A Giffler-Thompson genetic algorithm for the static job-shop scheduling problem

A Giffler-Thompson Focused Genetic Algorithm for the Static Job-Shop Scheduling Problem

Mark Moonen [a], Gerrit K. Janssens [b,*],

[a] *Centre for Industrial Management, Katholieke Universiteit Leuven, Celestijnenlaan 300A, B-3001 Heverlee-Leuven, Belgium*

[b] *Hasselt University - Campus Diepenbeek, Agoralaan, B-3590 Diepenbeek, Belgium*

Abstract

As the job shop scheduling problem is a difficult problem in combinatorial optimisation, a lot of research has been devoted to obtaining lower bounds for its objective function, in constructing branch-and-bound algorithms and in developing efficient heuristics and meta-heuristics. Several genetic algorithms have been developed for the job-shop scheduling problem. The design of the genetic algorithms for the problem under study varies in the way of encoding a solution and in the use of its various operators. In this design of the genetic algorithm a crossover operator is developed based on principles set by Giffler and Thompson to generate active schedules. Two alternative designs are tested on a number of benchmark problems. While performing well in general, the genetic algorithm performs rather poor on benchmark problems, which are known in the literature as extremely hard to solve. The genetic algorithm, as presented here, makes use of a parameter, which is believed to have an important influence on performance of the algorithm and of which an intelligent setting of its value might lead to promising results for the most difficult job-shop scheduling problems.

Keywords: Job-shop scheduling; Static; Genetic algorithm

1 Introduction

The job-shop scheduling problem (JSP) is known to be one of the hardest combinatorial optimisation problems [17]. Mathematical programming techniques are of little use as they could be used only to solve play-toy problems. Mathematical programming formulations of the problem have been proposed as an integer programming problem [3, 4], a mixed-integer programming problem [5, 6], and a dynamic programming problem [33].

Because of this intractability, heuristics may serve as a valid alternative. Conventional heuristics make use of a priority rule or a dispatching rule, i.e. a rule is used to select an operation on a

*Corresponding author.

Email addresses: Mark.Moonen@cib.kuleuven.be (Mark Moonen), Gerrit.Janssens@uhasselt.be (Gerrit K. Janssens).

machine from a list of unscheduled operations. Popular priority rules include: Shortest Processing Time first, Earliest Due Date first, Least Slack first, Critical Ratio, etc. (see e.g. [25], chapter 15). The quality of the solution obtained by these simple heuristics however is questionable. In [20] it is proposed that an effective priority rule has to be dynamic such that it takes into account additional information not available to the static rule and reflects the status of jobs, from time to time as the schedule progresses. While being more complex in computation compared to a simple priority rule, the 'shifting-bottleneck' algorithm [2] is considered to be the most powerful among all heuristics. Priority rules fit into a class of heuristic algorithms for the JSP, which generate non-delay schedules [34].

A good schedule may be used further in procedures, which try to improve this solution. Probabilistic local search methods such as Simulated Annealing, Tabu Search and Genetic Algorithms have been used to solve the job-shop scheduling problem. In this research we develop a new genetic algorithm for the JSP. Other implementations of genetic algorithms for the JSP will be discussed in later sections.

Our work deals with the classical job-shop model. There are N different **jobs** $\{J_1, J_2, \ldots, J_N\}$ to be scheduled on M different **machines** $\{M_1, M_2, \ldots, M_M\}$. The execution of a job i on a machine j is called an **operation** o_{ij}. The operation order of a job on the machines is pre-specified. Each operation o_{ij} is characterised by the required machine and the fixed processing time p_{ij}.

Other constraints on jobs and machines are:

- a job does not visit the same machine twice,
- no precedence requirements exist amongst operations of different jobs,
- an operation, once started, cannot be interrupted,
- each machine can process only one job at a time,
- neither release times nor due dates are specified.

In general the job-shop scheduling problem is to determine the operation sequences on the machines in such a way that they are compatible with the sequence requirements and in order to optimise a performance criterion [16]. In most cases the makespan, i.e. the time required to complete all jobs, is chosen as the performance criterion.

Davis [11] is generally recognised as the first publication mentioning the use of genetic algorithms to solve the job-shop scheduling problem. In order to design a genetic algorithm for the job-shop scheduling problem decisions have to be taken on: the representation of a solution, the operators, the creation of an initial population, the parent selection mechanism, the reproduction mechanism, and the use of elitism.

