shop problem in [130], see also Chapter 6) can be interpreted as a short term local search starting from one of the two chosen parents and directs the search towards the other parent. This operator uses a neighborhood and a strategy for deciding whether a move to a neighbor is accepted or not. The neighborhood applied is the crit-shift neighborhood using the block structure of a critical path of a solution. Mutation is only applied when parents are too close to each other. For this purpose, a *multi-step mutation fusion* (MSMF) operator has been introduced.
This new algorithm has been tested on the most challenging Taillard instances [110] with $m = 20$ machines. Several best known upper bounds for these instances were improved by this algorithm.

Etiler et al. [32] presented a GA that uses the LOX crossover operator with $P_C = 1.0$ and the shift mutation with a probability of $P_M = 0.05$. The population size is 60 and the number of generations is limited to $NGEN = 20$. The initial population is generated by means of the $m-1$ solutions constructed by the algorithm by Campbell et al. [18] and one solution by the Dannenbring [26] heuristic, while the other individuals are obtained by applying a pairwise interchange mutation to one of the individuals generated by the constructive algorithms.
This algorithm has been tested mainly against the NEH heuristic and the GA by Chen et al. [23] on own randomly generated instances in the range from $(n,m) = (5,5)$ until $(40,40)$. It has been found that this algorithm produces better results than the GA by Chen et al. [23] which results mainly from the superiority of the LOX operator over the PMX operator.

Ponnambalam et al. [85] compared constructive and iterative heuristics for the

flow shop problem. In particular, in Section 7 of their paper, they presented a GA. The initial population is randomly generated. The algorithm uses the *generalized position crossover* (GPMX, see also [12]) and the pairwise interchange mutation. In initial tests, the author experimented with $PS \in \{5,10,\ldots,35,40\}, P_C \in \{0.4,0.5,\ldots,0.9\}, P_M \in \{0.02,0.03,\ldots,0.09\}$ and $NGEN \in \{100,150,\ldots,350\}$. Then they fixed $PS = 20, P_C = 0.60, P_M = 0.05$ and $NGEN = 250$. 21 test instances with up to $n = 30$ jobs and $m = 10$ machines have been taken from the OR library found under http://www.people.brunel.ac.uk/∼mastjjb/jeb/info.html. This GA has been compared with several other heuristics, e.g., the NEH [74], the CDS [18] and Dannenbring's [26] heuristics. The GA turned out to be superior to the constructive heuristics, but only in 11 of 21 instances, the minimum makespan was received by the GA.

Chang et al. [21] combined a GA with a mutation-based local search. First, they presented a simple GA which starts with a randomly generated population using the permutation code for representing an individual. For choosing the parents, tournament selection has been chosen. A one-point crossover and the pairwise interchange mutation are applied for generating the offspring. The elitist strategy is used, where 10 % of the best individuals are directly passed to the next generation. Then they presented a refined hybrid algorithm. First, one individual obtained by the NEH [74] heuristic is included into the initial population. Then the mutation operator is extended to a restricted local search in the pairwise interchange neighborhood, where the number of applied mutations to the proto-offspring is equal to $nm/10$. The algorithm has been run with $PS = 50, P_C = 0.95$ and $P_M = 0.7$.
Both variants, i.e., the simple and the hybrid GA, have been compared with the NEH heuristic on a subset of 18 benchmark instances from [110]. The hybrid algorithm turned out to be superior to both the NEH and the simple GA.

Ruiz et et al. [99] suggested two genetic algorithms for the problem $F \mid prmu \mid C_{max}$. The initial population creates one individual by applying the NEH [74] insertion heuristic and the remaining individuals are either generated by a modification of this algorithm or randomly. The core of this paper is the development of four new crossover operators. The two best of them, denoted as SBOX and SB2OX, are as follows. The operator SBOX (*similar block order crossover*) first copies blocks of common jobs contained in both parents into the offspring, where a block consists of two consecutive genes. Then a cut point is determined and jobs up to this point are taken from the first parent, while the missing jobs are copied into the empty genes in the relative order of the other parent. This crossover is illustrated in Fig. 16, where one block of two subsequent genes is at the beginning and two such blocks are at the end of both parents. The operator SB2OX (*similar block two-point order crossover*) is very similar to SBOX but in the second step, two cut points are determined and the empty positions are filled as for a two-point crossover of the LOX type.
The second novelty is the *restart scheme* used in the GA which should avoid that the diversity within the population becomes too low. If for a specific number of generations the best makespan value has not been improved, the individuals in the current population are sorted according to non-decreasing order of the C_{max} values. Only the best 20 % of the individuals are left in the current population while the remaining 80 % are replaced by modified individuals. In particular, among the latter 80 % of individuals, 50 % of them are replaced by performing shift mutations in the best 20 % of individuals, 25 % are generated by a modified NEH heuristic and the other 25 % are replaced by new randomly generated individuals.

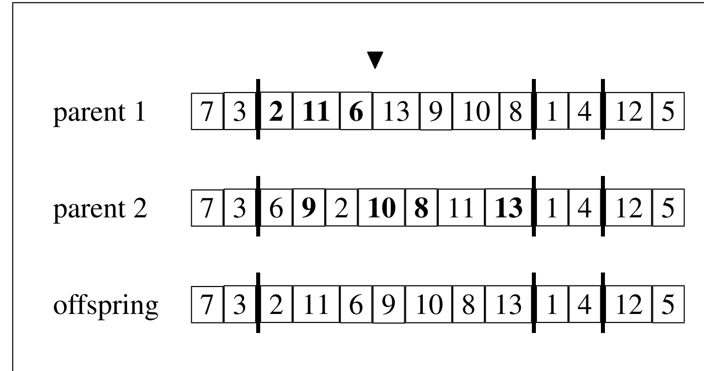

Figure 16. SBOX crossover

This is a similar strategy as used in *iterated local search* algorithms, where one tries to escape from a local optimum by disturbing the current solution (often denoted as a kick) so that after this perturbation step, local search in a smaller neighborhood is continued (for the application of this strategy to scheduling problems, see e.g. [15, 16]). The algorithm is hybridized by incorporating a local search in the shift neighborhood to the best individual in the population.
After a detailed calibration of the algorithm, tournament selection has been chosen, the SBOX operator, which was superior to seven other operators, was applied with $P_C = 0.4$, and the shift mutation was applied with $P_M = 0.01$. Moreover, they used $PS = 20$ and the restart procedure if no improvement has been obtained over 25 generations. The two new GA (denoted as GA-RMA and HGA-RMA for the hybrid variant) have been compared with 11 other algorithms, among them genetic (e.g. [3,72,94], simulated annealing, iterated local search and ant colony algorithms, on the 120 benchmark instances from [110]. The stopping criterion was a given maximum elapsed CPU time for all algorithms depending on the product *nm*, where three different proportionality factors were used. For all (short, medium and longer) runs, the new hybrid algorithm turned out to be the best algorithm.

Rajkumar and Shahabudeen [91] presented an improved GA (denoted as IGA). This algorithm includes the solution obtained by the NEH heuristic [74], while the remaining individuals of the initial population are randomly generated. It uses both roulette wheel selection and tournament selection for choosing the parents. Several crossover types are used: a two-point crossover, the PMX, the SJOX (similar job order crossover) and the LOX crossover. Three types of a mutation are considered: a three-job change (i.e., three jobs are selected and arbitrarily reinserted at these positions), a pairwise interchange and a shift mutation. The selection of a particular type of a crossover or a specific type of mutation is controlled by probability parameters. In addition, a restart scheme is applied to increase the diversity in the population. In particular, if the best fitness is not improved over a certain number of generations, the probability of a mutation is increased, and if the makespan is not converging for the current generation, 75 % of the individuals are replaced by randomly generated new individuals.
The algorithm has been run with $PS = 70, P_C = 0.9, P_M = 0.4$ and $NGEN = 10000$. This IGA has been tested against the NEH heuristic, a simple GA, a modified evolutionary programming and an existing hybrid heuristic on 29 benchmark instances, among

them 21 instances from [94]. IGA turned out to be the best algorithm among the tested ones.

We now consider some extensions of the classical permutation flow shop problem dealing with other single criterion problems or including additional processing constraints. All of the subsequent algorithms consider the permutation condition $\beta = prmu$.

Janiak and Portmann [49] presented a GA for the makespan minimization problem with controllable processing times, where these times linearly depend on the use of a resource. Individuals are represented in the permutation code and for an individual, the optimal resource allocation (and thus the processing times) are found by a modified algorithm from the literature. The initial population is formed by applying modifications of the algorithms by Campbell et al. [18], Dannenbring [26] and Nawaz et al. [74]. The fitness of an individual is defined as the maximal makespan value of the current population minus the makespan of the individual considered. As the crossover operator, a one-point crossover and a specific crossover based on the LOX operator but taking into account the properties of the block approach for makespan problems are used. As the mutation operator, a pairwise interchange mutation and a shift mutation that move an operation from the critical path outside this block (i.e., the generation of a neighbor in the crit-shift neighborhood) are used.
The algorithm stops after generating $NGEN = 50$ generations but nothing is set about the value PS. This algorithm has been tested on two rather small problem sizes $(n,m) = (10,10)$ and probably $(n,m) = (20,5)$ (although declared with opposite n and m) combined with five families of operation resource requirements and two groups of operation models. It turned out that the combination of the specific crossover operation using the structural properties with the classical pairwise interchange mutation worked best, however, the improvements over the best makespan value in the initial population are rather small.

A GA for the problem $F|prmu|\sum C_i$ of minimizing mean flow time has been given by Tang and Liu [111]. The solutions are encoded by the permutation code. The initial population is randomly generated with a population size $PS = 80$. However, before the GA starts, one-fifteenth of the population is improved by simple local search. Parents are chosen proportional to their fitness, but each individual is selected at most twice in each generation. The authors applied the PMX crossover with a crossover rate $P_C = 0.99$ and the shift mutation with a mutation rate $P_M = 0.12$. The maximum number of generations was settled as $NGEN = 1000$.
The performance was tested on randomly generated instances with five job levels ranging from $n = 50$ to $n = 150$ and four machine levels ranging from $m = 5$ to $m = 20$. The authors claimed that they obtained substantial improvements over a standard GA, and the results were also better than those by the algorithms from Rajendran [90] and Ho and Chang [46] although the GA required longer computational times.

Aldowaisan and Allahverdi [3] suggested heuristic algorithms including several GA for the problem $F|r_i \geq 0, prmu, no-wait\ ST_{sd} \mid C_{max}$. The initial population uses solutions generated by specific constructive procedures, some solutions obtained by pairwise interchanges from the previously generated solutions, and some randomly generated solutions. The population size is chosen as $PS = 95$. The selection of parents is done by roulette wheel selection. For the crossover, the PMX operator is applied with a probability $P_C = 0.725$. As the mutation operator, the pairwise interchange neighborhood is used with a probability

$P_M = 0.009$. The algorithm uses the elitist strategy for passing the best individual to the next generation, but all other individuals of the new generation are the offspring. The procedure stops after having generated $NGEN = 60$ generations.
Aldowaisan and Allahverdi [3] used three GA: the algorithm by Chen et al. [23] adapted to the no-wait problem (GA) and two variants of incorporating local search. The first of these variants (GA-1) incorporated the strategy of the NEH [74] heuristic, and the second one (GA-2) applied in addition local search in the pairwise interchange neighborhood. The algorithms have been tested on own randomly generated instances of the format (n,m) with $n \in \{40,60,80,100,120\}$ and $m \in \{5,10,15,20\}$. As a result, the best genetic algorithm (GA-2) has approximately the same solution quality as the best simulated annealing algorithm (denoted as SA-2)

Franca et al. [36] also considered the problem $F|r_i \geq 0, prmu, no-wait|C_{max}$. This algorithm addresses a new hierarchically organized complete ternary tree of 13 nodes to represent the population. Each node of this tree represents an agent that contains a pocket solution and a current solution. While the pocket solution acts like a long-term memory keeping track of the best individual in the agent, the current solution represents the offspring resulting from a recombination that may involve current and pocket solutions from other agents as well. This structure of the tree, resulting in four sub-populations, turned out to work best. The algorithm tested several crossover operators, among which the PMX operator turned out to be the best (which is probably caused by the relationship to the traveling salesman problem). To each generated offspring, a local search method is applied known as *recursive arc insertion*. It recursively inserts arcs in points to be considered as critical in the solution and is related to the 3-opt neighborhood for the TSP (which has been tested against other local search procedures). To maintain diversity, only pocket solutions with different fitness values are inserted into the population.
This algorithm has been tested on randomly generated instances with $n \in \{10,50,100\}$ and $m \in \{2,5,10\}$ as well as instances from TSPLIB with known optimal solutions. The new algorithm was superior to the best insertion heuristic in terms of the percentage deviation from the lower bound. For the problems with known optimal solutions, the new algorithm was able to obtain an optimal solution for all but two instances (in 10 runs for each instance).

Chen et al. [22] presented a hybrid algorithm for the re-entrant flow-shop scheduling problem. They used an operation-based representation of a solution described by permutations with repetition. The algorithm uses two sub-populations. The initial population is determined by choosing one specific heuristic for generating an individual in each subpopulation, while the other individuals are constructed by pairwise interchanges from these heuristic solutions. Parents are chosen according to roulette wheel selection. The authors applied a two-point crossover of the LOX type. For a mutation (denoted as neighborhood search operator), k genes (illustrated for $k = 3$ in the paper) are selected and the jobs at these positions can be arbitrarily re-arranged from which the best resulting neighbor is chosen as offspring. For determining the next generation, the best individual is direct passed to the next generation (elitist strategy).
For the computational tests, they used small (up to $n = 4, m = 5$), medium (up to $n = m = 10$ and large (up to $n = m = 30$) instances, where the level L of the re-entrant problem ranges from 2 to 10. The algorithm used $PS = 50, P_C = 0.8, P_M = 0.1$ and $NGEN = 100$ for small, $NGEN = 200$ for medium and $NGEN = 400$ for large problems. The new algorithm was

superior to the NEH heuristic, the generation of non-delay schedules and a pure GA.

Tseng and Lin [112] considered both the problems $F \mid prmu \mid C_{max}$ and $F \mid prmu \mid \sum C_i$. They developed a hybrid algorithm, where a GA does the global search and a novel local search scheme combining two local search methods does the local search. This papers introduced a new crossover operator, called *orthogonal array crossover*, applied to a solution represented in the permutation code with $P_C = 0.5$. This crossover determines multiple cut points, and several recombinations of subsequences based on the orthogonal array and Taguchi methods were used to select better subsequences for the resulting offspring. In theirs tests, Tseng and Lin [112] used six cut points for the small instances and 14 cut points for the medium and large instances. For applying local search, the shift operator was used (which turned out to be superior to the pairwise interchange operator, while the pairwise interchange operator was applied in a mutation). The fi rst local search is based on determining a best neighbor in a sub-neighborhood of the shift neighborhood which is accepted if it is an improvement, while the second search procedure with *cut-and-repair* has a larger diversification ability and may help to jump out of a local optimum. This cut-and-repair procedure is based on the determination of two adjacent pairs of jobs in the sequence that produce the largest idle times and determines other jobs which might be sequenced between these adjacent pairs so that idle time is reduced as much as possible.
The different variants of the above strategy have been first compared on 15 instances from [110], and later the hybrid GA on a set of 90 benchmark instances from [110] with two ant colony algorithms and a particle swarm algorithm from the recent literature. For the latter tests, the hybrid GA used $PS = 20$ for the instances with $n = 20$ and $PS = 100$ for the instances with $n \in \{50, 100\}$ for minimizing total flow time. The hybrid algorithm from this paper turned out to be superior to the other algorithms. In particular, for minimizing total flow time, the new hybrid algorithm improved 66 out of the 90 current best solutions reported in the literature on short-term search and achieved them for 23 of the remaining 24 instances. In addition, the new algorithm improved all the 20 currently best solutions reported in the literature on long-term search. For the makespan criterion, the authors used $PS = 100$ for all instances and the new hybrid GA also performed better than the other algorithms.

