A Generalized Permutation Approach to Job Shop Scheduling with Genetic Algorithms*

Christian Bierwirth

Department of Economics, University of Bremen, Germany

Abstract. In order to sequence the tasks of a job shop problem (JSP) on a number of machines related to the technological machine order of jobs, a new representation technique – mathematically known as "permutation with repetition" is presented. The main advantage of this single chromosome representation is – in analogy to the permutation scheme of the traveling salesman problem (TSP) – that it cannot produce illegal sets of operation sequences (infeasible symbolic solutions). As a consequence of the representation scheme a new crossover operator preserving the initial scheme structure of permutations with repetition will be sketched. Its behavior is similar to the well known Order-Crossover for simple permutation schemes. Actually the **GOX** operator for permutations with repetition arises from a **G**eneralisation of **OX**. Computational experiments show, that GOX passes the information from a couple of parent solutions efficiently to offspring solutions. Together, the new representation and GOX support the cooperative aspect of the genetic search for scheduling problems strongly.

1 Introduction

Since the publication of Davis paper in 1985 [3] a lot of research in the field of production scheduling with Genetic Algorithms (GAs) has been done.

The main difficulty in this subject arises from the question of how to represent the problem in the algorithm, which is known to be most important for genetic search. Some researchers approached the JSP from a TSP's point of view, e. g. Whitley/Starkweather [13]. Beside a fundamental obstacle – that will be discussed and removed by this paper – two important advantages arise from the connectivity of both problem areas. It allows to handle the JSP in a natural way as an ordering problem and it enables the use of a wide range of genetic operators, originally designed to tackle the TSP.

Within the last years other approaches with a stronger connection to scheduling domains were developed. In particular hybrid algorithms, combining local genetic search and scheduling heuristics, lead to (near-)optimal solutions for many large-scale benchmark problems, see Nakano/Yamada [8][14], Dorndorf/Pesch [4], Pesch [10], Storer/Wu/Vaccari [11]and Storer/Wu/Park [12].

* Supported by the Deutsche Forschungsgemeinschaft (Project Parnet)

In order to compare a new TSP-oriented approach with these algorithms the paper refers to a standard model of the general n-job, m-machine job shop problem [5], denoted by

$$n/m/G/* \ .$$

The parameter G indicates that jobs are connected with technological production rules, describing their processing order of machines. G is specified as a matrix

$$\mathcal{T} = \Big[\mathcal{T}_i\Big]_{1\leq i\leq n}, \qquad \mathcal{T}_i = \Big(M_{\phi_i(1)}, M_{\phi_i(2)} \ldots M_{\phi_i(m)}\Big)$$

of vectors $\mathcal{T}_i$, representing a technological rule of job J_i, which is a permutation of all (or a subset of all) machines. In case of a proper subset J_i is processed by $m_i < m$ machines, thus the last $m - m_i$ components of vector $\mathcal{T}_i$ are defined as "zero-machines" indicating no further treatment of the job J_i. The operation of J_i which is processed by M_j is denoted as O_{ij} and its processing time as p_{ij}.

The symbol "$*$" calls a performance-measure of optimization. In order to compare computational results with results from literature, this paper discusses the most common performance-measure – the minimum-makespan – denoted as C_{max}. If C_i denotes the completion time of J_i, "$* = C_{max}$" means to find a schedule of all operations O_{ij} that minimizes the completion time of the latest job, i. e. the makespan of a whole production program. Apart from that, the presented approach to job shop scheduling supports the handling of other performance-measures by small modifications only.

Usually the process of solving the JSP is decomposed into a sequencing process of operations on machines (according to the rules of $\mathcal{T}$) and a scheduling process of specifying starting times for all operations. That is why solutions of the JSP can be represented symbolically, e. g. this is done by specifying operation orders $\mathcal{S}_j$ for each machine M_j, by a single task sequence determining the scheduling order of all operations or by a set of binary variables defining the processing order of each of two operations on identical machines.

The difficulty of using any of these representations as a genetic coding scheme arises from the fact, that syntactically valid instantiations (generated by genetic operators) often get into conflict with the production rules of $\mathcal{T}$. In such a situation a symbolic solution cannot be transferred to a schedule (the scheduling process runs into a deadlock!), hence it is infeasible.

To overcome this situation, a new way to represent the JSP is presented in section 2. Section 3 discusses the process of schedule-building. As a consequence of the representation a new genetic operator is sketched in section 4. Section 5 reports computational results on the famous Muth-Thompson problems and 10 large-scale benchmark problems provided by Applegate and Cook. A comparison with other GAs shows the new genetic search strategy to be a promising approach for solving scheduling problems in general.