A first step in the design of a genetic algorithm (GA) consists of defining an appropriate *representation* of the solutions. Gen and Cheng [19] list nine representations for the job-shop scheduling problem, obtained from the literature. A distinction is made between direct and indirect representation approaches. *Direct* approaches encode a schedule into a chromosome and GAs are used to

evolve the chromosomes in order to obtain better schedules. *Indirect* approaches encode a sequence of rules or procedures into a chromosome. GAs are used to evolve the chromosomes in order to obtain better sequences. A solution is obtained by using the sequence of instructions encoded in the chromosome. For the reader's reference we list nine types of representations including a reference where further details may be found. They are: the operation-based representation [14], job-based representation [27], preference-list-based representation [12], job-pair-relation-based representation [31], disjunctive-graph-based-representation [35], completion-time-based representation [37], machine-based-representation [13], random key representation [8], and priority-rule-based representation.

In the design of a genetic algorithm, the design of the *crossover operator* is of major importance. Within a specific algorithm the operator has to be adapted according to the representation of the chromosome and other algorithm-specific elements. N-point crossover and uniform crossover are classical types of crossover operators and are of relevance in this problem too. Their specific details will be discussed further on. The inversion operator serves as a re-ordering operator. In practice a sequence between two points within the chromosome is reversed. This operator has shown to be less successful in our type of problems. The mutation operator, which is used to induce some population diversity, may also be of use in our type of problem.

Most GA research focuses on static job-shop scheduling problems, where all jobs have ready times equal to zero. In a dynamic job shop jobs arrive at known (resp. unknown) epochs in the future. The problems are called deterministic (resp. stochastic) dynamic job-shop scheduling problems. Genetic algorithms have been used in several studies of the dynamic job-shop scheduling problems. The design of the genetic algorithms for this type of problems is quite different amongst the various authors, which makes a comparison difficult. The authors, who study the stochastic version of the problem, make use of a method, which is proposed in [32]. They approach the stochastic problem as a sequence of deterministic problems. When, due to a stochastic event, a new job arrives to the system, a new deterministic problem is formulated.

Hart and Ross [24] design a heuristic-based genetic algorithm in which a chromosome encodes both the choice of an algorithm to generate the set of schedulable operations as the choice of a heuristic to choose the operation to be executed first. The algorithm is called the 'heuristically-guided GA' (HGA). Fang [15] develops an algorithm both for deterministic and stochastic job-shop scheduling problems. He makes use of an indirect representation, which at crossover and mutation always produces legal offspring, similar to [22] for the Travelling Salesman Problem. Vazquez and Whitley [36] construct an order-based Giffler & Thomson genetic algorithm (OBGT-GA). It is a hybrid algorithm, which makes use of a direct representation and in which order-based techniques are used to produce active and non-delay schedules. The order-based techniques have as main goal the repair of illegal offspring. Bierwirth and Mattfeld [10] also use an indirect representation for their genetic algorithm, called Precedence Preservative Crossover, in order to solve both deterministic and stochastic problems. Their approach allows re-scheduling without a cold start, each time a new job arrives. A similar genetic algorithm with the same re-scheduling characteristics, has been proposed in [30].

2 A Giffler-Thompson Focused Crossover Operator

Three types of feasible job shop schedules are distinguished:

Semi-active schedules No operation can start earlier without altering the machine sequence.

Active schedules No operation can start earlier without delaying another operation or without violating the precedence constraints.

Non-delay schedules No machine is kept idle when there is an operation that can start on this machine.

All other schedules are infeasible as they violate the precedence constraints or other assumptions of the JSP. Figure 1 shows the relation between the different types of schedules. From this figure it is clear that all non-delay schedules are active schedules, and that all active schedules are semi-active schedules. French [16] states however that the optimal schedule (for minimising the makespan) is always an active schedule. This observation allows a reduction of the search space considerably, which is of high value given the NP-hardness of the JSP [18]. Giffler and

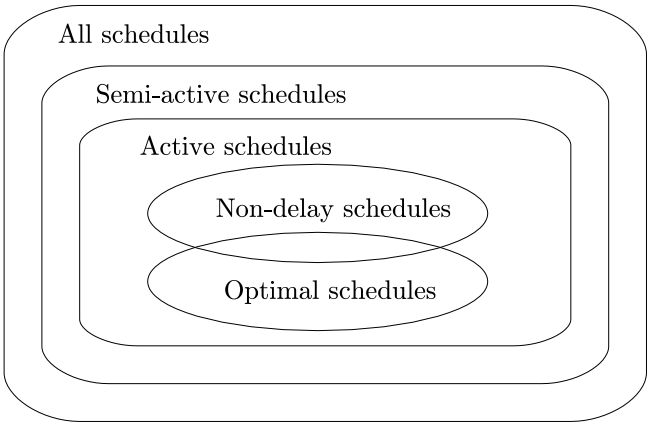

Fig. 1: Schedule Types

Thompson [21] developed a recursive procedure to generate active schedules in a systematic way. The presentation of the Giffler-Thompson (GT) algorithm makes use of the following symbols:

- o_k is the operation k ($1 \leq k \leq nm$).
- p_k is the operation time of operation o_k.
- P_t is the partial schedule of the $(t-1)$ scheduled operations.
- S_t is the set of operations that can be inserted in the schedule at iteration t. Operations, which should be performed before the operations in S_t, are already included in the schedule P_t.
- ρ_k is the earliest time at which operation o_k from S_t can start.
- b_k is the earliest time at which operation o_k from S_t can be terminated, i.e. $b_k = \rho_k + p_k$.

With these symbols defined, the Giffler-Thompson algorithm proceeds as follows:

Step 1 Let $t = 1$ and $P_1 = \{\}$. S_1 is the set of all operations without predecessors.

Step 2 Find $b^* = \min_{o_k \in S_k}\{b_k\}$ and the machine M^* at which o_k with b^* is executed. When there are multiple M^*, choose arbitrarily.

Step 3 Choose an operation o_j from S_t such that:

- operation o_j needs machine M^*, and
- $\rho_j < b^*$

Step 4 Continue to next iteration by:

- adding o_j to P_t, resulting in P_{t+1}
- removing o_j from S_t and creating S_{t+1} by adding the successor of o_j to S_{t+1} (unless o_j is the last operation)
- $t = t + 1$

Step 5 If $S_t \neq \{\}$ go to step 2. Else Stop.

At each iteration the GT procedure generates a conflict set S_t from which an operation is chosen and scheduled. The recursive GT procedure is repeated until all operations are scheduled. This algorithm generates feasible active schedules. The precedence order is preserved. Steps 2 and 3 of the GT algorithm assure that no operation can start earlier, while respecting the precedence constraints, without delaying another operation, resulting in an active schedule. By considering all operations in S_t one can create all active schedules and the relevant search space.

By modifying steps 2 and 3 of the GT algorithm (see below) also non-delay schedules may be generated, in such a way that a machine is never idle when there is an operation ready to be executed on that machine.

Step 2 Find $\rho^* = \min_{o_k \in S_k}\{\rho_k\}$ and the machine M^* at which o_k with ρ^* is executed. If multiple M^* exist, choose arbitrarily.

Step 3 Choose an operation o_j from S_t such that:

- this operation o_j needs machine M^*, and
- $\rho_j = \rho^*$

Using the non-delay (ND) variant of the GT algorithm reduces the search space even further [7]. The reduced search space however comes at a cost, as the danger exists that the optimal solution is excluded from consideration. Della Croce et al. [12] found that in some cases a non-delay scheduler improves the solution quality while in other cases an active scheduler might be more appropriate.

Within the context of genetic algorithms, the GT algorithm and its non-delay variant are used to convert chromosomes to feasible schedules [36]. Crossover or mutation may result in chromosomes

which do not respect the precedence constraints. The GT algorithm forces false chromosomes into a correct schedule. Other genetic algorithms apply the GT or the ND algorithm to translate a chromosome with indirect representation to a feasible active or non-delay schedule (see e.g. [24]).

Hart and Ross [24], among others, point out that the size of the conflict set (CS) varies throughout the consecutive loops of the GT procedure. The operation that eventually will be scheduled during a loop of the GT algorithm, is selected from the conflict set S_t. If the CS contains only one operation, this operation will be scheduled. But more operations in the CS leads to a choice. Crossover at a gene with only one operation in the CS will result in the same operation being scheduled at this position. Therefore Hart and Ross [24] advise to avoid these genes as a crossover point and focus on the genes with a large CS.

Fang [15] however remarks that the more decisive genes are located in the beginning of a chromosome. The more the end of a chromosome is approached, the higher the chance that the genes are redundant. Their meaning is fixed by the scheduling of the preceding operations.

We develop a new crossover operator, which we indicate as a Giffler-Thompson Focused (GTF) crossover, that takes into account both findings. Our presentation of the new operator assumes chromosomes with a permutation representation of all operations, i.e. every operation has its own number and the order of these operations in the chromosome determines the order in which they are scheduled. The representation belongs to the operation-based representations.

The GTF operator combines an order-based crossover operator with a one-point crossover operator. An order-based crossover operator selects a number of genes from one parent and copies them to the offspring. The missing genes are taken from the second parent in the order that they appear in this parent. A one-point crossover operator generates randomly a crossover point. It leaves the part of the chromosome before this point unchanged and exchanges the part behind the crossover point between the two parents. Some researchers have modified the one-point crossover operator by "directing" the choice of the crossover-point.