Three genetic algorithms for the problem $F|prmu|\sum T_i$ of minimizing total tardiness have been given by Vallada and Ruiz [114]. They used two initialization schemes, where in addition to randomly generated solutions the EDD sequence (variant 1) and both the EDD and the NEH [74] heuristics are considered. The n-tournament selection procedure is applied but a given percentage of the population is randomly selected for producing the offspring. After experiments, the authors used the SBOX crossover operators from [99] and a one-point crossover as well. Mutations are based on the shift neighborhood. The authors use a steady state algorithm in which the offspring are inserted into the new generation if they are better than the worst individuals if they are not already in the population. The GA is combined with a local search algorithm based on the shift neighborhood, determining the best neighbor. This procedure is applied to best individual in the initial population and to the generated offspring by the genetic operators with a specific probability. The authors also include a *diversity value* based on the number of times that jobs appear at the different positions of the solutions currently in the population and three variants of a *restart mechanism*, which is applied when the diversity falls below a given threshold

value. The GA also includes a *path relinking mechanism* that is based on the pairwise interchange neighborhood, which complements the mutations based on shifts of jobs. The authors suggested three variants of a GA. The first one (GAPR) substitutes the crossover operator in favor of the path relinking technique. The second one (GAPR2) maintains crossover and applies the path relinking technique as an additional step whenever the best solution of the population has not been improved over a given number of generations, and the third one (GADV) does not use the path relinking technique but considers the restart mechanism.

After calibrating the GA, all variants worked with $PS = 30$, the one-point crossover with $P_C = 1.0$ (if applied), $P_M = 0.02$, and local search is applied with probability 0.15 to an offspring. In the computer experiments, the new algorithms were compared with existing ones (a total of 15 variants have been considered) on the 540 benchmark instances from [115] with three different CPU time limits proportional to nm. The three proposed genetic algorithms clearly outperformed both the adapted and the existing methods for the total tardiness problem. It has also been observed that most algorithms developed for other objectives adapted by the authors to the total tardiness problem showed a better performance than the existing algorithms for the total tardiness problem.

Jarboi et al. [50] presented a hybrid GA for the no-wait flow shop problem. The objective is either the minimization of makespan or of total flow time. A solution is represented by the permutation code. The initial population generates individuals based on the use of the NEH [74] heuristic with different insertion orders of the jobs. Parents are selected using the same ranking procedure as in [94]. Jarboi et al. [50] suggested a new crossover operator, called the *block order crossover*. It is based on blocks of jobs in both parents at any (not necessarily the same) positions. For a mutation, the shift operator was applied. This GA was combined with a variable neighborhood search procedure. They applied two neighborhoods, namely the pairwise interchange and the shift neighborhoods. A generated offspring replaces an individual from the subset of the 20 % worst individuals if it is better.

For the computational tests, 21 instances from Reeves [94] and 110 instances from Taillard [110] (together with two other instances) have been used. The parameters used for their hybrid GA were $PS = 10, P_C = 1.0$ and $P_M = 0.8$. This algorithm has been compared e.g. with several variants of iterative improvement, tabu search, variable neighborhood search and another hybrid GA. It turned out that the new hybrid algorithm is superior to the compared algorithms for both criteria (and the best known upper bounds were improved for some instances), which indicates that the hybridization procedure between the GA and the variable neighborhood search can coordinate the main properties of both algorithms. It was also observed that this hybridization reduced the range of the results between the maximum and minimum deviations compared with the other algorithms tested, and so it produced more robustness. An exception were larger instances from [110], where the pure variable neighborhood algorithm obtained better results than the new algorithm.

Finally, we briefly sketch some algorithms considering *multi-criteria flow shop* problems.

Murato et al. [71] presented a hybrid GA for the multi-objective problem minimizing

a linear combination of makespan and total tardiness, and in addition also combined with total flow time. This algorithm uses a two-point crossover for generating one offspring and the shift mutation. They incorporated local search into the GA which leads to an improvement of the performance. The selection procedure is based on combining the multiple criteria into a linear combination but with variable (i.e., randomly generated) weights which realize various search directions. The elitist strategy is applied in a way that, when using n objectives, the n best individuals (one for each criterion) are kept in the population. Finally, the GA outputs a set of Pareto optimal solutions.
The algorithm has been run with $PS = 10, P_C = 1.0$ on a randomly generated instance with $n = 20$ and $m = 10$ such that 100,000 solutions are evaluated. The authors concluded that their algorithm could find better solutions than a previous vector evaluated GA and the corresponding single-objective GA.

Onwubulu and Mutingi [81] considered the flow shop problem with three different objectives: minimization of total tardiness, minimization of the number of tardy jobs and the minimization of a linear combination of both criteria. The crossover operator is a two-point crossover (as for a binary code), where illegal parts are repaired by pairwise interchange mutations. The algorithm used $PS = 20, P_C = 0.7$ and $P_M = 0.4$. When diversity falls below a predetermined value, the mutation operator is used until the diversity is above the preset value. However, the best individuals are maintained (so that the individual with the largest fitness is never lost). The algorithm has been tested on own test instances with up to 30 jobs and 15 machines.

Basseur et al. [9] dealt with the bi-criteria flow shop problem of makespan and total tardiness. They presented an adaptive Pareto GA, where the choice of genetic operators is dynamically made during the search. In particular, several mutation operators are considered and applied during the search with changing rates. Here, the authors considered the shift and pairwise interchange mutations. In a similar way, this can be done for crossover and/or local search operators. Moreover, a *diversification mechanism* is considered, which is based on the *sharing principle*. This consists in the degradation of the fitness of an individual belonging to search regions with a high concentration of solutions. The new fitness of an individual is equal to the old fitness divided by a so-called *sharing counter* of the individual. This GA is hybridized with a memetic algorithm, where the mutation operator is replaced by a local search heuristic.
The algorithm has been tested on a subset of the benchmark instances from Taillard [110]. The authors introduced two complementary types of performance indicators for a multi-criteria optimization problem: *contribution* (the contribution of a set S_1 of solutions in relation to S_2 is the ratio of non-dominated solutions in S_1) and *entropy* (which describes the diversity of the solutions found).

Ponnambalam et al. [86] presented a GA for minimizing a weighted sum of makespan, mean flow time and machine idle time, where the weights are not constant but they are randomly specified in each selection (*multi-directional search*). For constructing the initial population, a distance matrix is defined and then an insertion heuristic for the TSP is applied to obtain one individual, while the other solutions are determined by applying shifts to selected jobs. For the crossover, the PMX operator is applied with $P_C = 0.6$ while as the mutation operator, the pairwise interchange operator is applied with $P_M = 0.05$. The algorithm stops after generating $NGEN = 250$ generations.

The algorithm has been applied to 21 instances from the OR Library. The authors emphasized that the best solutions obtained are close to the initial solutions but, unfortunately, no comparison with other algorithms has been made.

Pasupathi et al. [87] considered the problem of minimizing the makespan and total flow time in a permutation flow shop. They presented a Pareto-ranking based GA with an archive of non-dominated solutions subjected to a local search (denoted as PGA-ALS), i.e., this GA maintains an archive of non-dominated solutions which is updated by applying local search at the end of each generation. Their algorithm uses the principle of *non-dominated sorting* combined with the use of a metric for the crowding distance being used as a secondary criterion. The initial population is formed by four individuals determined by four heuristic procedures and applying an improvement scheme to each of them, while the remaining individuals are randomly determined. Binary tournament is applied to select the parents. The crossover used is a standard order-based one-point crossover applied with $P_C = 1.0$. The mutation operator generates a shift neighbor with $P_M = 0.1$. For determining the next generation, successive non-dominated fronts are formed using both the parents and the offspring, and then primary and secondary ranks are assigned to every chromosome, which are used to select $PS/2$ individuals, while the remaining individuals are randomly chosen.
For the computational results, 90 benchmark instances from [110] have been considered. The new algorithm has been compared with the three best existing genetic algorithms for multi-criteria problems. The non-dominated solutions found by the particular algorithms have been compared and subsequently combined to obtain a *net non-dominated front*. The tests showed that the new GA found most of the solutions in this front when compared with the existing algorithms.

Chang et al. [20] presented a sub-population based GA with mining gene structures for the multi-objective flow shop problem. The population is partitioned into independent and unrelated sub-populations, and the multiple objectives are scalarized into a single objective in such a way that each sub-population is assigned a different weight. The resulting mining problem of elite chromosomes is formulated as a linear programming problem, and a greedy heuristic using a threshold is applied to eliminate redundant information. A storage file is setup to record non-dominated solutions obtained during the search process.
For the computational study, this algorithm (denoted as MGSPGA) has been tested on instances with $n \in \{20, 40, 60, 80\}$ and $m = 20$. For MGSPGA, the parameters $PS = 200, P_C = 0.9$ and $P_M = 0.6$ have been used. The number of sub-populations is 20, and the algorithm stopped after 1,000,000 fitness evaluations. This algorithm has been compared with a previous sub-population GA, a strength Pareto evolutionary algorithm and a non-dominated sorting GA. It turned out that the new algorithm obtained superior results.

Sridhar and Rajendran [105] presented a GA for the flow shop problem with the three objectives of minimizing the makespan, total flow time and machine idle time. The algorithm uses two sub-populations of size n. The first sub-population uses the solution obtained by the NEH [74] heuristic and the remaining individuals result from the $n-1$ adjacent pairwise interchanges in this sequence, and the second sub-population uses a constructive algorithm for the mean flow time problem, and all other individuals result again from the possible adjacent pairwise interchanges, so that in total $2n$ individuals are obtained. For the crossover, the PMX operator is used. A mutation is based on a pairwise interchange of two jobs, but is only applied after each 5 generations. For the replacement of parents by the

offspring, a specific evaluator, called DELTA, was presented. The parameter $NGEN$ was limited to 11.
A computational comparison with a heuristic by Ho and Chang [46] has been made on randomly generated instances with up to $n = 30$ jobs and $m = 25$ machines. The presented algorithm was clearly superior, where for the final solution, all three objectives had better values.

Uysal and Bulkan [113] presented a genetic (and a particle swarm) algorithm for the bicriteria problem of minimizing a linear combination of makespan and maximum tardiness. They used a randomly generated population with $PS = 2n$. For generating two offspring, one parent is chosen and the other one is selected according to binary tournament. The latter is also used to construct the next generation. A one-point crossover is applied with probability $P_C = 1.0$, and a shift mutation is applied with $P_M = 0.3$. This GA (as well as the particle swarm algorithm) are hybridized with a variable neighborhood search, where both the shift and pairwise interchange neighborhoods are applied.
These algorithms have been tested on the benchmark instances given by Demirkol et al. [29]. Comparing the pure GA and the particle swarm (PSO) algorithm, the PSO was superior if the T_{max} portion is dominant in the objective function while the GA performed better if C_{max} is dominant. For the hybrid versions, the PSO variant was slightly superior to the GA for all linear combinations of C_{max} and T_{max} considered (which was also faster).

Dhingra and Chandna [30] considered the flow shop problem with sequence-dependent setup times and multiple objectives. In particular, they considered the minimization of

$$\alpha \sum T_i + \beta \sum E_i + \gamma C_{max},$$

where α, β and γ are given weights and E_i denotes the earliness of job J_i. Note that this is not a regular criterion. Unfortunately, nothing is said about the representation of a solution although a feasible solution may have inserted idle times between some operations due to the earliness term in the objective function. The algorithm used the insertion heuristic from [74] applied with four different insertion orders of the jobs for constructing a part of the initial population. The population size was set to $PS = 50$. Some standard crossover operators have been compared to each other, and it has been found that the OX operator worked best which was applied with a rate $P_C = 0.8$. The mutation operator was of the pairwise interchange type and was applied with $P_M = 0.15$. According to the elitist strategy, the two best individuals are directly inserted into the next generation. The termination criterion was a maximum time limit of $n \cdot m \cdot 0.25$ s. The algorithm has been tested on instances generated from Taillard's set [110].

Finally, we only mention that a GA for the special case of a two-stage flow shop problem has been given by Nepalli et al. [75] (which we do not discuss here).

5.1 FLEXIBLE FLOW SHOPS

Sivrikaya Serifoglu and Ulusoy [106] considered the FFS scheduling problem with identical parallel machines and multi-processor tasks, where particular tasks (or operations) can require more than one machine of the corresponding stage. The objective is to minimize the makespan. The presented GA is based on a permutation representation of the jobs describing the first stage sequence. A sequence is decoded into a schedule by scheduling the jobs at the other stages according to non-decreasing completion times at the immediately

preceding stage as soon as the number of simultaneously required processors is available. The major part of the initial population is randomly generated but three specific dispatching rules (SPT, LPT and shortest total processing time) are also used. Parents are chosen according to roulette wheel selection. The authors experimented with a uniform order-based (or PBX) crossover and the two-point crossover from [72]. Finally, the uniform o rder-based operator was superior and applied with $P_C = 0.75$. For the mutation operator, the shift and pairwise interchange operators were tested, and finally the pairwise interchange operator was chosen and applied with $P_M = 0.75$. Moreover, after initial tests, $PS = 50$ and $NGEN = 400$ have been chosen and the elitist strategy was applied by inserting the two best individuals into the next generation.
The authors considered 10 instances for each of the combinations (n,k) with $n \in \{5,10,20,50,100\}$ and $k \in \{2,5,8,10\}$, each for two types of problems, where the number of processors per stage was randomly determined from $\{1,\dots,5\}$ and where this number was always equal to 5. This GA has been compared with a lower bound, the three heuristic rules and an optimal solution for the small problems as well as the estimated optimum values for larger problems. Particularly for small problems, the deviations from the lower bound were still rather large but the improvements of the GA over the heuristic rules were substantial.

Oguz and Erdan [79] also considered the FFS problem with multiprocessor tasks and identical machines at each stage. A solution is encoded by a string of length n describing the job sequence in the first stage. Then a resulting schedule for the k-stage problem is constructed by a list scheduling algorithm. The initial population is randomly generated. Selection of parents is done via roulette wheel selection. The authors suggested a new crossover operator (denoted as NXO). This operator aims to keep the best characteristics of the parents in terms of adjacent jobs. If two jobs are adjacent in two parents with a good fitness value, then this structure is intended to be used also in the offspring. However, if such a structure cannot be found, one looks for the next job that fits in terms of the processor allocation. These criteria contributed to keeping idle times small. The operator always produces legal offspring. The two mutation operators considered are the shift and the pairwise interchange operator. The stopping criterion was based on a maximum number of 10000 generations and a maximum CPU time limit of 30 minutes.
For the computational tests, 10 randomly generated instances (n,k) have been generated with $n \in \{5,10,20,50,100\}$ and $k \in \{2,5,8\}$ in two settings: the number of machines at each stage is either a randomly determined number from $\{1,\dots,5\}$ or always equal to 5. The authors made intensive computational tests for several combinations of the parameters. They found that $PS = 100$, the use of the NXO operator with $P_C = 0.8$ and the use of the shift mutation with $P_M = 0.1$ can be recommended. In particular, the new operator was slightly superior to the standard PMX operator. This GA was also superior to a tabu search algorithm. Finally, the authors compared the variants of the GA and tabu search on the machine-vision problem with $n \in \{8,10,12,18\}$, where tabu search performed better for the problems with $n = 18$.

Kahraman et al. [56] presented a GA for the FFS problem which is based on a permutation representation of the jobs. The initial population is randomly generated. Then they made detailed computer experiments with standard selection, crossover and mutation operators. They tested both roulette wheel and tournament selection, the PBX, OX, PMX,

CX, LOX and OBX operators. In addition, several mutation operators have been examined: shift, API, pairwise interchange and inversion mutation and a more general neighborhood by selecting three jobs and permuting them arbitrarily.
The algorithm has been tested on the benchmark instances ranging from $(n,k) = (10,5)$ to $(15,10)$ given in Carlier and Neron [19], where 77 instances are classified into 13 groups. After initial tests, roulette wheel selection, $PS = 25$, the PBX crossover with $P_C \in \{0.1, 0.2, 0.3\}$ (in dependence on the problem type), and inversion mutation with $P_M \in \{0.1, 0.2\}$ have been chosen. The CPU time was limited to 1600 s. The GA was compared with a branch and bound algorithm and an artificial immune system. The GA found an optimal solution for all instances classified as easy instances (type a and b problems in this paper), and this algorithm performed also better for the hard instances (type c and d problems); the average percentage deviations from a lower bound increased from type a to type d). Both the GA and the branch and bound algorithm were able to solve about 70 % of the hard instances, but the percentage deviations from the lower bound were smaller for the GA than for the other two approaches.