Our approach also "directs" the crossover operator. We choose the first gene with the largest CS as the crossover point. We incorporate the crossover operator in a strategy similar to neighbourhood-search-based mutation ([19], p. 219). This means that the crossover operator does not generate a fixed number of offspring, but that the number of (intermediate) offspring equals the size of the conflict set. The GTF crossover operator may be presented as:

Step 1 Choose a first parent ($P1$).

Step 2 Determine for $P1$ the gene with the largest CS. Let p^* be the locus of this gene, CS^* the conflict set and $|CS|$ be the number of operations in CS^*.

Step 3 Generate $|CS|$ intermediate offspring. The first part of these intermediate offspring are the genes before p^* of $P1$.

Step 4 For the $|CS|$ offspring the operation at p^* are the operations from CS^*.

Step 5 The following genes are taken from a parent $P2$ in the order as they appear in $P2$, with $P2$ different for each intermediate offspring and $P2 <> P1$.

The working of the GTF operator is illustrated by the following example:

Step 1:

P1 = 1 3 5 6 4 2 9 8 7

Step 2:

Assume that CS* occurs at gene 4 (p*) with |CS| = 3 and CS*={6, 4, 9}

Step 3:

Offspring 1: 1 3 5

Offspring 2: 1 3 5

Offspring 3: 1 3 5

Step 4:

Offspring 1: 1 3 5 6

Offspring 2: 1 3 5 4

Offspring 3: 1 3 5 9

Step 5:

P2 = 5 4 3 1 9 7 8 2 6

Offspring 1: 1 3 5 6 4 9 7 8 2

Offspring 2: 1 3 5 4 9 7 8 2 6

Offspring 3: 1 3 5 9 4 7 8 2 6

An ideal GA should be able to preserve a high degree of diversity within the population in order to perform an efficient search [23]. In order to avoid premature convergence of the population we cannot insert all intermediate offspring into the population. By doing this we would destroy the genetic diversity. The GA would be directed into a specific direction of the search space eliminating other possibly interesting areas of the search space. Therefore step 6 is added to our GTF operator procedure in which only the best intermediate offspring is inserted into the population.

Step 6:

Return the best chromosome from the |CS| intermediate offspring.

Essentially the GTF crossover operator exploits problem specific information generated during the application of the GT algorithm, knowing that hybridisation of a GA can improve the performance.

3 Implementation and Experimental Design

The performance of the GTF operator, which has been described in section 2, is evaluated by an implementation in a GA. The implementation of the GA is largely inspired on the work by Bierwirth and Mattfeld [10] and by Bierwirth [9].

Two experimental settings are analysed. In the first setting the Precedence Preservative Crossover (PPX) operator is used (see [9, 10]). This setting serves as a benchmark to evaluate a second setting in which we use the GTF crossover operator in combination with the PPX operator.

The implementation of the first setting is described in detail. Later the modifications, which are required to analyse the GTF operator, are explained.

3.1 First experimental setting

An operation-based representation is used for a chromosome in which each operation has a unique number. The ordering of the operations is used to build an active schedule using the GT algorithm.

The GA starts from a randomly generated population. Because the objective is to minimise the makespan, parent selection is based on inverse proportional fitness. Two parents reproduce to generate one offspring. A generation-to-generation reproduction strategy is followed [26]. Parents in the old generation can reproduce and generate offspring until the new population is completely filled up. Once filled up, the new population replaces the old one. To avoid that the solution quality decreases over the different generations, elitism is applied. This means that if the best solution of the new generation is worse than the best solution of the old generation, the latter solution replaces the worst solution of the new generation.

The PPX-operator is used as a crossover operator. The mutation operator picks an operation from the chromosome and inserts it at a randomly chosen position in the chromosome. As the genetic operators do not take into account precedence relationships, offspring chromosomes do not necessarily respect these precedence relationships. To solve this problem Bierwirth and Mattfeld apply a technique called 'forcing' [10]. If, in an iteration of the GT algorithm, an operation is found that cannot be scheduled due to the precedence constraints, the GT algorithm takes the next operation until it finds an operation that does respect the precedence constraints. Every iteration this technique is repeated, in such a way that the resulting chromosome satisfies the precedence constraints. Finally, the resulting sequence of operations is translated into a feasible and legal chromosome.

The values of crossover and mutation probabilities are taken from [10]. The crossover probability P_c is set to 0.8 and the mutation probability P_m equals 0.2. In most designs P_m is set to a low value. A value of 0.2 might seem rather high but Bierwirth and Mattfeld argue that the effective percentage is lower because the application of the mutation operator often results in illegal offspring [10]. The population consists of 100 chromosomes and the algorithm runs for 150 generations (see also [9]). In figure 2 we give the general outline of the GA.

3.2 Second experimental setting

The second setting is similar to the first setting, but it differs in the choice of the crossover operator. If the GA decides to execute a crossover operation, it chooses with a probability P_{gtf} for the GTF-operator and with a chance $(1 - P_{gtf})$ for the PPX-operator. We do this because the GTF operator generates and evaluates each time |CS*| offspring while the PPX-operator only produces one. Setting P_{gtf} equal to one results in a low number of generations, not allowing the

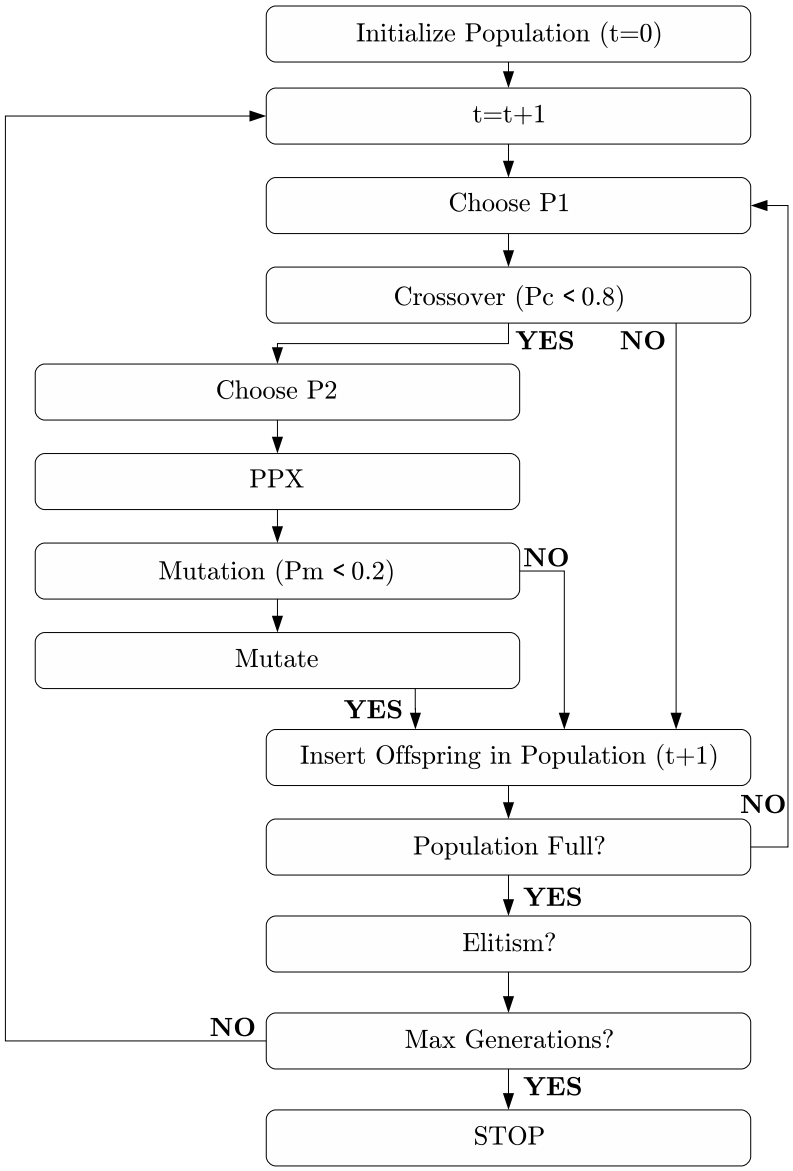

Fig. 2: GA Outline

GA to explore the search space adequately. In our experiments we set P_{gtf} arbitrarily equal to 0.5. The GA with the GTF operator generates and evaluates more offspring in one generation. To come to an honest comparison we allow this GA to evaluate only as much chromosomes as the GA without GTF, i.e. 15.000 chromosomes.

3.3 Benchmark problems

Both genetic algorithms are tested on a number of benchmark problems known from the literature. All benchmark problems are n by m static benchmark problems, with n the number of jobs and m the number of machines. They include:

- the 10x10 orb01-orb10 problems from [1],

- the 20x10 la26-la30 from [29],
- the 15x15 la36-la40 from [29],
- the 50x10 swv11-swv20 from [34].

Both genetic algorithms are implemented in C++ and run on a AMD1800+ PC with 256 MB memory. Both genetic algorithms are run for the benchmark problems for 10 independent runs.