Jolai et. al [52] addressed the no-wait flexible flow line problem with due windows and job rejection, i.e., each job has to be completed within a predetermined due window and thus, some jobs may be rejected. The objective is to maximize the total profit gained from the jobs scheduled. A solution is represented by a string of the n jobs, were each gene contains the starting time of a job. A heuristic method is used to construct a feasible schedule for each member of the initial population. Parents are chosen according to roulette wheel selection. A one-point crossover is used (it has been found that $P_C \in [0,6,0.9]$ should be chosen), and a gene is mutated with $P_M = 0.01$. Moreover, the algorithm used $PS = 80$ and $NGEN = 100$.
The algorithm has been tested on instances with up to 50 jobs and 5 stages. A comparison has been made with optimal solutions found by applying LINGO 8 to the mixed integer programming model derived in this paper and a 'simple GA'. Among the instances considered, the new GA found an optimal solution for 264 instances (over 65 % of the considered ones). The maximal error for the instances with 2 stages was 1.02 % and for the other instances, the maximal error was less than 1 %. The average computation times of the new GA were smaller than those with the LINGO 8 package. Moreover, the performance of the new algorithm was significantly better than that of the simple GA.

Jungwattanakit et al. [53] presented a GA for the FFS problem with given sequence-dependent setup times, release dates of the jobs, unrelated parallel machines at each stage and with the objective of minimizing a linear combination of makespan and the number of tardy jobs, and this algorithm has been compared in detail for many settings to simulated annealing and tabu search algorithms in [54]. This GA used the permutation code for describing an individual which represents the job sequence at the first stage. In this paper, several heuristics for the classical flow shop problem have been adapted to the flexible flow shop. However, for the problem considered, the processing times depend both on the chosen machine of a particular stage and on the job previously processed. For this reason, these constructive algorithms are run for nine combinations of the representatives of speeds of the machine and setup times: use all combinations of the minimum, average and maximum data, determine a heuristic solution for each particular combination and choose the best sequence among them. To do this, there is given an algorithm, which generates a complete

schedule for the flexible flow shop problem from a particular job sequence describing the job sequence at the first stage. If this sequence has been chosen, the job sequences for the remaining stages are either determined by the permutation rule (i.e. they are identical) or by the FIFO rule.
The initial population is determined by means of several constructive methods for finding the job sequence on the first stage such as the NEH [74] and the CDS [18] heuristics, the heuristics by Dannenbring [26] or simple dispatching rules like SPT, LPT, EDD etc. as well as fast polynomial time iterative algorithms. The remaining individuals for the initial population are randomly generated. As the crossover operator, both the PMX and the OPX (combined linear order and position-based operator, which generated one offspring by the LOX operator and the second offspring by the PBX operator) crossovers were tested. The mutation operator is either based on the shift or on the pairwise interchange neighborhood. In initial tests, appropriate parameter settings have been determined. For the population size, values $PS \in \{10,30,50,70\}$ have been tested and finally, a value of $PS = 30$ has been used. It turned out that the OPX operator is substantially better than the PMX operator Crossover and mutation rates have been tested in steps of 0.1 within a range from 0.1 to 0.9. The shift mutation turned out to be better if the weighted makespan is dominant while the pairwise interchange mutation is superior if the weighted number of tardy jobs is dominant. Moreover, the crossover rate $P_C = 0.6$ and the mutation rate $P_M = 0.3$ have been recommended.
In [54], a comparison with simulated annealing and tabu search has been made on test instances of the format $(n,m) \in \{10,5),(30,10),(50,20)\}$ using the same time limit of all three types of algorithms depending only on the problem size (1 s for the instances with $n = 10$, 10 s for the instances with $n = 30$, and 30 s for the instances with $n = 50$). For this setting, simulated annealing turned out to be slightly superior to the GA (however, due to the chosen value of PS, this means that simulated annealing performs a substantially larger number of iterations than the number of generations in the GA). On the other hand, for small problems it turned out that the GA with including the insertion heuristic into the initial population (algorithm NEHGA) produced a rather small deviation from the optimal solution found by means of CPLEX (in 20 of the 40 classes of instances considered, the average relative deviation was smaller than 2 %).

Yaurima et al. [132] considered the FFS problem with unrelated parallel machines, sequence-dependent setup times, machine availability constraints and limited buffers for a television printed circuit-board production. Each solution is represented by a string of the jobs characterizing the job sequence from which a schedule is determined in a greedy manner. The algorithm uses a binary tournament for selecting the parents. The core of the paper is the consideration of eight crossover operators, among them five existing variants: OBX, PPX, a one-segment crossover, a two-point crossover, and SB2OX. In addition, the authors introduced three new operators: *Two-point inverse* (TPI), *order based setup time crossover* (OBSTX), and *setup time two-point crossover* (ST2PX). The latter two variants arose from the generalization of existing crossover operators to the problem with setup times. To maintain diversity in the population, they applied a slightly modified ex isting replacement strategy.
Based on detailed computational tests, the parameters $PS = 200$, ST2PX crossover with $P_C = 0.8$, and pairwise interchange mutation with $P_M = 0.1$ were applied. The algorithms

stopped when the best fitness value was not improved over 25 generations. For the comparison, the authors used the GA from [98] adapted to the case of limited buffers. The instances included $n \in \{50, 100\}$ jobs and $k \in \{2, 3, 6\}$ stages. The authors found that their algorithm outperformed the adapted algorithm from [98] for all types of problems.

Zhan et al. [134] presented a hybrid GA for the FFS problem with unrelated parallel machines, where the objective is to minimize the makespan with load balancing. A chromosome is composed by a machine allocation and an operation sequencing string. The initial population is randomly generated by respecting the constraint that the load difference between parallel machines is smaller than 60 %. As the crossover operator, an evolution-based multiple offspring crossover is used, and a mutation can be performed both in the allocation (change of a selected gene) and the sequence (based on a pairwise interchange) sub-chromosome in such a way that always a legal offspring results. The GA is combined with a simple local search procedure. 10 % of the parents and offspring with high fitness are directly passed to the new generation while the other individuals for the next generation are chosen by roulette wheel selection. Unfortunately, computational results have been presented only for an instance with 50 jobs under various settings. The authors claimed that their algorithm is better than a 'general GA'.

Rashidi et al. [92] presented a hybrid parallel GA for the flexible flow shop problem with unrelated parallel machines. The problem includes sequence-dependent setup times and processor blocking, and the objective is to minimize a linear combination of makespan and maximum tardiness. A solution is described by a string of length n, each gene representing a job, filled with random keys. This chromosome describes the first-stage sequence. Then the jobs in the remaining stages are assigned and sequenced in a greedy manner taking into account the setup times, speeds of machines and possible constraints.
The algorithm starts with a randomly generated population. Parents are randomly selected, and a one-point crossover is applied. The mutation operator is of the pairwise interchange type. A new individual replaces only its parent, if the objective function value is better and if its job sequence is not already in the population. This GA is used as a parallel algorithm consisting of 21 sub-populations (algorithm MOPGA). After initial experiments, the values $PS = 1050, P_C = 0.7, P_M = 0.1$ together with an elitist strategy using a rate of 20 % are used. Then this procedure is refined (algorithm IHMPOGA). On 10 % of the best individuals in each sub-population a local search method is applied. Two chromosomes are chosen and decoded into a schedule. Similarities of these schedules are identified and copied into a new (incomplete) chromosome. Then several ways (as they call options) of completing such a solution are considered, where an upper bound on this number is imposed. To overcome local optima and maintain diversity, this local search procedure is embedded into a new method called 'redirect'. If local search does not yield an improvement, the 20 % best individuals are maintained, among the other 80 %, 50 % are replaced by applying a pairwise interchange mutation, and the other 50 % are replaced by new randomly generated solutions. For this hybrid algorithm, a one-point crossover, a two-point crossover and a parametrized uniform crossover are applied, and both the shift and pairwise interchange mutation have been tested.
For the tests, 27 instances with $n \in \{6, 30, 100\}$, the number of stages k equal to 2, 4 or 6 and a number of machines per stage uniformly distributed in $\{1, \ldots, 4\}$ were generated. Both variants have been tested against a standard approach, where the multi-objective problem

is transformed into a single-criterion problem using constant weights. The latter approach turned out to be comparable with variant MOPGA, but the hybrid algorithm IHMPOGA was superior to the other variants in terms of non-dominated solutions found.

Zandieh et al. [133] presented a GA for a flexible flow line including constraints such as sequence-dependent setup times, machine release dates or time lags. The encoding scheme is comprised of two parts: the job sequence and the machine assignment parts, but this representation is only applied to the first stage. Roulette wheel selection is applied for selecting the parents. The crossover is of a parametrized uniform crossover type, where with probability 0.7 the gene from the first parent is taken and otherwise the gene from the second parent. For applying a 'mutation', a portion of the next generation is filled by randomly generating new chromosomes (which differs from the classical realization of a mutation).
For finding the optimal parameters, the *response surface methodology* has been applied. The authors distinguished three problem classes. For small problems, they found the optimal values $PS = 157, NGEN = 153, P_C = 0.81, P_M = 0.13$. For medium problems, these parameters were $PS = 255, NGEN = 375, P_c = 0.64, P_M = 0.29$. For large problems, the optimal parameters were $PS = 255, NGEN = 370, P_C = 0.75, P_M = 0.20$. The GA has been tested on a total of 768 small, 1152 medium and 1536 large problems and compared with the NEH [74], the SPT and LPT heuristics. It has been found that the GA outperforms these heuristics for all three types of problems with respect to solution quality but with larger computational times.

6 JOB SHOP PROBLEMS

For the problem $J \,||\, C_{max}$, a first GA has been given by Davis [27] although his algorithm did not solve any real problems. This GA encodes the representation of a schedule in such a way that the genetic operators operate in a 'meaningful' way and that a decoder always produces a legal solution for the problem.

Yamada and Nakano [128] developed an algorithm often denoted as GT-GA. This algorithm uses a direct representation in the form of a string of the operation completion times of an active schedule. They introduced a crossover based on the G&T algorithm as follows. As each decision point, one parent is randomly selected, but the operation which is schedulable and has the earliest completion time in the parent schedule is scheduled next. This strategy corresponds to a uniform crossover combined with the G&T algorithm as an interpreter to transform an illegal offspring into a feasible active schedule.
The algorithm has been tested on the three benchmark instances from [35]. The algorithm found an optimal solution for the instance with $n = m = 10$. The authors also observed that, without including the crossover, the computational times were longer and the results of the individual runs varied widely.

Della Croce et al. [28] introduced a GA, where the encoding scheme is based on preference rules such that always a feasible chromosome arises. However, this encoding generates non-delay schedules (so it is possible that an optimal solution cannot be encoded). The encoding scheme defines a string for each machine (sub-chromosome) which describes a *preference list* (i.e., it does not necessarily describe the sequence of the operations on this machine). This list is used at the end of an operation when one has to choose an operation

among those waiting to be scheduled. The actual schedule is then determined by a simulation that analyzes the state of the waiting queues by using the preference lists. By means of two selected parents, two offspring are generated, where the LOX operator is applied to each sub-chromosome and as the mutation operator, the pairwise interchange operator is applied to a randomly chosen sub-chromosome. To refine this algorithm, the authors tried to extend the search space outside non-delay schedules by incorporating a so-called *look-ahead evaluation* in the form of a look-ahead simulation method. Basically, this procedure should decide whether it is preferable to keep a machine idle for a certain time because an operation with higher priority enters the queue for this machine or not. However, also by this extension, not every active schedule can be generated. In addition, the algorithm uses an *updating technique* based on classes of equivalent chromosomes (using eigenvectors). The authors have found that these modifications lead to an improvement of the performance of the algorithm.
The GA has been tested on the benchmark instances from [35] and some of the instances from [61]. The GA has been run with $PS = 300, P_C = 1, P_M = 0.03$ and steady-state reproduction, where in each generation 10 new individuals are inserted into the population. The number of generations has been fixed in such a way that totally 30000 individuals were generated in each run of the algorithm. The solution quality is similar to the shifting bottleneck heuristic, a simulated annealing and a tabu search algorithm, but computational times are larger for the GA.

Dorndorf and Pesch [31] provided a *probabilistic learning strategy* based on evolution principles, i.e., the authors used an underlying GA as a meta-strategy to guide an optimal design of local decision rule sequences. They considered two strategies to be incorporated into a genetic framework, namely one approach based on priority rules and another one based on the advanced shifting bottleneck strategy using a partial enumeration tree. The reason for this was the observation that particularly for the job shop problem, the convergence of genetic algorithms is often rather low if no particular improvement heuristics are used in the GA.
The first variant, a priority-based genetic algorithm (P-GA), uses a string of entries the number of which is equal to the number of operations. Each entry represents one rule out of 12 given priority rules such as LRPT (longest remaining job processing time), SRPT (shortest remaining job processing time), FCFS (first come first served), and others. The entry in position i denotes the priority rule which is applied in the i-th iteration of the G & T algorithm for solving the corresponding conflict. The goal is to determine the best sequence of priority rules used by the genetic framework. The crossover is a simple two-point crossover (always generating feasible chromosomes). The mutation operator switches a string position to another one with low probability.
In the second variant, the shifting bottleneck based genetic algorithm (SB-GA), the GA controls the selection of nodes in the enumeration tree of the shifting bottleneck heuristic (variant SB2). A complete enumeration tree is generated up to a specific level, while for the higher search levels only a partial enumeration tree is considered. Here the length of a string of an individual is equal to the depth of the enumeration tree which corresponds to the number of machines, and a partial string of the first k entries describes the sequence in which the machines are considered in the SB1 heuristic (note that the SB2 heuristic is based on a repeated application of the SB1 heuristic that predetermines the sequence in which the

single machine problems are solved). The crossover operation is the CX operator, while mutation is not applied.
These algorithms have been tested on benchmark instances (e.g. the instances from [35] and the 40 instances from [61]) as well as 21 own problem types each consisting of five instances. For P-GA, two variants have been considered. The main parameters for the first variant are $PS = 200, P_C = 0.65, P_M = 0.001$ and in addition an inversion rate of 0.7, and for the second variant $PS = 300, P_C = 0.65$ have been chosen. Two variants have also been considered for SB-GA: the first variant used $PS = 40, P_C = 0.75$, and the second variant used $PS = 40, P_C = 0.65$. All variants considered the elitist strategy, and the termination criterion was a given maximal number of function value evaluations depending on the variant of the algorithm. The new algorithms were compared with the shifting bottleneck heuristic, genetic local search and simulated annealing. While P-GA was not competitive in the solution quality, it has nevertheless advantages such as ease of implementation and robustness to problem changes. Algorithm SB-GA produced a similar solution quality as genetic local search and simulated annealing for small problems, and it was superior to these algorithms and the pure shifting bottleneck heuristic for larger problems.

Bierwirth [11] presented a generalized permutation approach for the makespan problem, where the chromosome is represented as a permutation with job repetitions. The crossover operator used is a *generalized OX* operator, denoted as GOX. This operator is illustrated in Fig. 17 for a job shop problem with $n = m = 3$ using job repetitions. For each operation, we give its index, where index k means that this operation is the k-th operation of the corresponding job. By means of two parents, one offspring is generated. From the donating chromosome (here parent 1) a sub-string is chosen (1 3 2 3 of genes 4 - 7). Then all genes of the sub-string in the receiving chromosome (parent 2) are deleted with respect to their index of occurrence in the receiving chromosome (see the underlined numbers). Then this sub-string is implanted into the receiver at the position, where the first gene of the sub-string has occurred (here it is position 2 in the offspring). Note also that here the string 1 3 2 3 in the offspring refers to the operations with the same indices as in parent 1 which is not necessarily the case since the given technological routes have to be respected. The algorithm has been tested on some benchmark instances from [8, 35, 61], where PS was either 100 or 150 depending on the problem size. The authors mentioned that without tuning and weak hybridization only, the algorithm reached a solution quality comparable to other genetic algorithms.

Kobayashi et al. [58] presented a GA, where a schedule is encoded by a list of job sequences on the machines. The core is the introduction of a new crossover operator SXX (*subsequence exchange crossover*), which is derived from the sub-tour exchange crossover for the TSP. This operator exchanges subsequences in parents consisting of the same jobs on each machine. This operator is illustrated in Fig. 18, where two offspring are generated. Given two parents represented by a partitioned permutation (job sequence matrix), in each sub-chromosome, two 'exchangeable' sub-sequences are interchanged (sub-sequences $J_2, J_4, 3_4$ on M_1, J_4, J_5 on M_2 and J_2, J_1, J_4 on M_3 in parent 1 are interchanged with sub-sequences J_4, J_2, J_3 on M_1, J_4, J_5 on M_2 and J_2, J_1, J_4 on M_3 in parent 2). The G & T algorithm is then used to transform the generated offspring into an active schedule. Mutation is not performed since the major goal was to evaluate the SXX operator. From the two parents and the two generated offspring, the two best individuals are inserted into the new

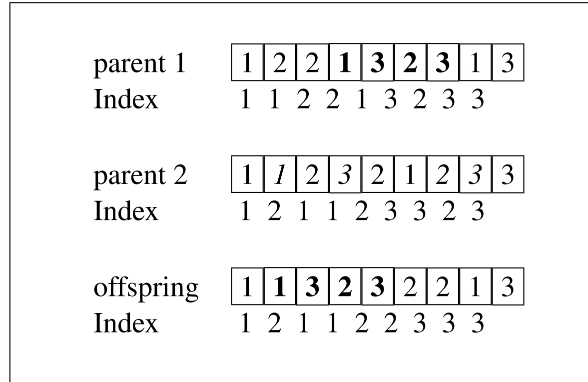

Figure 17. GOX crossover

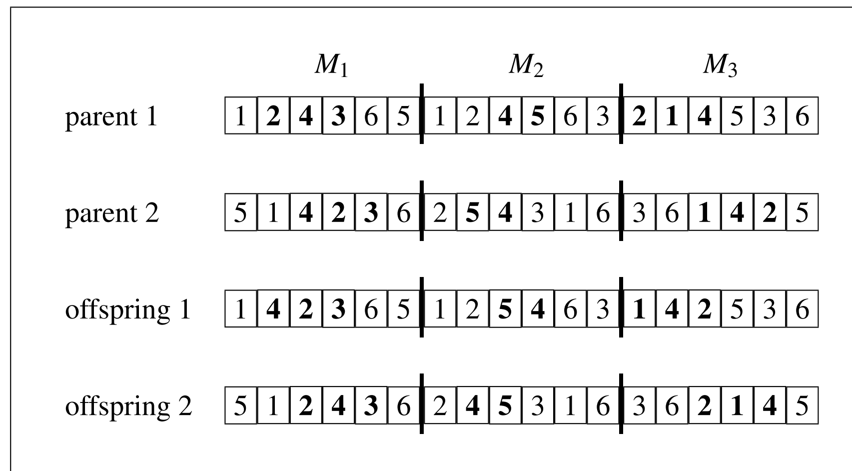

Figure 18. SSX crossover

generation (2/4 selection).
This algorithm has been compared to the algorithm from [128] on the benchmark instance with $n = m = 10$ from [35] using $PS = 600$. The authors found that their algorithm is superior to the GA-GT algorithm from [128]. They also found that replacing the SXX operator by the OX crossover yielded worse results. It turned out that the use of the SXX operator led in 51 of 100 runs to the optimal makespan value of 930, while the use of this algorithm with the OX operator produced an average makespan value of 964 in 100 runs. However, their algorithm did not find an optimal solution for the instance from [35] with $(n,m) = (20,5)$.

Bierwirth et al. [12] also used an operation-based representation described by permutations with repetitions. The core of this paper is the comparison of three crossover operators, namely the GOX operator (see [11]), the *generalized position crossover* (GPMX) and the *precedence preserving crossover* (PPX). The GPMX operator is illustrated in Fig. 19 for the same parents and sub-string as in Fig. 17. GPMX works similar as GOX, however, the chosen string is implanted into the receiver at those positions, where it occurs in the donator (positions 3 - 6). The PPX operator defines a vector of length n filled randomly with numbers 1 and 2 which defines the order in which the genes are drawn from parent

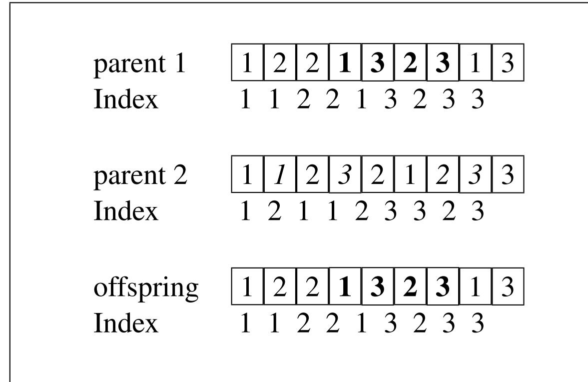

Figure 19. GPMX crossover

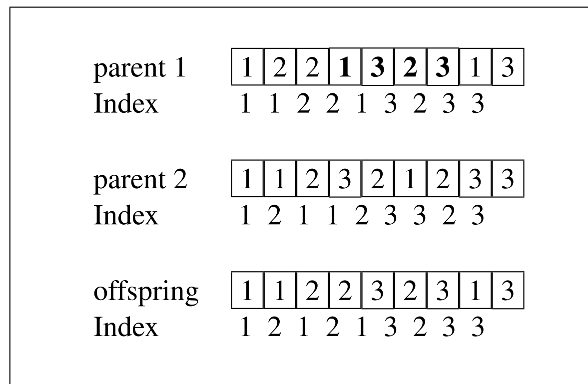

Figure 20. PPX crossover

1 and 2, respectively (starting from left to right). After a gene is drawn from one parent and deleted in the other one, it is appended to the partial chromosome of the offspring. The PPX operator is illustrated in Fig. 20 in a two-point fashion, where genes 4-7 are taken from parent 1 (so the cut points are 4 and 7) and the other ones from parent 2. Thus, the GOX operator tends to pass the relative order of the genes. GPMX tends to pass the positions of genes to the offspring while the PPX operator respects the absolute order of the genes which leads to a preservation of the precedence relations among the genes. The main part of the computational results was dedicated to the comparison of the three crossovers, where $PS = 500, P_C = 0.6$ and $NGEN = 100$ have been used, while mutation is not applied. Using five benchmark instances from [107], the PPX operator outperformed GPMX, while GOX produced the largest relative error. This underlined that an operator should preserve precedence relations among the operations.

Yamada and Nakano [129] introduced a new crossover operator denoted as *multi-step crossover* (MSX). This operator utilizes a neighborhood structure together with a distance measure in the space of the solutions. The basic idea is to generate the offspring along the path connecting the two parents (in the underlying neighborhood graph). The idea is to evaluate a generated neighbor x of the first parent P1 not by the objective function value but by the distance between x and the second parent P2. Beginning from P1, the neigh-

bor x is step by step further modified in a uni-directional way approaching P2. Finally, an individual is chosen from the generated list which is superior to or almost equally distant from both parents to be included into the next generation. The distance measure used is characterized by the number of disjunctive arcs differently oriented in the two solutions considered. This procedure can be generalized by interchanging the role of the two parents to a bi-directional procedure. In a similar way, mutation can be performed as a *multi-step mutation* (MSM). The generation of neighbors is repeatedly applied in such a way that the subsequently generated individuals move away from the chosen parent. As the neighborhood, both the shift and the pairwise interchange neighborhoods have been considered. For the makespan problem, a refinement is possible by considering the blocks of a critical path, i.e., the corresponding crit-neighborhoods are used. The algorithm has been tested against other genetic algorithms and a simulated annealing on the (10,10) (with $PS = 500$) and (20,5) (with $PS = 100$) benchmark instances from [35]. The MSX-GA was competitive or slightly superior to the other methods.

Yamada and Nakano [130] refined the MSX operator from [129] by combining it with local search which results in the *multi-step crossover fusion* (MSXF) operator (see also [95] in Section 5). The local search included is not a simple descent procedure but a more efficient stochastic procedure, which is a restricted simulated annealing variant. Starting from one of the parents, this individual is stochastically replaced by a rather good solution, where the replacement is done in the direction of the second parent. After a certain number of iterations has been done, the best generated solution is chosen as offspring. The local search works on the critical path of a schedule. If the distance between the chosen parents is too small, a mutation operator called *multi-step mutation fusion* (MSMF) is used, which is based on the same ideas. A comparison with other genetic algorithms has been made on the (10,10) and (20,5) instances from [35]. In contrast to all other variants, only the new GA/MSXF algorithm obtained the optimal values in all of the 10 runs with $PS = 10$ in a CPU time of about 1.5 minutes.

Lin et al. [66] presented parallel genetic algorithms for the problem of minimizing the makespan. Their approach used a direct representation encoding the starting times of the operations, which means that the length of a string is equal to the number of operations. They introduced the *time horizon exchange crossover* (denoted as THX), which works on the schedule level. The crossover selects a cut point which is used as a decision point in the G & T algorithm to exchange information between both schedules. The portion before the cut point is taken from the first parent while the temporal relationships between the operations in the remaining part are taken from the second parent up to a possible extent so as to ensure that always a feasible schedule results. They also introduced a THX mutation based on the blocks of a critical path. In particular, two operations of a block are selected and reversed such that by the G & T algorithm, always a feasible active sc hedule is generated.

The authors compared five different variants of a parallel GA. They used both a *fine-grained GA*, where individuals are spatially arrayed in a specific manner and an individual in the population can only interact with individuals not too far away from them, and an *island-parallel GA* (also denoted as a *course-grained* GA), where a single GA is applied to each sub-population and only at certain times, some individuals may migrate from one sub-population to another one. In particular, they tested two variants of an island-parallel, one

variant of a fine grained and two hybrid variants with $PS \in \{250, 500, 1000, 2000\}$ on the (10,10) problem from [35]. The parallel algorithms found better solutions in shorter time compared to a single-population GA. The number of islands in an island-parallel algorithm had a greater influence than increasing the population size. One of the hybrid variants combining the advantages of both strategies turned out to the best parallel algorithm.

Yamada and Nakano [131] discussed several representations and genetic operators for the job shop problem. In particular, the authors refined their approach from [128]. They noticed that their earlier approach was suitable for small and medium problems but for larger problems, it is necessary to combine a GA e.g. with local search or with the shifting bottleneck procedure. In particular, they incorporated the MSFX operator from [130] into the GA and compared the resulting algorithm on ten difficult benchmark instances. They found that this MSFX-GA outperformed other genetic algorithms at least for the benchmark instances from [35].

Wang and Brunn [119] presented a GA for the job shop problem (and in addition, for the special case of a single machine problem), where a chromosome consists of m sub-chromosomes each of them representing a machine and each sub-chromosome consists of a number of genes equal to the number of operations on the corresponding machine. For the crossover, the authors used a so-called *sequence-extracted crossover* applied to each sub-chromosome, which is basically a one-point crossover, where the part not taken from parent 1 is used in the order as it appears in parent 2. The mutation operator is based on an adjacent pairwise interchange within a sub-chromosome. The population size has been chosen in the range from 10 to 200 in dependence on the problem size. The major termination criterion is a fixed number of generations and in addition, the procedure stops if there is a sufficient improvement in the best fitness. Mainly, the algorithm has been tested on own randomly generated test instances and some instances from the literature (e.g. the 'easy' (6,6) instance from [35]).

Vazquez and Whitley [116] presented a GA and compared it with other algorithms for the job shop problem with minimizing the makespan. The authors used a direct representation containing the current schedule, i.e., it includes a permutation of the jobs on each machine and any operation includes its starting time. Half of the initial population are active schedules and the remaining 50 % are non-delay schedules. For selecting the parents, each individual of the population is chosen once and the second parent is randomly chosen. They applied an order-based crossover and used the G & T algorithm for repairing illegal offspring. After generating the offspring, an iterative improvement algorithm based on the crit-shift neighborhood is applied. Then the worst individual is replaced by the best offspring if it has a better fitness. The authors also reported that the order-based crossover by Davis is superior to the OBX operator.
Computational tests have been made for some benchmark instances from the sets given by Fisher and Thompson [35], Yamada and Nakano [128], Lawrence [61] and Taillard [110]. The presented algorithm using an order-based crossover combined with the Giffler and Thompson repair method was competitive to the other tested genetic algorithms. It has been found that all tested genetic algorithms are well suited for particular classes of benchmark instances, while no algorithm was superior to all other algorithms for all problem classes.

Goncalves et al. [41] gave a hybrid GA in which the chromosome representation is

based on random keys. Solutions are constructed by a procedure that generates parameterized active schedules. A chromosome consists of $2n$ genes describing the priorities of the operations and the delay times. The crossover is a parametrized uniform crossover. Instead of using a classical mutation of particular genes, one or several members of the population are randomly generated from the same distribution as the original population to increase diversity. If the offspring have been generated, they are improved by local search which is based on the neighborhood introduced by Nowicki and Smutnicki [78]. The population size PS is chosen to be equal to twice the total number of operations. Moreover, $P_C = 0.7, P_M = 0.2$ and $NGEN = 400$ have been chosen. The elitist strategy was chosen by inserting the top 10 % of individuals directly into the next generation.
The algorithm has been compared to several genetic algorithms, e.g. by Dorndorf and Pesch [31], GRASP algorithms, the hybrid genetic and the simulated annealing algorithms from [120] and the tabu search algorithm by Nowicki and Smutnicki [78] on the test instances from [35] and [61]. The new algorithm obtained the best known solution in 72 % of the problem instances (only for the (15,15) instances, the tabu search algorithm from [78] obtained better results).

Wang and Zheng [120] suggested a GA using an operation-based representation of an individual with job repetitions. A chromosome is encoded into an active schedule. For generating two offspring, the best individual and a randomly generated individual are chosen. The specific crossover applied is based on partitioning the job set into two subsets and inheriting the genes of one subset of each parent such that a legal offspring results. Mutation is replaced by a simulated annealing procedure, where different neighborhoods have been used. In particular, the pairwise interchange neighborhood is used in 50 % of the neighbor generations while each of the shift and inversion neighborhoods is applied in 25 % of the neighbor generations.
This algorithm has been run with $PS = 20$ on the three benchmark instances from [35] and eight instances from [61]. In nine of the 11 instances, the authors reached the best known value by their GA when it was run 20 times for each instance. These best values obtained in the 20 runs for each instance were superior to the other tested algorithms: simulated annealing, tabu search, the shifting bottleneck procedure and a standard GA.

Park et al. [83] suggested both a single and a parallel GA. They used an operation-based representation (as an unpartitioned permutation with job repetitions). The initial population is determined by using the G & T algorithm for generating individuals describing active and non-delay schedules. The authors tested four crossover operators, three of them resulted from the PPX, the GOX and the GPMX operators. The mutation operator is based on the selection of three different genes and permuting the corresponding numbers arbitrarily, where two variants were considered. For selecting the parents, seed selection (where superior individuals are preferred) and tournament selection were considered. Moreover, the elitist strategy was used, where different elitism sizes were tested. The algorithm has been run with a single population and as a parallel algorithm with separate sub-populations. The authors used an island-parallel algorithm which maintains distinct sub-populations. In addition, at some times individuals can migrate from one sub-p opulation to another one. In particular, the parallel algorithm worked with two or four sub-populations. The algorithm has been tested on several sets of benchmark instances, e.g. from [8, 35, 128]. The authors claimed that their approach has obtained substantial improvements over traditional genetic

algorithms.

Peng and Salim [84] used some slight modifications of the GT-GA by Yamada and Nakano [128]. As the major modification, they used a binary tournament selection for choosing the parents (instead of random pairing applied in [128]). The comparison was performed with the same parameters using the Visual Prolog programming language. For the instance from [35] with $n = m = 10$, the results were slightly better than for GT-GA, but the final makespan values are still rather far away from the optimal value of 930.

Liu et al. [67] presented a hybrid GA which combines a traditional GA with the Taguchi method. It uses an operation-based representation, where each operation has four entries. For applying the genetic operators, a chromosome is partitioned into several sections each of which is considered as a factor of the Taguchi method. A parent is chosen according to roulette wheel selection. For applying a (non-standard) crossover, two proto-offspring are generated from one parent by applying a pairwise interchange of two genes in each section. The Taguchi method is inserted between the crossover and mutation operators. A two-level orthogonal array is used to study a large number of decision variables with a small number of experiments. The Taguchi concept is based on maximizing performance measures, called signal-to-noise ratios (which refer to the mean-square deviation of the objective function). In particular, an appropriate two-level orthogonal array is chosen, two chromosomes are randomly chosen for doing matrix experiments, the fitness values and signal-to-noise ratios of some experiments in the array are calculated and based on the effects on the various factors, an optimal chromosome is generated. For a mutation, a pairwise interchange operation is performed in a chromosome.
This algorithm was run with $PS = 40, P_C = 0.8, P_M = 0.1$ and $NGEN = 5000$. Computational results have been presented for the (10,10) and (20,5) instances from [35], and it has been found e.g. that the average makespan values obtained by their GA were smaller than the average values obtained by the GA from [120] and two other GA variants developed earlier in 20 independent runs for each instance.