4 Results and Discussion

In table 1 the results of the experiment are presented. It shows the average makespan over 10 runs, and the procentual difference between the averages for both settings. To assess whether the averages from both set-ups are significantly different, a t-test on a 5% significance level on the averages is performed. In table 1 the significant differences are marked with an asterisk.

For the second setting the table also gives the average number of generations and the run-time of the GA (in seconds). Additionally the optimal or best-known solutions for the benchmark problems, taken from [28], are indicated. The last column gives the procentual difference between setting 2 and the optimal or best-known solutions.

On a total of 30 benchmark problems the second experimental setting outperforms 26 times the first setting. For these 26 benchmark problems this difference is significantly on a 5% significance level. In three cases where the GTF-approach does not perform significantly better, (benchmarks swv16 to swv18), both algorithms consistently generate the optimal solution. Only for a single benchmark problem (swv20) the PPX-approach outperforms the GTF-approach, but not significantly.

Using the GTF-operator in combination with the PPX-operator reduces on average the required makespan with 2.8%, indicating the usefulness of the GTF-operator. A comparison of the results obtained with the known optimal solutions shows that both approaches do not perform so well, except for the benchmark problems swv16, swv17, swv18 and swv20 where the GTF-approach obtains the optimum in all runs. This comes as no surprise as these benchmark problems are known to be very easy to solve. The worst results are achieved for the benchmark problems swv11-swv15, which have the reputation to be notoriously difficult to solve.

These moderate results, as compared to known optimal and best-known solutions, might be explained by the elementary implementation of the GA. Although largely inspired by the implementation of [10] we did not consider the generation of hybrid schedules. Bierwirth and Mattfeld indicate that this procedure might positively influence the solution quality [10].

In table 2 an overview is given of the average number of generations the GTF approach uses. A general relationship exists between the number of jobs and the number of generations the GA performs within the allowed number of chromosome evaluations. A larger number of jobs goes with a smaller number of generations. This is due to the fact that the GTF crossover operator creates as much intermediate offspring as there are operations in the largest conflict set. A higher

number of jobs leads to a higher probability that there are more operations in this conflict set. For the benchmarks swv11-swv20 the number of generations is extremely low. This result indicates that the value of the parameter P_{gtf} should be set with great care. By varying the value of P_{gtf} the number of generations in a given time frame can be controlled.

5 Conclusions and Future Research

In this paper a new crossover operator has been developed, called the Giffler-Thompson Focused (GTF) crossover, which exploits problem specific information generated by the Giffler-Thompson algorithm. The development of this operator is inspired by a finding of Hart and Ross who remark that the conflict sets generated during subsequent iterations of the GT-algorithm vary in size, and that genetic operators should best focus on genes with large conflict sets [24]. Our GTF crossover therefore sets as a crossover point the gene with the largest conflict set in the GT-algorithm, generates a number of offspring equal to the number of operations in the conflict set and returns the best candidate.

Our operator is tested on a number of benchmark problems known in literature and its performance is tested to a setting in which we use the PPX-operator developed by Bierwirth and Mattfeld [10]. Our operator always outperforms the PPX-operator except for one benchmark problem. Compared to the known optimal or best-known solutions the proposed GA does not perform very well, which may be due to the fact that the implementation of the GA is rather elementary. The experimental setting allows however to assess the value of the GTF-operator. The proposed GTF-operator is flexible as it can be adapted for the dynamic JSP and it can be integrated in different GA applications for the JSP. It also allows for multiple objective functions.

We believe that opportunities for improvement exist in the design of the GA. In our experiment the value of P_{gtf} has been arbitrarily fixed at 0.5. We showed that a relation exists between the value of P_{gtf} and the number of generations performed in a given time frame. A high value of P_{gtf} reduces the number of generations due to an increased number of jobs to evaluate in one generation, maybe limiting the GA to perform a thorough investigation of the search space. It might be useful to vary the value of P_{gtf} during the optimisation process. The process could start with a low value of P_{gtf} and increase it with the number of generations or when the GA converges, resulting in a more thorough search.

Also a more intelligent choice of the crossover point might improve the results. In our experiment we always chose the first largest conflict set. Integration with a tabu list of crossover points also might guide the GTF crossover operator to the more interesting parts of a chromosome, avoiding a premature population convergence.

References

[1] D. Applegate, W. Cook, A computational study of the job-shop scheduling problem, ORSA Journal on Computing 2 (1991) 149-156.

[2] J. Adams, E. Balas and D. Zawack, The shifting bottleneck procedure for job shop scheduling,

Management Science 34 (1988) 391-401.