Moonen and Janssens [70] presented a GA for the problem $J \,||\, C_{max}$ which uses an operation-based representation and develops a new crossover operator, denoted as *Giffler-Thompson Focused* (GTF) crossover. This operator combines an order-based crossover with a one-point crossover. However, the authors 'direct' the choice of the cut point by the largest conflict set. As a consequence, the number of offspring is not constant but is equal to the size of this conflict set. Mainly, the effect of the new GTF crossover has been tested. To this end, the GA was run with the PPX operator and tested against a variant that uses both the GTF and the PPX crossovers on 30 benchmark instances from [8, 61, 107]. The algorithm was run with $PS = 100, P_C = 0.8, P_M = 0.2$ and $NGEN = 150$ for the first setting. Since the application of the GTF operator creates more offspring, in the second setting the number of generated chromosomes is fixed to 15000 to allow a fair comparison. The combined use of the GTF and PPX operators obtained on average better results than the exclusive use of PPX for all but one of the benchmark instances. However, compared to the best known solutions for these benchmark instances, the presented GA does not perform very well.

Matsui and Yamada [68] discussed a parameter-free GA and tested it on a variety of large benchmark instances from the literature. The proposed algorithm uses a random key permutation representation. The length of the chromosome is $2nm$, where the first part contains the random keys, and the second part contains values controlling the generation of

hybrid (i.e., active or non-delay) schedules by allowing a maximal idle time on the corresponding machine). The crossover is an n-point operator and for one randomly chosen offspring, an inversion mutation is performed. Among the two parents and the two offspring, between one and three individuals are inserted into the new population depending on their fitness values. The authors tested also a parallel GA with a number of sub-populations equal to a power of 2 (with at most 64 sub-populations) on a set of benchmark instances from [8, 107, 110]. The maximal number of fitness evaluations was limited to 1,000,000. I t was observed that the quality of the results increases with the number of sub-populations.

Li and Chen [62] also used an operation-based representation with job repetitions (as in [11]). They considered two permutation crossover operators. The new generation is formed by considering both the parents and the offspring. Unfortunately, results are only discussed for one small instance with five machines and real processing times using $PS = 50$ and $NGEN = 50$.

Abdelmaguid [2] presented a computational study for testing six of the representations of a solution for a job shop problem, namely three processing sequence based and all three algorithm-based representations. The initial population is completely randomly generated. For each representation, the PMX and OX crossovers have been used, both as a one-point and a two-point crossover. As a mutation, the shift, the pairwise interchange, the insertion and a so-called *displacement* (i.e., a block of jobs is selected and shifted to another position in the string) mutations have been used. Tests have been made on 40 instances chosen from several benchmark sets mentioned in Section 3. The machine-based representation obtained the lowest optimality gap (2.55 %), but with the largest computational times. On the other hand, both the random key and preference list representations did not yield competitive results. We only note that a similar discussion of representations of individuals for the job shop problem has been given in [104].

Next, we discuss some genetic algorithms for extensions of the classical makespan minimization job shop problem.

The GA by Falkenauer and Bouffouix [33] used a preference list based representation split into sub-chromosomes containing the operations to be performed on the same machine. The criterion considered penalized earliness and tardiness of the jobs in a non-symmetric way. The crossover is the LOX operator applied independently to each sub-chromosome, while the mutation operator considers inversions of a segment. The algorithm has been tested on three types of problems including 24, 60 or 250 operations. The authors used $PS = 30, P_C = 0.6$ and an inversion mutation with $P_M = 0.33$. They observed a slight superiority of the LOX operator over the PMX operator.

Lin et al. [65] considered job shop scheduling problems with given release dates. They investigated both the deterministic and the stochastic version of such a problem and examined several objectives, namely the minimization of total weighted flow time ($\sum w_i(C_i - r_i)$), maximum tardiness, weighted total tardiness, weighted total lateness, the weighted number of tardy jobs and weighted total earliness plus tardiness. The algorithm is a modification of their algorithm for the static problem and is based on the G & T algorithm, which is used to interpret the offspring. For each individual, a direct representation using the starting time of each operation is used. The crossover is of the THX type (working on the schedule level),

and a mutation performs a pairwise interchange of two operations of a block of the critical path. While the adaptations to the deterministic problem with release dates are straightforward, the stochastic problem is decomposed into a s eries of deterministic problems (such a deterministic problem is generated whenever a new job enters the system).
For the deterministic version, $PS = 50, P_C = 0.6, P_M = 0.1$ and $NGEN = 50n$ were chosen. Both a single population and a parallel GA using 25 sub-populations each of size 20 were considered. The adapted GA's for the deterministic version were tested on 12 benchmark instances against 12 priority rules used for particular objective functions. Both GA performed significantly better than the priority rules. For the stochastic version, the approach was tested under several manufacturing environments with respect to the machine workload, the imbalance of the workload and the tightness of the due dates. Again, the GA were robust and outperformed the priority rules.

Vazquez and Whitley [117] compared genetic algorithms for the job shop problem with given release dates. Basically, the structure of this paper is very similar to that by the same authors for the static problem [116]. It discussed four existing genetic algorithms and presented an algorithm which generates an initial population containing 50 % non-delay schedules and 50 % active schedules. It uses a direct chromosome representation containing the schedule itself. The crossover and mutation operators are of the order-based type, in particular the uniform order-based crossover and the *order-based scramble mutation* (i.e., two positions i and j are selected and the sub-list of elements between these two positions is randomly permuted).
Their order-based Giffler and Thompson (OBGT) GA has been tested against other algorithms (priority rules, a heuristically guided GA and one using the THX operator) on 12 benchmark instances. The authors considered four different sum optimization criteria. The OBGT-GA performed better than the other algorithms, particularly for larger problems.

Moon and Li [69] considered a job shop problem with alternate routings of the jobs, where the objective is to minimize total flow time. This problem is decomposed into two sub-problems: to allocate the operations to a specific machine and to determine a feasible sequence of the operations. A feasible solution is characterized by two strings of a length equal to the number of operations, where the k-th position represents the k-th operation in the subsequent listing of the operations of the jobs J_1, J_2, etc., namely an allocation string (a particular gene gives the machine on which this operation is performed) and a sequencing string (a gene gives the corresponding number of this operation in the sequence). In the allocation string, one- and two-point crossovers are used while in the sequencing string, the PMX, the OX, the PBX, the CX, and a uniform order-based crossover have been used (and in addition an edge recombination crossover, which we do not explain here). Among the mutation operators, the authors used pairwise interchange, shift, inversion and so-called job sequence and gene-sequence mutations. The GA was also combined with a simple tabu search algorithm for the allocation of the operations to the machines. The procedure was illustrated on a small instance with four jobs (each of them consisting of three operations) and six machines.

Sakawa and Mori [101] presented a GA for the job shop problem with triangular fuzzy processing times and trapezium fuzzy due dates. The objective is the maximization of the *minimum agreement index*. An individual is represented by the matrix of the fuzzy completion times for each operation. The initial population is randomly generated in the

form of active schedules. To keep sufficient diversity in the population, the concept of *similarity of individuals* has been introduced and the *degree of similarity* was calculated. In particular, individuals were only added to the population if the degree of similarity is 0.8 or less compared with the other individuals. The specific crossover is organized in such a way that by means of two parents, c offspring are generated trying to eliminating conflicts and from the $c+2$ parents and children, two individuals with the largest minimum agreement index were considered for the next generation. At the time when the crosso ver operator is applied, operations are randomly selected with a specific mutation rate. Tests have been performed for two instances derived from the $(6,6)$ and $(10,10)$ benchmark instances given in [35]. This algorithm has been compared with a simulated annealing algorithm, where the GA turned out to be superior, both in quality and computational times.

Xing et al. [127] presented a fuzzy GA for job shop scheduling problems, where the processing times are modelled as triangular fuzzy numbers and the due dates are modelled as trapezium fuzzy numbers. The objectives were based on a so-called *agreement index*, namely it was the minimization of a linear combination of the minimum agreement and the average agreement indices. They applied a modified form of the *partial schedule exchange crossover* with $P_C = 0.8$. In both parents blocks are chosen (not necessarily starting and ending at the same position). Then the genes of these blocks are interchanged and the proto-offspring are legalized. The mutation operator is of the pairwise interchange type applied with $P_M = 0.2$. The authors run their algorithm on the two instances with $n = m = 6$ and $n = m = 10$ from [35].

Cheung and Zhou [24] presented genetic algorithms for the job shop scheduling problem with sequence-dependent setup times, where a solution is represented as permutations of the jobs on the machines. However, a chromosome is only formed by means of the first operations on the machine and for such a string, the complete solution is obtained by a simulation procedure using two priority rules: the first one gives higher priority to a job with a smaller sum of the processing and setup times, while the second rule gives higher priority to a job which can start its processing earlier. The crossover is a standard two-point crossover. The authors used an adaptive mutation probability P_M which changes according to the diversity in the population. This algorithm has been tested on own randomly generated instances with up to $n = m = 20$. It has been found that the GA was superior to a heuristic approach given by Choi and Korkmaz [25].

Vela et al. [118] considered genetic (and local search) algorithms for the job shop problem with sequence-dependent setup times. A chromosome is a permutation of the operations (described by job repetitions) from which an active schedule is generated. The algorithm uses the *job order crossover* (JOX) introduced in Bierwirth [11] with probability $P_C = 1$. The procedure was combined with local search, where three neighborhoods have been used which are based on the blocks of a critical path. In the first neighborhood, an adjacent pairwise interchange of the first or last two operations of a block is allowed. The second one is a restricted shift neighborhood, where an operation of a critical path is directly shifted before the first or after the last operation of this block. The third one is more complex, where at most six neighbors result from a critical block by considering the first two operations of a block together with the predecessor operation on this machine or the last two operations of a block together with the successor operation on this machine. These neighborhoods have been generalized further to the problem with sequence-dependent setup

times (denoted as N_1^S, N_2^S and N_3^S) maintaining feasibility and satisfying a necessary condition for an objective function improvement. E.g., N_1^S contains all neighbors obtained by reversing one arc of a critical path so that a feasible schedule results which might lead to an improvement.
The algorithm has been tested on the benchmark instances from [17] and some new harder instances derived from those given by [8] for the problem without setup times. The union $N_1^S \cup N_2^S \cup N_3^S$ turned out to be the best neighborhood (if only one of them is used, then N_1^S was the best structure). The hybrid algorithm has obtained the best known solutions in 14 of the 15 instances considered (and six of the best known values were improved), where 30 runs were made for each instance. The authors have also generated new benchmark instances available at http://www.aic.uniovi.es/tc/spanish/repository.htm.

Xie [126] presented an algorithm for the job shop problem with batch allocation and minimizing the makespan. This means that batch sizes have to be determined and sequence-dependent setup times arise if a new batch starts. The derived algorithm uses an indirect two-dimensional representation of a solution, namely two sub-chromosomes to represent the dispatching rules for finding the operation sequence and the number of batches, respectively, which are both described by integers of a limited range. For describing the dispatching rules, a 3-dimensional binary string is used and for describing the batch size, a 2-dimensional binary string is used. The chromosome is transformed into a non-delay schedule. By means of the objective function value, the fitness value is determined via a linear scaling. The population size has been set to $PS = 50$. A one-point crossover is applied with $P_C = 0.95$, and a mutation changing the value of a binary variable is applied with $P_M = 0.05$. The algorithm is of the steady state type. In particular, the probability of replacing individuals of the old generation is 0.9. The algorithms has been tested on three benchmark instances, among them the $(10,10)$ and $(20,5)$ instances from [35].

Zhang and Wu [136] presented an immune algorithm for the problem of minimizing total tardiness, which is based on bottleneck jobs. To describe the characteristic information on bottleneck jobs, a fuzzy interference system is used to transform human knowledge into the bottleneck characteristic values. The encoding scheme of the solutions is based on a priority list of the operations. The crossover is of the LOX type and applied to the best individual together with a randomly selected individual. The mutation operator is based on a pairwise interchange of two operations. Then the extracted bottleneck information is used as an immune operator with the goal of increasing the convergence speed.
The algorithm has been tested with $PS = 30, NGEN = 20, P_C = 0.8, P_M = 0.5$ and an immune probability of 0.8 on 10 instances with up to 100 jobs and 20 machines with $n \times m \leq 500$. The authors claimed that their algorithm is better than standard genetic algorithms for medium and large problems.

6.1 FLEXIBLE JOB SHOPS

Finally, we briefly discuss some applications of genetic algorithms to flexible of hybrid job shop (FJS) problems.

Ghedjati [38] presented a GA for the FJS makespan minimization problem with unrelated parallel machines (which has similarities to algorithm P-GA from [31] but applied to this more general problem). In particular, Ghedjati presented a procedure called *heuristic mixing method*. For coding a solution, the author used two strings with a length equal to

the number of operations. The k-th gene of the first string denotes the priority rule for choosing the k-th operation, while the corresponding gene in the second string denotes a heuristic rule for choosing a machine. By means of two chosen parents, two offspring are generated, where a one-point crossover and the pairwise interchange mutation operator are applied. The algorithm has been tested on one instance from the literature and several own randomly generated instances. It has been found that a population size of $PS = 50$ worked better than smaller population sizes.

Kacem et al. [55] presented two approaches for the FJS problem (with total and partial flexibility), where one is an evolutionary approach controlled by an assignment model. The objectives are the minimization of the makespan and balancing the workload on the machines. The authors used an operation-machine coding in such a form that looking at a row, the execution of operations is described while looking at a column, the operations on each machine with the starting and completion times are obtained. The crossover is arranged in such a way that the memberships of the new individuals to the assignment schemata are maintained. In addition, two mutations were introduced, one mutation that tries to reduce the effective processing time of a job and another mutation trying to balance the workload on the machines. The algorithm has been applied with $PS = 100, P_C = 0.88, P_M = 0.12$ and $NGEN = 500$.

Zhang and Gen [135] considered the FJS problem with the objectives of minimizing the makespan, the maximum workload of a machine and the total workload of all machines. For this problem, they presented a multi-stage operation-based approach (denoted as moGA), where a chromosome has the following structure. The operations are listed subsequently for the jobs $J_1, \dots, J_n$. Each sub-chromosome includes a gene for any operation, where the entry is the machine of the corresponding stage on which the corresponding operation is performed. As a consequence, it can be used for applying standard crossover and mutation operators. For the multi-stage operation-based description, a permutation chromosome is encoded, and a decoding procedure is given which constructs from the permutation chromosome the resulting schedule. The initial population has been randomly generated with $PS = 100$. They applied a one-point crossover with $P_C = 0.6$ and a mutation with $P_M = 0.3$. The algorithm has been tested on the two instances from [55], where the authors also considered both partial and total flexibility. For these two instances, the new algorithm obtained better results than the algorithm from [55] and a classical GA.

Gao et al. [37] presented a hybrid GA for the FJS problem with the three objectives of minimizing the makespan, the maximal machine workload and the total workload. For representing a solution, the algorithm uses two vectors: a machine assignment vector and a operation sequence vector. The crossover operator is the LOX operator applied to the string of operation sequences, and the algorithm considers also a uniform crossover. Mutation is applied both to the operation sequence and the machine assignment vectors. For the operation sequence string, they apply a pairwise interchange mutation while for the machine assignment, a mutation means that another machine is chosen for this operation. In addition, they apply a so-called *immigration mutation* which means that a number of new individuals are generated from the same distribution used to generate the initial population. In each generation, a number of best individuals are chosen as parents while the remaining parents are chosen by roulette wheel selection. For selecting the new generation, both the parent and offspring can be chosen with the same chances. Dependent on the problem size, the

population size PS ranges from 300 to 3000. All crossover and mutation operators are applied with $P_C = P_M = 0.4$. To improve the convergence, this procedure is combined with a variable neighborhood descent procedure. The neighborhoods are defined in such a way that either one or two operations are selected from the critical path and shifted (i.e., basically one or two moves in the crit-shift neighborhood are allowed). The algorithms have been tested on 181 benchmark instances from various data sets from the literature for the flexible job shop problem (with up to 30 jobs and 18 machines).

Lei [60] presented an efficient decomposition-integration genetic algorithm (denoted as DIGA) for the FJS problem with fuzzy processing times to minimize the maximum fuzzy completion time. This GA uses a two-string representation. The first string is used for sequencing the jobs while the second one is used for the assignment of the machines. To obtain a feasible schedule, the first string is transformed into a list of operations and then by means of the second string, a machine is assigned from left to right. The initial population is randomly determined and decomposed into two sub-populations (for job sequencing and machine assignment, respectively). Binary tournament selection is applied. For the job sequencing part, generalizations of the GPMX and the PPX crossovers are considered while for the machine assignment part, a two-point crossover is considered. The pairwise interchange operator is used for an mutation.
This GA has been run with a population size of $PS = 100, P_C \in \{0.65, 0.7, 0.8\}, P_M = 0.1$ and $NGEN = 1000$. It has been compared with a previous hybrid algorithm combining particle swarm optimization with simulated annealing and another GA on four (10,10) instances. The authors observed that the new DIGA performed better than the other two algorithms both in terms of the average results and the worst solutions obtained.

7 OPEN SHOP PROBLEMS

Fang et al. [34] suggested an approach which combines a GA with heuristic rules for the schedule construction. In a first variant (denoted as JOB+OP), a solution is directly encoded into a string of a length equal to twice the number of operations, where two subsequent genes refer to one operation: e.g. 2, 4 means: schedule the 2nd untackled operation of the 4th uncompleted job. In a second variant (denoted as FH), an operation from the currently considered job is chosen by using eight fixed heuristic rules (e.g. SPT and LPT based heuristics). Finally, they considered a third variant (denoted as EHC for evolving heuristic choice, where the chosen heuristic rule may vary) where the subsequent genes 3, 4 from the chromosome mean: choose the third heuristic for scheduling an operation of the fourth uncompleted job). The elitist strategy, a uniform crossover and a mutation based on the pairwise interchange of two genes were used.
The algorithm has been run with $PS = 200, P_M = 0.5$ and $NGEN = 100$. The probability of applying a crossover started with $P_C = 0.8$ and was reduced in steps of 0.0005 until $P_C = 0.3$ The algorithm has been tested on the benchmark instances from [110] using ten runs for each instance. By their tests, they discovered one new best known solution for a (7,7) and a (10,10) instance.

Khuri and Miryala [57] presented three variants of a GA for the problem $O \mid\mid C_{max}$: a permutation GA (PGA), a hybrid GA (HGA) and a selfish gene (FGA) algorithm. The first variant uses an operation-based representation and applies roulette wheel selection, uni-

form crossover and pairwise interchange mutation. The hybrid GA uses an operation-based representation but described by job repetitions. This variant incorporates the LPT heuristic into the schedule construction. The selfish gene algorithm does not rely on crossover and it does not model a particular population. Instead of this, it works with a virtual population which models the gene concept via statistical measures. The representation of a solution is described by a string of length $2nm$.
The three algorithms have been tested on the benchmark instances from [110]. For the first two variants, $P_C = 0.6$ and $P_M = 0.1$ have been chosen together with $PS = 200, NGEN = 500$. For the large instances with $n = m \in \{15, 20\}$, HGA delivered the best results often equal to the best known solutions. However, for the instances with $n \in \{7, 10\}$, all three variants were not able to reach the best known solutions (in 100 runs for each instance). For the small instances with $n \in \{4, 5\}$, PGA and FGA mostly reached the best values and outperformed the variant HGA.

Prins [89] gave a GA for the problem $O \,||\, C_{max}$. It uses the operation code for describing an individual, where a chromosome is defined as a sequence of the operations. This algorithm suggests to include only individuals with different makespan values into the population. The selection of individuals is done by a simple ranking mechanism. The crossover operators used in the test are a one-point crossover, the OX and the LOX crossover, where the LOX/OX operators are clearly superior. After applying the genetic operators, Prins reordered the chromosomes of every new schedule after the computation of the makespan value in increasing order of the starting times of the operations. The mutation operator is based on a shift or a pairwise interchange mutation (tested both for fixed and variable mutation rates). The major termination criterion is a maximal number of crossovers performed. Before making detailed tests, Prins [89] suggested several improvements, e.g., changes in generating the initial population, the inclusion of the *probing technique* (where an independent procedure is applied to the same parents to generate four active schedules) and the replacement of the mutation operator by improvement heuristics.
Many variants of the algorithms have been tested with $PS \in \{30, 60, 100\}$. A larger population gave only slightly better results with longer computational times so that finally $PS = 30$ has been chosen. Tests have been made for the benchmark instances from [14, 43, 110]. Compared with other algorithms, the GA yielded competitive results, even when compared with branch and bound algorithms (which cannot solve all instances with $n = m = 7$). Prins concluded that the excellent performance of his algorithm has the following four reasons: generation of active schedules only, use of small populations with distinct makespan values, use of chromosome reordering to increase the efficacy of crossover, and good heuristics used for generating the initial population and applied in the mutation operator.

Liaw [64] presented a hybrid GA for the problem $O \,||\, C_{max}$. The initial population generates one individual by a particular dispatching rule and the remaining ones are generated by the G & T algorithm such that they represent active schedules. For the representation of solutions, the operation code is used. The selection scheme uses the elitist strategy combined with 2/4 selection. It has been found that the convergence of the algorithm has been accelerated by the fact that the objective function values of the best and worst individuals are non-increasing. Concerning the crossover operator, based on initial tests, the LOX crossover has been chosen which turned out to be superior to e.g. the PMX, the OX or the CX operators. For the mutation operator, a union of the shift and pairwise

interchange neighborhoods is applied to the sequence of operations, where each of the sub-neighborhoods is used with the same probability. If new offspring have been generated, a basic tabu search is applied. The neighborhood structure is based on blocks of operations on a critical path. In particular, several neighborhood types have been considered, where one, two or three arcs may be reversed in their orientation. The tabu search procedure stops if a maximal number of iterations has been performed (or an optimal solution has been found). For the parameters of the GA, a population size $PS = 121$, a maximal number of generations $NGEN = 150$, a crossover probability $P_C = 0.6$ and a mutation probability $P_M = 0.2$ as well as a maximal number of 125 iterations in the tabu search procedure have been chosen. The algorithm developed has been tested on randomly generated instances and on the benchmarks sets by Taillard [110] and Brucker et al. [14]. It has been found that this GA outperformed other existing algorithms from the literature, and some of the benchmark instances have been solved to optimality for the first time.

Senthilkumar and Shahabudeen [103] presented a GA for the problem $O \,||\, C_{max}$. This algorithm uses a chromosome of a length equal to the number of operations which is split into sub-chromosomes each of them representing the machine sequence of a job. The initial population is randomly generated. The crossover applied with $P_C = 0.8$ is a two-point crossover of the OX type within a sub-chromosome. A mutation is performed as a pairwise interchange of two genes of a sub-chromosome applied with $P_M = 0.2$. Unfortunately, this algorithm has been compared only with a particular heuristic (which does not perform very well for the open shop problem) on small instances with up to $n = m = 4$. In addition, nothing is said about the parameters PS and $NGEN$.

Andresen et al. [6] presented a GA for the problem $O \,||\, \sum C_i$. This algorithm uses the *rank matrix* to encode an individual. The initial population is formed by means of the constructive algorithms given in [13]. The parents for generating the offspring are randomly selected. The crossover and mutation operators are rather different from those used in other algorithms so that we explain them here more in detail. The crossover exchanges the ranks of a randomly chosen set of operations. This yields two (usually infeasible) proto-offspring. Then the relative order of the remaining operations from the parent is maintained, and both parts (i.e., the relative orders of the operations with exchanged ranks and the relative orders of the remaining operations) are now combined into a feasible rank matrix. To this end, a linear order is put on the operations in such a way that a smaller number in the linear order indicates that the rank of this operation is not greater than the rank of an operation with a larger number (i.e., this linear order describes the sequence in which the operations are considered to assign the smallest possible rank). In the case of ties, a lexicographical order is applied with the following exception: The operations for which the exchange of the ranks takes place get the smallest possible numbers among all operations with the same rank (this guarantees that the operations with a new rank have a 'higher priority'). Finally, from this linear order of operations, the resulting rank matrix is constructed.

To illustrate the crossover, we consider the individuals described by the rank matrices

$$R_1 = \begin{pmatrix} \mathbf{1} & 3 & 4 \\ \mathbf{3} & \mathbf{1} & 2 \\ 4 & \mathbf{2} & 3 \end{pmatrix} \quad \text{and} \quad R_2 = \begin{pmatrix} \mathbf{2} & 4 & 1 \\ \mathbf{1} & \mathbf{2} & 3 \\ 4 & \mathbf{3} & 2 \end{pmatrix}.$$

Assume that the operations $(1,1),(2,1),(2,2)$ and $(3,2)$ marked in bold face above are

chosen for performing the crossover, i.e., the ranks of the chosen operations in R_1 and R_2 are exchanged. This yields the two proto-offspring

$$PO_1 = \begin{pmatrix} \mathbf{2} & 3 & 4 \\ \mathbf{1} & \mathbf{2} & 2 \\ 4 & \mathbf{3} & 3 \end{pmatrix} \quad \text{and} \quad PO_2 = \begin{pmatrix} \mathbf{1} & 4 & 1 \\ \mathbf{3} & \mathbf{1} & 3 \\ 4 & \mathbf{2} & 2 \end{pmatrix}.$$

and the corresponding linear orders of the operations

$$LO_1 = \begin{pmatrix} \mathbf{2} & 6 & 8 \\ \mathbf{1} & \mathbf{3} & 4 \\ 9 & \mathbf{5} & 7 \end{pmatrix} \quad \text{and} \quad LO_2 = \begin{pmatrix} \mathbf{1} & 8 & 3 \\ \mathbf{6} & \mathbf{2} & 7 \\ 9 & \mathbf{4} & 5 \end{pmatrix}$$

from which the resulting two offspring can be constructed:

$$OFF_1 = \begin{pmatrix} \mathbf{2} & 4 & 5 \\ \mathbf{1} & \mathbf{2} & 3 \\ 5 & \mathbf{3} & 4 \end{pmatrix} \quad \text{and} \quad OFF_2 = \begin{pmatrix} \mathbf{1} & 3 & 2 \\ \mathbf{2} & \mathbf{1} & 4 \\ 4 & \mathbf{2} & 3 \end{pmatrix}$$

To illustrate the use of the mutation operator, consider the individual described by the rank matrix

$$R = \begin{pmatrix} 1 & 3 & 4 \\ \mathbf{3} & 1 & 2 \\ 4 & 2 & 3 \end{pmatrix}.$$

Assume that operation $(2,1)$ is chosen for applying a mutation. We have $r_{21} = 3$ (drawn in bold face above). There are four possibilities for performing a mutation in this case, namely since the largest rank in row 2 and column 1 is $k = r_{31} = 4$, we can set the rank $r_{21} \in \{1,2,4,5\}$. Let us consider the case $r_{21} = k^* = 1$. We get the subsequent proto-offspring PO, then the corresponding linear order LO of the operations (note that the mutated operation has now a smaller number than the other operations $(1,1)$ and $(2,2)$ with rank 1). Finally, assigning the smallest possible ranks to the operations according to LO, we obtain the offspring OFF:

$$PO = \begin{pmatrix} 1 & 3 & 4 \\ \mathbf{1} & 1 & 2 \\ 4 & 2 & 3 \end{pmatrix} \implies LO = \begin{pmatrix} 2 & 6 & 8 \\ \mathbf{1} & 3 & 4 \\ 9 & 5 & 7 \end{pmatrix} \implies OFF = \begin{pmatrix} 2 & 4 & 5 \\ \mathbf{1} & 2 & 3 \\ 5 & 3 & 4 \end{pmatrix}.$$

It has been found that the sum of the crossover and mutation probabilities should be around 1. In particular, it has been found that larger mutation than crossover probabilities should be used, e.g. $P_M \in \{0.6, 0.8\}$. For selecting the individuals for the new generation, the elitist strategy is used in combination with 2/4 selection. Population sizes between 5 and 100 have been tested.

For the tests, short runs with 30000 generated solutions were considered and compared to a simulated annealing variant. It has been observed that the problems become more difficult with increasing ratio n/m.

For problems with $n < m$, the best solution has often an objective function value either equal to close to the lower bound. For these easier problems, large population sizes (50 or

100) can be chosen. However, for problems with $n > m$, it turned out that the population size should be small (e.g. 10) and the mutation rate should be large ($P_M = 0.8$). Percentage improvements over the constructive algorithms are typically small. For this reason, it is necessary to start with a good initial population (since the number of improvements during the search is large so that a larger number of generations is required). For further algorithmic developments, the focus should be on problems with $n > m$.

Naderi et al. [73] presented a study on the problem $O \mid\mid \sum T_i$. A major part of this paper (Section 4) is dealing with the development of a GA. A solution is represented as a permutation of the operations. To decode the permutation list, a direct (from the list, the resulting semi-active schedule is constructed) and an indirect (from the list, a non-delay schedule is constructed) decoding procedure have been used. The initial population is randomly determined. As the selection mechanism, both ranking and tournament selection have been considered. As crossover operators, the SBOX operator from [99], a parametrized uniform crossover (using random keys), a one-point crossover, a specific modification of a two-point crossover and a mixture of two variants have been tested. As a mutation, the generation of a shift, a pairwise interchange, a inversion neighbor and the application of a mutation denoted as *giant leap* (the jobs are randomly chosen and they are placed on randomly determined positions) were tested. The elitist strategy is used in the form that a pre-specified number of chromosomes with the smallest tardiness values is directly copied into the next generation The stopping criterion is a time limit proportional to the product nm.
Based on initial tests, $PS = 40$, tournament selection, the SBOX crossover, and the shift mutation with $P_M = 0.3$ have been chosen. The algorithm has been tested on own small instances and on the three sets by Taillard [110], Gueret and Prins [43] and Brucker et al. [14]. The authors also used a hybrid variant incorporating local search. From the tests, they found that both a variable neighborhood search algorithm and the hybrid GA with an indirect encoding yielded the best results.

Kokosinski and Studzienny [59] presented a hybrid GA for the problem $O \mid\mid C_{max}$. A solution is represented by a permutation list of the operations described by job repetitions, and two greedy LPT-based heuristics are suggested to decode a chromosome into a feasible schedule. The binary tournament selection scheme is used. For the crossover, the LOX operator is applied and as a mutation, both the pairwise interchange and inversion operators are considered. In addition to a single population, an island-parallel GA with migration was tested. Migration is used in such a way that migrated individuals remain a member of their original sub-population. The major parameters were $PS = 300, P_M = 0.75$ and $NGEN \in \{500, 1000\}$, and the parallel GA considered three sub-populations.
The different variants have been tested on own randomly generated instances with $n = m \in \{20, 25, 30, 40, 50\}$. It appeared that the inclusion of the two greedy heuristics improved the efficiency both of the single-population GA and the parallel GA. However, in their experiments, the parallelization did not lead to improvements over the use of a single population.

Andresen et al. [7], compared the algorithm from [6] with a simulated annealing algorithm for the problem $O \mid r_i \geq 0 \mid \sum w_i T_i$ and several of its special cases. For the tests, a set of 7,680 instances have been used for particular variants of generating the release dates, processing times, due dates and job weights. The instances are based on the generation of test instances described in [13] using the random generator from Taillard [110]. The comparison has been made for short runs (30000 generated solutions) and longer runs

(200000 generated solutions). The detailed computational tests confirmed and extended the results from [6]. It turned out that the GA was superior to simulated annealing for the problems with $n \leq m$, while the simulated annealing algorithm was better for the problems with $n > m$. In addition, the recommended population size for the GA depends on the ratio of n and m. For the easier problems with $n \leq m$, a larger population size is preferred, while for the difficult problems with $n > m$, a large population size is not competitive (when fixing the same number of generated solutions).