[3] E. Balas, An additive algorithm for solving linear programs with zero-one variables, Operations Research 13 (1965) 517-546.

[4] E. Balas, Discrete programming by the filter method, Operations Research 15 (1967) 915-957.

[5] E. Balas, Machine sequencing via disjunctive graphs: An implicit enumeration algorithm, Operations Research 17 (1969) 1-10.

[6] E. Balas, Machine sequencing: disjunctive graphs and degree-constrained subgraphs, Naval Research Logistics Quarterly 17 (1970) 941-957.

[7] K. Baker, *Introduction to Sequencing and Scheduling*, John Wiley & Sons, New York, 1974.

[8] J. Bean, Genetic algorithms and random keys for sequencing and optimization, ORSA Journal on Computing 6 (1994) 154-160.

[9] C. Bierwirth, A generalized permutation approach to job shop scheduling with genetic algorithms, OR Spektrum 17 (1995) 87-92.

[10] C. Bierwirth and D.C. Mattfeld, Production Scheduling and Rescheduling with Genetic Algorithms, Evolutionary Computation 7 (1999) 1-17.

[11] L. Davis, Job shop scheduling with genetic algorithms, in: Grefenstette J. (Ed.), Proceedings of the First International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, Mahwah, NJ, 1985, pp. 136-140.

[12] F. Della Croce, R. Tadei and G. Volta, A genetic algorithm for the job shop problem, Computers and Operations Research 22 (1995) 15-24.

[13] U. Dorndorf. and E. Pesch, Evolution based learning in a job shop scheduling environment, Computers and Operations Research 22 (1995) 24-40.

[14] H. Fang, P. Ross and D. Corne, A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems, in: S. Forrest (Ed.), Proceedings of the Fifth International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, 1993, pp. 375-382.

[15] H. Fang, Genetic Algorithms for solving production scheduling problems, Ph.D. Thesis, Department of Artificial Intelligence, University of Edinburgh, Scotland, 1994.

[16] S. French, *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop*, Wiley & Sons, Chichester, England, 1987.

[17] M. Garey, D. Johnson and R. Sethi, The complexity of flow-shop and job-shop scheduling, Mathematics of Operations Research 1 (1976) 117-129.

[18] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman & Co., New York, 1979.

[19] M. Gen and R. Cheng, *Genetic Algorithms and Engineering Design*, John Wiley & Sons, New York, 1997.

[20] W.S. Gere, Heuristics in job-shop scheduling, Management Science 13 (1966) 167-190.

[21] J. Giffler and G.L. Thompson, Algorithms for solving production scheduling problems, Operations Research 8 (1960) 487-503.

[22] J. Grefenstette, R. Gopal, B. Rosmaita and D. Van Gucht, Genetic algorithms for the TSP, in: J. Grefenstette (Ed.), Proceedings of the First International Conference on Genetic Algorithms and their Applications, Lawrence Erlbaum, Hillsdale, NJ, 1985, pp. 160-168.

[23] J. Grefenstette, Incorporating problem specific knowledge into genetic algorithms, in: L.D. Davis (Ed.), Genetic Algorithms and Simulated Annealing, Pitman, London, 1987.

[24] E. Hart and P. Ross, A Heuristic Combination Method for Solving Job-Shop Scheduling Problems, in: A. Eiben, T. Bck, H. Schwefel, M. Schoenauer (Eds.), Parallel Problem Solving from Nature V, Springer Verlag, Berlin, 1998, pp. 845-854.

[25] J. Heizer. and B. Render, *Production and Operations Management: Strategies and Tactics (3rd ed.)*, Allyn & Bacon, Boston, 1993.

[26] J. Holland, *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor, 1975.

[27] C. Holsapple, V. Jacob, R. Pakath and J. Zaveri, A genetic-based hybrid scheduler for generating static schedules in flexible manufacturing contexts, IEEE Transactions on Systems, Man and Cybernetics 23 (1993) 953-971.

[28] A. Jain and S. Meeran, A state-of-the-art review of job-shop scheduling techniques, Technical report, Department of Applied Physics, Electronic and Mechanical Engineering, University of Dundee, Scotland, 1998.

[29] S. Lawrence, Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (Supplement), Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 1984.

[30] S.-C. Lin, E.D. Goodman and W.F. Punch, Investigating parallel genetic algorithms on job shop scheduling problems, in: P.K. Angeline, R.G. Reynolds, J.R. McDonnell, R. Eberhart (Eds.), Evolutionary Programming VI, Springer, Berlin, 1997, pp. 383-393.