Harmanani et al. [45] presented a parallel GA for the problem $O \,||\, C_{max}$ with genetic operators that combine the use of *deterministic* and *random* moves. A chromosome represents an operation-based description followed by a *scratch area*. The initial population is completely randomly generated, and roulette wheel selection is applied. The genetic operators are either randomly or deterministically applied which is controlled by a probability. In order to apply deterministic moves, the authors introduce a '*Reduce Gap*' operation which tries to reduce large idle times on a machine by pairwise interchanges of operations. Thus, the operation looks at pairs of values that indicate positions in the chromosome to be interchanged. This information is communicated with the genetic operators via the scratch area of the chromosome. As mentioned before, the mutation is based on the pairwise interchange of two operations, while the authors considered two crossover operators, one is based on a uniform crossover ensuring feasibility and the second one used the information stored in the last gene of the scratch area (i.e., it uses the position determined by the 'Reduce Gap' operator which basically indicates the position at which a schedule loses possibly its quality). These components are incorporated into a parallel GA allowing migration. In particular, Harmanini et al. [45] considered a hybrid variant of an island-parallel and a course-grained parallel GA, based on a SPMD (single program multiple data) *master-slave model*.
Based on detailed initial tests, the authors finally fixed $PS = 400, NGEN = 1000, P_C = 0.25$ and $P_M = 0.75$. Moreover, 60 % of the moves were fixed as deterministic ones. The algorithm has been tested on the 60 benchmark instances from [110]. Small problems could be solved to optimality, while for the larger instances the deviation from the lower bound is stated (however, we recall that the Taillard instances are not particularly hard for the open shop problem).

Palacios et al. [82] considered the open shop problem with uncertain processing times, which are modelled as triangular fuzzy numbers. The objective is to minimize the expected makespan. The authors used an operation-based representation which is decoded either into a semi-active or an active schedule. Parents are randomly selected. Among the crossover operators, PMX, LOX and PBX were considered. For a mutation, both the shift and pairwise interchange operators are used. The individuals for the new generation are chosen according to 2/4 selection.
The test instances have been obtained by modifying the instances from [110] with $n = m \in \{4,5,7,10\}$. These new instances are available from http://www.aic.uniovi.es/tc/spanish/repository.htm. After initial tests, decoding into active schedules is preferred, the PMX operator (but the LOX operator produced similar results) with $P_C = 0.7$ and the shift mutation operator with $P_M = 0.05$ were applied. A computational comparison has been made with a dispatching rule (denoted as DS/LRPT) adapted to fuzzy durations, where the GA is superior.

8 USE OF THE PROGRAM PACKAGE 'LISA - A LIBRARY OF SCHEDULING ALGORITHMS'

There exist several program packages for solving scheduling problems. In this section, we briefly sketch the use of the program package 'LiSA - A Library of Scheduling Algorithms' developed during the last decade at the Otto-von-Guericke-University Magdeburg. This package has been particularly developed for the exact and heuristic solution of shop scheduling problems.

As far as genetic algorithms are concerned, the implemented algorithm is the algorithm described in [6] for solving open shop scheduling problems. Usually, this algorithm generates a randomly determined initial population, but the user can also include good starting solutions by selecting appropriate constructive heuristics for its generation.

This GA can be controlled by the following parameters:

- **population size**;
- **population initialization**: here it is settled which types of sequences are randomly generated, namely **RANDOM ORDER** (individuals are generated the schedules of which can be active or non-delay), **ACTIVE DISPATCH** (individuals are generated the schedules of which are exclusively active ones) or **NON-DELAY DISPATCH** (individuals are generated the schedules of which are exclusively non-delay schedules).
- **number of generations**;
- **random seed for the random generation of the initial population**;
- **crossover probability**;
- **mutation probability**;
- **local improvement steps**: after applying crossover and/or mutation, one can apply simple local search within a particular neighborhood. This parameter settles the number of iterations applied to any new individual;
- **local improvement**: this parameter settles the neighborhood in the local search routine applied to the generated individuals. Possible main options are the adjacent pairwise interchange, the pairwise interchange and the shift neighborhood. In addition, one can use the restriction of the latter two neighborhoods to the corresponding crit-neighborhoods (which is of interest for C_{max} problems).

For a more detailed description, see the handbook of the program package [5].

9 CONCLUSION

In this chapter, we presented an overview on some genetic algorithms developed for shop scheduling problems within the last 30 years. We emphasized some common and different features of the particular algorithms and sketched the main results obtained. In this

survey, we did not discuss other population-based methods such as ant colony algorithms, particle swarm optimization or differential evolution algorithms (which is behind the scope of this review).

While for the permutation flow shop problem a solution is usually represented as a permutation (i.e., a job sequence), there exist several representations for job shop problems, where it is not a priori clear which representation is the best. Since it has been often observed that genetic algorithms may converge slow, the combination of a GA with other heuristics often improves the results. For instance, this can be a combination with other metaheuristics, where mutation is extended or replaced by a neighborhood search. For makespan problems, one can also use specific neighborhoods based on blocks of a critical path. In addition, path relinking turned out to become a successful tool for improving a GA. This means that the landscape between two parents is explored to find better solutions on a path between them. This strategy can also be used for exploring the vicinity of one offspring and to 'move away' from this offspring. A similar idea has been used by Werner [125] in form of the path algorithms which explore paths of different lengths in dependence on the behavior of the objective function value. In contrast, path relinking strategies are based on various distance measures between solutions.

From the discussion in the previous sections, it follows that the range of the parameters characterizing a GA is very large. For this reason, when looking at a new problem, usually intensive tests and a calibration of a designed algorithm is required. In addition, fixing suitable parameters often depend on the type of problems or the allowed time for running a particular algorithm. Some of the algorithms generate 20000 - 50000 individuals while other GA generate more than 1,000,000 individuals. In addition, when calibrating a GA, one should decide whether short, medium or long runs are used since this has e.g. influence on the population size. In addition, when calibrating a GA, one should decide whether short, medium or long runs are used since this has e.g. influence on the population size. However, even when having calibrated a GA under particular conditions and one wishes e.g. to double the allowed time, the question is what is better: to increase the population size, to increase the number of generations of the GA or to use the additional time for incorporating refined local search procedures. Practically, in all situations, the use of some specific constructive algorithms for generating a part of the initial population can be recommended. However, during the run of the algorithm, diversity should be maintained e.g. by incorporating some random solutions, or perform a kick as in iterated local search, etc.

For the comparison with other algorithms, one should allow every algorithm to evaluate the same number of feasible solutions (or even assign the same time limit to each algorithm). When generating the same number of solutions, a GA often requires more time than a corresponding metaheuristic algorithm such as e.g. simulated annealing (since the management of the population, the selection mechanism and the use of the genetic operators are more time consuming than a single neighbor generation with a simple acceptance criterion). While for flow and job shop problems, many genetic and hybrid algorithms are available, which are often able to generate the best known solutions, for open shop problems not so many genetic algorithms are available, most of the them for the makespan problem. Here further algorithms are required taking into account that branch and bound algorithms can solve exactly only hard instances of a (very) small size. While for makespan problems, the focus is on instances with $n = m$, for the sum criteria, problems with $n > m$ are essentially

harder, and further constructive and iterative algorithms are needed in this case.

Since the first presentation of evolutionary algorithms for scheduling problems about 30 years ago, substantial improvements in the solution quality have been obtained. It can be expected that also in the future, further developments and refinements in the area of GA will be observed. Probably, the major focus will be on the clever combination with other contemporary strategies to bundle the strengths of the particular algorithms.

ACKNOWLEDGEMENTS

The author is grateful to Julia Lange for her contributions in the preparation of the figures.

References

[1] Ablay, P. Optimieren mit Evolutionsstrategien. Reihenfolgeprobleme, nichtlineare und ganzzahlige Optimierung. *Ph.D. Thesis*, University of Heidelberg, 1979.

[2] Abdelmaguid, R. Representations in genetic algorithm for the job shop scheduling problem: A computational study. *J Software Engineering & Applications*, 2010, 3, 1155 - 1162.

[3] Aldowaisan, T.; Allahverdi, A. New heuristics for no-wait flowshops to minimize makespan. *Comp Oper Res*, 2003, 30, 335 – 344.

[4] Allahverdi, A.; Ng, C.T.; Cheng, T.C.E.; Kovalyov, M. A survey of scheduling problems with setup times or costs. *Eur J Oper Res*, 2008, 187, 985 – 1032.

[5] Andresen, M.; Bräsel, H.; Engelhardt, F.; Werner, F. LiSA - A library of scheduling algorithms, Handbook for Version 3.0. *TR 10-02, FMA, OvGU Magdeburg*, 2010, 107 pp.

[6] Andresen, M.; Bräsel, H.; Mörig, M.; Tautenhahn, T.; Tusch, J.; Werner, F. Simulated annealing and genetic algorithms for minimizing mean flow time in an open shop. *Math Comp Modell*, 2008, 48, 1278–1293.

[7] Andresen, M.; Bräsel, H.; Plauschin, M.; Werner, F. Using Simulated annealing for open shop scheduling with sum criteria. In: Cher Ming Tan (ed.), *Simulated Annealing*, In-Teh, 2008, 49 – 76.

[8] Applegate, D.; Cook, W. A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, 1991, 3, 149 – 156.

[9] Basseur, M.; Seynhaeve, F.; Talbi, E. Design of multi-objective evolutionary algorithms: Application to the flow-shop scheduling problem. In: Congress on Evolutionary Computation (CEC'2002), 2002, 2, 1151 – 1156.

[10] Beasley, D.; Bull, D.R.; Martin, R.R. An overview on genetic algorithms: Part I, Fundamentals. *University Computing*, 1993, 15, 58 – 69.

[11] Bierwirth, C. A generalized permutation approach to job shop scheduling with genetic algorithms. *OR Spectrum*, 1995, 17, 87 – 92.

[12] Bierwirth, C.; Mattfeld, D.C.; Kopfer, H. On permutation representations for scheduling problems, *in: 4th PPSN*, 1996, 310 – 318.

[13] Bräsel, H.; Herms, A.; Mörig, M.; Tautenhahn, T.; Tusch, J.; Werner, F. Heuristic constructive algorithms for open shop scheduling to minimize mean flow time. *Eur J Oper Res*, 2008, 189, 856 – 870.

[14] Brucker, P.; Hurink, J.; Jurisch, B,; Wöstmann, B. A branch & bound algorithm for the open-shop problem. *Discr Appl Math*, 1997, 76, 43 – 59.

[15] Brucker, P.; Hurink, J.; Werner, F. Improving local search heuristics for some scheduling problems - I. *Discr Appl Math*, 1996, 65, 97 – 122.

[16] Brucker, P.; Hurink, J.; Werner, F. Improving local search heuristics for some scheduling problems. Part II. *Discr Appl Math*, 1997, 72, 47 – 69.

[17] Brucker,P.; Thiele, O. A branch and bound method for the general-job shop problem with sequence-dependent setup times. *OR Spektrum*, 1996, 18, 145 – 161.

[18] Campbell, H.G.; Dudek, R.A.; Smith, M.L. A heuristic algorithm for the n job, m machine sequencing problem. *Management Sci*, 1970, 16, 630 – 637.

[19] Carlier, J.; Neron, E. An exact method for solving the multiprocessor flowshop. *RAIRO Operations Research*, 2000, 34, 1 – 25.

[20] Chang, P.C.; Chen, S.H.; Liu, C.H. Sub-population genetic algorithm with mining gene structures for multiobjective flowshop scheduling problems. *Expert Systems with Applications*, 2007, 33, 762 – 771.

[21] Chang, P.-C.; Liu, C.-H.; Fan, C.-Y. A depth-first mutation-based genetic algorithm for flow shop scheduling problems, *International Conference on Hybrid Information Technology*, 2006, 1, 25 – 32.

[22] Chen, J.-S.; Pan, J.C.-H.; Lin, C.-M. A hybrid genetic algorithm for the re-entrant flow-shop scheduling problem. In: Levner, E. (ed), *Multiprocessor scheduling: Theory and Applications* INTECH, 2007, 154 – 166.

[23] Chen, C.-L.; Vempati, V.S.; Aljaber N. An application of genetic algorithms for flow shop problems. *Eur J Oper Res*, 1995, 80, 389 – 396.

[24] Cheung, W.; Zhou, H. Using genetic algorithms and heuristics for job shop scheduling with sequence-dependent setup times. *Ann Oper Res*, 2001, 107, 65 – 81.

[25] Choi, I.C.; Korkmaz, O. Job shop scheduling with separable sequence-dependent setup times. *Ann Oper Res*, 1997, 70, 155 – 170.

[26] Dannenbring, D. An evaluation of flow shop sequencing heuristics. *Management Sci*, 1977, 23, 1174 – 1182.

[27] Davis, L. Job Shop scheduling with genetic algorithms. In: Grefenstette (ed.), *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*. Lawrence Erlbaum: Hillsdale, NJ, 1985, 136 – 140.

[28] Della Croce, F.; Tadei, R.; Volta, G. A genetic algorithm for the job shop scheduling problem. *Comp Oper Res*, 1995, 22, 15 - 24.

[29] Demirkol, E.; Mehta, S.; Uzsoy, R. Benchmarks for shop scheduling problems. *Eur J Oper Res*, 1998, 109, 137 – 141.

[30] Dhingra, A.; Chandna, P. Hybrid genetic algorithm for multicriteria scheduling with sequence dependent setup time. *International Journal of Engineering*, 3, 2010, 510 – 520.

[31] Dorndorf, U.; Pesch, E. Evolution based learning in a job shop scheduling environment. *Comp Oper Res*, 1995, 22, 25 – 40.

[32] Etiler, O.; Toklu, B.; Atak, M.; Wilson J. A genetic algorithm for flow shop scheduling problem. *J Oper. Res. Soc.*, 2004, 55, 830–835.

[33] Falkenauer, E; Bouffouix, S. A genetic algortihm for the job-shop. In: *Proceedings of the IEEE International Conference on Robotics and Automation*, 1991, 824 – 829.

[34] Fang, H.L.; Ross, P.; Corne, D. A promising hybrid GA/heuristic approach for open shop scheduling problems. *Proceedings of the 11th European Conference on Artificial Intelligence*, John Wiley and Sons, 1994, 590 – 594.

[35] Fisher, H.; Thompson, G.L. Probabilistic learning combinations of local job-shop scheduling rules. In: Muth, J.F.; Thompson, G.L. (eds.) *Industrial Scheduling, Prentice Hall, Englewood Cliffs*, 1963.

[36] Franca, P.M.; Tin, G.; Buriol, L.S. Genetic algorithms for the no-wait flowshop sequencing problem with time restrictions. *Int J Prod Res*, 2006, 44, 939 - 957.

[37] Gao, J.; Sun, L.; Gen, M. A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Comp Oper Res*, 2008, 35, 2892 – 2907.

[38] Ghedjati, F. Genetic algorithms for the job-shop scheduling problem with unrelated parallel constraints: heuristic mixing method machines and precedence. *Comp Ind Engn*, 1999, 37, 39 – 42.

[39] Giffler, B.; Thompson, G.L. Algorithms for solving production scheduling problems. *Oper Res*, 1960, 8, 487 – 503.

[40] Goldberg, D.E. Genetic algorithms in search, optimization and machine learning. Addison-Wesley, Reading, 1989.

[41] Goncalves, J.F.; Magalhaes Mendes, J.J.; Resende, M.G.C. A hybrid genetic algorithm for the job shop scheduling problem. *AT&T Labs Research Technical Report TD-5EAL6J*, 2002.

[42] Graham, R.L.; Lawler, E.L.; Lenstra, J.K.; Rinnooy Kan, A.H.G. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 1979, 5, 287 – 326.

[43] Gueret, C.; Prins, C. A new lower bound for the open-shop problem. *Ann Oper Res*, 1999, 92 (1), 165 – 183.

[44] Hall, N.G.; Posner, M.E. Generating experimental data for computational testing with machine scheduling problems. *Oper Res*, 2001, 49, 854 - 865.

[45] Harmanani, H.M.; Drouby, F.; Ghosn, S.B. A parallel genetic algorithm for the open-shop scheduling problem using deterministic and random moves. *Proceedings of SpringSim'09*, SCS/ACM, 2009.

[46] Ho, J.C.; Chang, Y.-L. A new heuristic for the n-job, m-machine flow-shop problem.*Eur J Oper Res*, 1991, 52, 194 – 202.

[47] Holland, J.A. Adaptation in natural and artificial systems. *Ann Arbor: University of Michigan*, 1975.

[48] Jain, A.S.; Meeran, S. Deterministic job-shop scheduling: Past, present and future. *Eur J Oper Res*, 1999, 113, 390 – 434.

[49] Janiak, A; Portmann, M.-C. Genetic algorithm for the permutation flow-shop scheduling problem with linear models of operations. *Ann Oper Res*, 1998, 83, 95 – 114.

[50] Jarboi, B.; Eddaly, M.; Siarry, P. A hybrid genetic algorithm for solving no-wait flowshop scheduling problems. *Int J Adv Manuf Technol*, 2011, 54, 1129 - 1143.