[31] R. Nakano and T. Yamada, Conventional genetic algorithms for job-shop problems, in: R. Belew, L. Booker (Eds.), Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufman, San Mateo, 1991, pp. 447-479.

[32] N. Raman, R.V. Rachamadagu and F.B. Talbot, Real-time scheduling of an automated manufacturing center, European Journal of Operational Research 40 (1989) 115-132.

[33] V. Srinivasan, A hybrid algorithm for the one machine sequencing problem to minimize total tardiness, Naval Research Logistics Quarterly 18 (1971) 317-327.

[34] R. Storer, S. Wu and R. Vaccari, New search spaces for sequencing problems with application to job shop scheduling, Management Science 38 (1992) 1495-1510.

[35] H. Tamaki and Y. Nishikawa, A paralleled genetic algorithm based on a neighborhood model and its application to the jobshop scheduling, in: R. Männer, B. Manderick (Eds.), Parallel Problem Solving from Nature II, Springer Verlag, Berlin, 1992, pp. 573-582.

[36] M. Vazquez and L.D. Whitley, A Comparison of Genetic Algorithms for the Dynamic Job Shop Scheduling Problem, in: L.D. Whitley, D.E. Goldberg, E. Cantú-Paz, L. Spector, I.C. Parmee, H.-G. Beyer (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000), Morgan Kaufmann, 2000, pp. 1011-1018.

[37] T. Yamada and R. Nakano, A genetic algorithm applicable to large-scale job-shop problems in: R. Männer, B. Manderick (Eds.), Parallel Problem Solving from Nature II, Springer Verlag, Berlin, 1992, pp. 281-290.

Table 1: Results on Benchmark Problems

	Setting 1	Setting 2			Time(sec)	Bound	S2 vs opt
	Average	Average	%	# gener			
orb01	1202.0	1182.4	*1.7%	47.3	1.5	1059	11.7%
orb02	976.4	950.5	*2.7%	54.1	1.5	888	7.0%
orb03	1227.7	1144.7	*7.3%	51.9	1.5	1005	13.9%
orb04	1116.3	1088.5	*2.6%	33.5	1.5	1005	8.3%
orb05	997.1	957.7	*4.1%	47.9	1.5	887	8.0%
orb06	1163.9	1118.4	*4.1%	45.7	1.5	1010	10.7%
orb07	433.0	424.0	*2.1%	54.0	1.5	397	6.8%
orb08	1056.9	1000.5	*5.6%	47.6	1.5	899	11.3%
orb09	1009.2	988.9	*2.1%	33.5	1.5	934	5.9%
orb10	1081.4	1035.7	*4.4%	47.7	1.5	944	9.7%
la26	1338.3	1317.9	*1.5%	43.6	5.5	1218	8.2%
la27	1457.0	1401.7	*3.9%	45.5	5.5	1235	13.5%
la28	1394.0	1355.1	*2.9%	45.7	5.5	1216	11.4%
la29	1417.3	1357.3	*4.4%	36.2	5.7	1152	17.8%
la30	1465.6	1432.4	*2.3%	36.1	5.7	1355	5.7%
la36	1421.9	1378.3	*3.2%	50.2	5.4	1268	8.7%
la37	1583.5	1564.5	*1.2%	43.3	5.5	1397	12.0%
la38	1387.9	1363.8	*1.8%	49.4	5.5	1196	14.0%
la39	1426.0	1390.9	*2.5%	52.2	5.4	1233	12.8%
la40	1354.0	1339.0	*1.1%	54.9	5.5	1222	9.6%
swv11	4080.7	3929.4	*3.9%	16.9	51.0	2983	31.7%
swv12	4085.2	3940.2	*3.7%	19.5	49.0	2972	32.6%
swv13	4158.7	3969.3	*4.8%	16.5	50.0	3104	27.9%
swv14	3972.1	3856.8	*3.0%	16.6	49.5	2968	29.9%
swv15	3964.4	3824.7	*3.7%	15.3	49.2	2885	32.6%
swv16	2924.0	2924.0	0.0%	21.3	44.9	2924	0.0%
swv17	2794.0	2794.0	0.0%	19.8	45.6	2794	0.0%
swv18	2852.0	2852.0	0.0%	23.0	45.4	2852	0.0%
swv19	3033.0	2921.1	*3.8%	21.7	43.4	2843	2.7%
swv20	2823.6	2825.3	-0.1%	20.7	45.6	2823	0.1%

Table 2: Number of Generations using the GTF-approach

	jobs	machines	# generations
orb01-orb10	10	10	46.3
la26-la30	20	10	41.4
la36-la40	15	15	50.0
swv11-swv20	50	10	19.1