[51] Jones, G. Genetic and Evolutionary Algorithms. *University of Sheffield*, 2004.

[52] Jolai, F.; Sheikh, S.; Rabbani, M.; Karimi, B. A genetic algorithm for solvng no-wait flexible flow lines with due windows and job rejection. *Int J Adv Manuf Technol*, 2009, 42, 523 – 532.

[53] Jungwattanakit, J.; Reodecha, M.; Chaovalitwongse; Werner, F. Algorithms for flexible flow shop problems with unrelated parallel machines. *Int J Adv Manuf Technol*, 2008, 37, 354 – 370.

[54] Jungwattanakit, J.; Reodecha, M.; Chaovalitwongse; Werner, F. A comparison of scheduling algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteria. *Comp Oper Res*, 2009, 36, 358–378.

[55] Kacem, I.; Hammadi, S.; Borne, P. Approach by localization and multi-objective evolutionary optimization for flexible job-shop scheduling problems. *IEEE Transactions on Systems, Man, and Cybernetics*, Part C, 2002, 32 (1), 408 - 419.

[56] Kahramann, C.; Engin, O.; Kaya, I.; Yilmaz, M.K. An application of effective genetic algorithms for solving hybrid flow shop scheduling problems. *Intern J of Computational Intelligence Systems*, 2008, 1, 134 – 147.

[57] Khuri, S.; Miryala, S.R. Genetic algorithm for solving open shop scheduling problems. In: *Proceedings 9th Portuguese Conference on Artificial Intelligence*, 1999, 357 – 368.

[58] Kobayashi, S.; Ono, I.; Yamamura, M. An efficient genetic algorithm for job shop scheduling problems. *Proceedings of ICGA '95*, 1995, 506 – 511.

[59] Kokosinski, Z.; Studzienny, L. Hybrid genetic algorithms for the open-shop scheduling problem. *IJCSNS International Journal of Computer Science and Network Security*, 2007, 7, 136 – 145.

[60] Lei, D. A genetic algorithm for flexible job shop scheduling with fuzzy processing time. *Int J Prod Res*, 2010, 48, 2995 – 3013.

[61] Lawrence, S. Supplement to resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques. *Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh*, 1984.

[62] Li, Y.; Chen, Y. A genetic algorithm for job-shop scheduling. *Journal of Software*, 2010, 5, 269 – 274.

[63] Lei, D. A genetic algorithm for flexible job shop scheduling with fuzzy processing time. *Intern J Prod Res*. 2009, 48 (10), 2995 – 3013.

[64] Liaw, C.-F. A hybrid genetic algorithm for the open-shop scheduling problem. *Eur J Oper Res*, 2000, 124, 28 – 42.

[65] Lin, S.-C.; Goodman, E.; Punch, W.F. A Genetic Algorithm Approach to Dynamic job shop scheduling problems.

[66] Lin, S.-C.; Goodman, E.; Punch, W.F. Investigating parallel genetic algorithms on job shop scheduling problems, 1997.

[67] Liu, T.-K.; Tsai, J.-T.; Chou, J.-H. Improved genetic algorithm for the job-shop scheduling problem. *Int J Adv Manuf Technol*, 2006, 27, 1021 – 1029.

[68] Matsui, S.; Yamada, S. An empirical performance evaluation of a parameter-free genetic algorithm for job-shop scheduling problem. *IEEE Congress on Evolutionary Computation (CEC 2007)*, 2007, 3796 – 3803.

[69] Moon, I.; Lee, J. Genetic algorithm application to the job shop scheduling problem with alternative routings. *Pusan National University*, 2000.

[70] Moonen, M.; Janssens, G.K. A Giffler-Thompson focused genetic algorithm for the static job-shop scheduling problem. *The Journal of Information and Computational Science*, 2007, 4, 609 – 624.

[71] Murata, T; Ishibuchi, H.; Tanaka, H. Multi-objective genetic algorithm and its applications to flowshop scheduling. *Computers Ind Engng*, 1996, 30, 957 – 968.

[72] Murata, T; Ishibuchi, H.; Tanaka, H. Genetic algorithms for flowshop scheduling problems. *Computers Ind Engng*, 1996, 30, 1061 – 1071.

[73] Naderi, B.; Fatemi Ghomi, S.M.T.; Aminnayeri, M.; Zandieh A study on open shop scheduling to minimize total tardiness. *Int J Prod Res*, 2011, 49 (15), 4657 – 4678.

[74] Nawaz, M.; Enscore , E.E.; Ham, I. A heuristic algorithm for the *m*-machine, *n*-job flow shop sequencing problem. *Omega*, 1983, 11, 91 – 95.

[75] Nepalli, V.R.; Chen, C.L.; Gupta, J.N.D. Genetic algorithm for the two-stage bicriteria flowshop problem. *Eur J Oper Res*, 1996, 95, 356 – 373.

[76] Nissen, V. An overview of evolutionary algorithms in management applications. In: Biethahn, J.; Nissen, V. (eds.) *Evolutionary algorithms in management applications*, *Springer*, 1995, 44 – 97.

[77] Nissen, V.; Biethahn, J. An introduction to evolutionary algorithms. In: Biethahn, J.; Nissen, V. (eds.) *Evolutionary algorithms in management applications*, *Springer*, 1995, 3 – 43.

[78] Nowicki, E.; Smutnicki, C. A fast taboo search algorithm for the job-shop problem. *Management Sci*, 1996, 42, 797 – 813.

[79] Oguz, C.; Ercan, M.F. A genetic algorithm for hybrid flow-shop scheduling with multiprocessor tasks. *J Sched*, 2005, 8, 323 – 351.

[80] Oliver, I.M.; Smith, D.J.; Holland, J. A study of permutation crossover operators on the traveling salesman problem. In: *Proc 2nd Int Conf on Genetic Algorithms (ICGA'87)*, Cambridge, 1987, 224 – 20. *Eur J Oper Res*, 2006, 171, 674 – 692.

[81] Onwubolu, G.C.; Mutingi, M. Genetic algorithm for minimizing tardiness in flowshop scheduling. *Prod Planning and Control*, 1999, 10, 462 – 471.

[82] Palacios, J.J.; Puente, J.; Vela C.R.; Gonzalez-Rodriguez, I. A genetic algorithm for the open shop problem with uncertain durations. *Proceedings of IWINAC 2009, Part I, Lect Notes Comp Sci*, 2009, 5601, 255 – 264.

[83] Park, B.J.; Choi, H.R.; Kim, H.S. A hybrid genetic algorithm for the job shop scheduling problems. *Comput Ind Engn*, 2003, 45, 597 - 613.

[84] Peng L.H.; Salim, S. A modified Giffler and Thompson genetic algorithm on the job shop scheduling problem *Mathematika*, 2006, 22, 91 – 107.

[85] Ponnambalam, S.G.; Aravindan, P.; Chandrasekaran, S.: Constructive and improvement flow shop scheduling heuristics: an extensive evaluation. *Prod Planning Control*, 2001, 12, 45 – 60.

[86] Pannambalam, S.G.; Jagannathan, H.; Kataria, M.; Gadicherla, A. A TSP-GA multiobjective algorithm for flow-shop scheduling. *Int J Adv Manuf Technol*, 2004, 23, 909 – 915.

[87] Pasupathy, T.; Rajendran, C.; Suresh, R.K. A multi-objective genetic algorithm for scheduling in flow shops to minimize the makespan and total flow time of jobs. *Int J Adv Manuf Technol*, 2006, 27, 804 – 815.

[88] Potts, C.N.; Strusevich, V.A. Fifty years of scheduling: A survey of milestones. *J Oper Res Soc*, 2009, 60, S41 – S68.

[89] Prins, C. Competitive genetic algorithm for the open-shop scheduling problem. *Math Meth Oper Res*, 2000, 52, 389 – 411.

[90] Rajendran, C. Heuristic algorithm for scheduling in a flowshop to minimize total flow-time. *Intern J Prod Econ*, 1993, 29, 65 – 73.

[91] Rajkumar, R.; Shahabudeen, P. An improved genetic algorithm for the flowshop scheduling problem. *Intern J Prod Res*, 2009, 47 (1), 233 – 249.

[92] Rashidi, E.; Jahandar, M.; Zandieh, M. An improved hybrid multi-objective parallel genetic algorithm for hybrid flow shop scheduling with unrelated parallel machines. *Int J Adv Manuf Technol*, 2010, 49, 1129 – 1139.

[93] Rechenberg, I. Evolutionsstrategie. Optimierung technischer Systeme nach Prinzipien der biologischen Evolution, *Frommann-Holzboog*, Stuttgart, 1973.

[94] Reeves, C.R. A genetic algorithm for flowshop sequencing. *Comp Oper Res*, 1995, 22, 5 – 13.

[95] Reeves, C.; Yamada, T. Genetic algorithm, path relinking and the flowshop sequencing problem. *Evolutionary computation*, 1998, 12, 335 - 344.

[96] Reza Hejazi, S.; Saghafian, S. Flowshop scheduling problems with makespan criterion: a review. *Intern J Prod Res*, 2005, 43, 2895 - 2929.

[97] Ruiz, R.; Maroto A comprehensive review and evaluation of permutation flow shop heuristics. *Eur J Oper Res* 2005, 165, 479 – 494.

[98] Ruiz, R.; Maroto, C.. A genetic algorithm for hybrid flowshops with sequence-dependent setup times and machine eligibility

[99] Ruiz, R.; Maroto, C.; Alcaraz; J. Two new robust genetic algorithms for the flowshop problem. *Omega*, 2006, 34, 461 – 476.

[100] Ruiz, R.; Vazquez-Rodriguez The hybrid flow shop scheduling problem. *Eur J Oper Res*, 2010, 205, 1 – 18.

[101] Sakawa, M.; Mori, T. An efficient genetic algorithm for job-shop scheduling problems with fuzzy processing time and fuzzy duedate. *Comput Ind Engng*, 1999, 36, 325 – 341.

[102] Schwefel, H.P. Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie. *Birkhäuser, Basel*, 1977.

[103] Senthilkumar, P.; Sahhabudeem, P. GA based heuristic for the open shop scheduling problem. *Intern J Adv Man Technol*, 2006, 30 (3-4), 297 – 301.

[104] Sharma, B.; Yao, X Characterizing genetic algorithm approaches to job shop scheduling. *UK Workshop on Computational Intelligence*, Loughborough University, 2004.

[105] Sridhar, J.; Rajendran, C. Scheduling in flowshop and cellular manufacturing systems with multiobjectives - a genetic algorithmic approach. *Prod Plann Control*, 2007, 7, 734 – 382.

[106] Sivrikaya Serifoglu, F.; Ulusoy, G. Multiprocessor task scheduling in multistage hybrid flow shops: A genetic algorithm approach. *J Oper Res Soc*, 2004, 55, 504–512.

[107] Storer, R.H.; Wu, S.D.; Vaccari, R. New search spaces for sequencing problems with applications to job-shop scheduling. *Management Sci*, 1992, 38, 453 – 467.

[108] Sun, Y.; Zhang, C.; Gao,L.; Wang X. Multi-objective optimization algorithms for flow shop scheduling problem: a review and prospects. *Int J Adv Manuf Technol*, 2011, 55, 723 – 739.

[109] Syswerda, G. Schedule optimization using genetic algorithms. In: L. Davis (ed.), *Handbook of Genetic Algorithms*, 1990, 332 – 349.

[110] Taillard, E. Benchmarks for basic scheduling problems. *Eur J Oper Res*, 1993, 64, 278 – 285.

[111] Tang, L.; Liu, J. A modified genetic algorithm for the flow shop sequencing problem to minimize mean flow time. *J Int Manuf*, 2002, 13, 61 – 67.

[112] Tseng, L.-Y.; Lin, Y.-T. A hybrid genetic local search algorithm for the permutation flowshop scheduling problem. *Eur J Oper Res*, 2009, 198, 84 – 92.

[113] Uysal, O.; Bulkan, S. Comparison of genetic algorithm and particle swarm optimization for bicriteria permutation flowshop scheduling problem. *Int J Comp Intell Res*, 2008, 4, 159 – 175.

[114] Vallada, E.; Ruiz, R. Genetic algorithms with path relinking for the minimum tardiness permutation flow shop problem. *Omega*, 2010, 38, 57 - 67.

[115] Vallada, E.; Ruiz, R.; Minella, G. Minimising total tardiness in the *m*-machine flowshop problem: A review and evaluation of heuristics and metaheuristics. *Comp Oper Res*, 2008, 35, 1350 – 1373.

[116] Vazquez, M.; Whitley, L.D. A comparison of genetic algorithms for the static job shop scheduling problem, *Lecture Notes Comp Sci*, 2000, 1917, 303 – 312.

[117] Vazquez, M.; Whitley, L.D. A comparison of genetic algorithms for the dynamic job shop scheduling problem. *Genetic and Evolutionary Computation Conference*, 2000, 1011 – 1018.

[118] Vela, C.R.; Varela, R.; Gonzalez, M.A. Local search and genetic algorithm for the job shop scheduling problem with sequence dependent setup times. *J Heuristics*, 2010, 16, 139 – 165.

[119] Wang, W.; Brunn, P. An effective genetic algorithm for job shop scheduling. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, 2000, 214 (4), 293 – 300.

[120] Wang, L.; Zheng, D.-Z. A modified genetic algorithm for job shop scheduling. *Int J Adv Manuf Technol*, 2006, 27, 1021 – 1029.

[121] Werner, F. On the solution of special sequencing problems. *Ph.D. Thesis*, TU Magdeburg, 1984 (in German).

[122] Werner, F. An adaptive stochastic search procedure for special scheduling problems. *Economicko-Matematicky Obzor*, 1988, 24, 50 – 67 (in German).

[123] Werner, F. On the structure and approximate solution of special combinatorial optimization problems. *Habilitation Thesis*, TU Magdeburg, 1989 (in German).

[124] Werner, F. Some relations between neighborhood graphs for a permutation problem, 1991, 22, 297 - 306.

[125] Werner, F. On the heuristic solution of the permutation flow shop problem by path algorithms. *Comp Oper Res*, 1993, 20, 707 – 722.

[126] Xie, H. A genetic algorithm approach to job shop scheduling problems with a batch allocation issue. *Proceedings of the 17th International Workshop on Artificial Intelligence*, Seattle, 2001, 126 – 131.

[127] Xing, Y.J.; Wang, Z.Q.; Sun, J.; Meng, J.J. A multi-objective fuzzy genetic algorithm for job-shop scheduling problems. *Journal of Achievements in Materials and Manufacturing Engineering*, 2006, 17, 297 – 300.

[128] Yamada, T.; Nakano, R. A genetic algorithm applicable to Large-Scale job-shop problems. *Parallel Problem Solving from Nature* 1992, 2, 281 – 290.

[129] Yamada, T.; Nakano, R. A genetic algorithm with multi-step crossover for job-shop scheduling problems. *GALESIA'95*, 1995, 146 – 151.

[130] Yamada, T.; Nakano, R. Scheduling by genetic local search with multi-step crossover. Fourth International Conference on Parallel Problem Solving from Nature (PPSN IV), 1996, 960 – 969.

[131] Yamada, T.; Nakano, R. Genetic algorithms for job-shop scheduling problems *Proceedings of Modern Heuristic for Decision Support*, London, 1997, 67 – 81.

[132] Yaurima, V.; Burtseva, L.; Tchernykh, A. Hybrid flowshop with unrelated machines, sequence-dependent setup time, availability constraints and limited buffers. *Comput Ind Engng*, 2008,.

[133] Zandieh, M.; Mozaffari, E.; Gholami, M. A robust genetic algorithm for scheduling realistic hybrid flexible flow line problems. *J Intell Manuf*, 2010, 21, 731 – 743.

[134] Zhan Y.; Qiu, C.H.; Xue, K. A hybrid genetic algorithm for hybrid flow shop scheduling with load balancing. *Key Engineering Materials*, 2009, 392 – 394, pp. 250 – 255.

[135] Zhang, H.; Gen, M. Multistage-based genetic algorithm for flexible job-shop scheduling problem. *Journal Complexity International*. 2005, 11, 223 – 232.

[136] Zhang, R.; Wu, C. An immune genetic algorithm based on bottleneck jobs for the job shop scheduling problem. *Proceedings of EvoCOP 2008, Lecture Notes Comp Sci*, 2008, 4972, 147 – 157.