2 Genetic Representation of the JSP (Permutations with Repetition)

If it is impossible to represent an optimization problem with standard coding techniques (binary or permutation) in a way, that infeasible solutions cannot get into the coding scheme, three types of remedy are distinguished in literature:

1. A detected infeasible genotype can be penalized with a relatively "bad" fitness value to drive the individual out from the population.
2. A detected infeasible genotype can be transformed into a similar feasible genotype by a small modification of the string representation.
3. A non-standard representation can be designed, which avoids the coding of infeasible solutions. If ordinary genetic operators destroy the scheme-structure of the representation, new operators (at least a crossover) preserving this structure, have to be developed.

The first alternative can be ruled out to handle the JSP. In general there exist much more infeasible symbolic solutions than feasible ones. That is why it is pointless to reject infeasible solutions.

Most of the so far known approaches to the JSP use the second alternative, e. g. the decoding/encoding strategy from Davis, the local and global harmonization of Nakano/Yamada or the Priority-rule based GA of Dorndorf/Pesch are of this type.

Well known examples for type 3 are the early applications of GAs to ordering problems, in particular the TSP. The binary representations of this problem led to difficulties which could be avoided by the use of a simple permutation representation. As a consequence many permutation-crossover operators were developed; for a comparison see Oliver/Smith/Holland [9].

This section presents a new genetic representation of the JSP, which follows the third alternative. The representation is "complete" in the way, that it covers all feasible solutions of a JSP but no infeasible one. It is based on a generalization of the concept of permutation and therefore respects the main characteristic of the JSP as a complex ordering problem.

A simple permutation is defined as an ordering of the elements of a finite set I of size n. An ordering in which elements may appear more than once is called permutation with repetition. The number of different permutations with repetition that contain element $i \in I$ exactly m_i times is given by

$$\frac{(m_1 + m_2 + \ldots m_n)\,!}{m_1! \cdot m_2! \cdot \ldots m_n!}\,.$$

If $m_i = 1$ for all $i \in I$ this number is equal to $n!$, which is just the number of different simple permutations of size n.

In the standard job shop model n denotes the number of jobs J_i under the index set I. If m_i denotes the number of tasks of J_i, a permutation with repetition of three jobs that contains $m_i = 3, 4, 3$ tasks, is for example given by

$$\Big(J_1, J_2, J_2, J_1, J_3, J_1, J_2, J_3, J_2, J_3\Big) .$$

Here J_2 has to be processed on all machines whereas J_1 and J_3 have to be processed only by three of them. To obtain a feasible solution, the permutation with repetition is interpreted as a task sequence

$$\Big(T_{11}, T_{21}, T_{22}, T_{12}, T_{31}, T_{13}, T_{23}, T_{32}, T_{24}, T_{33}\Big) .$$

Reading it from left to right hand side, a task T_{ij} of job J_i has to be scheduled on machine $M_{\phi_i(j)}$ as determined by the technological order $\mathcal{T}_i$. The procedure is controlled by counting-indices of the matrix $\mathcal{T}$ and a symbolic solution $\mathcal{S}$.

Figure 1 illustrates the phenotype evaluation of the example above for three jobs A, B and C in detail. The pseudo C-code shows the program of the whole procedure. It is the only module of the GA that takes access to problem specific data ($\mathcal{T}$ and $[p_{ij}]$) indicated by the dashed box.

In the sketched state 6 of 10 tasks are already scheduled. They are shown in the gantt-chart (e). The chromosome (a) indicates the 7th task to be an operation of job B. The counting-index `T.next[B]` identifies the task to be the third task of B. According to table (b) it has to be processed on M_3. Hence – using the denotation of section 1 – it leads to an operation O_{B3}. In the optional generated symbolic solution $\mathcal{S}$ this would be the second operation of M_3. After sequencing the operation, a starting time has to be scheduled for O_{B3}. The procedure uses a semiactive schedule-builder, which starts each operation immediately after the longer completion time of the last scheduled task of the same job or the same machine. In the example O_{B3} starts immediately after O_{B2}, hence an idle time of one unit results on M_3.

3 Schedule Building

The schedule-builder is a module of the evaluation procedure and should be chosen with respect to the performance-measure of optimization. Most of the important performance-measures of the JSP are regular measures, which means that optimal solutions are always semiactive, see French [5]. That is why semiactive scheduling allows to generate optimal solution from the new representation for a wide range of performance-measures.

Computational experiments showed, that genetic optimization of the minimum-makespan-JSP improves by the use of a more powerful schedule-builder, in particular an active scheduler. This observation is in common with Dorndorf/Pesch and Nakano/Yamada, who used the Giffler-Thompson algorithm [6] as an active scheduling-procedure.

An active schedule-builder performs a kind of local-search, which can be used to introduce heuristic-improvement into genetic search. This strategy, first used by Nakano/Yamada, is called forcing. Forcing allows the schedule-builder to modify a chromosome if a permissible left shift of a task is detected. An active scheduler only performs the left shift to build up the solution (ganttchart).

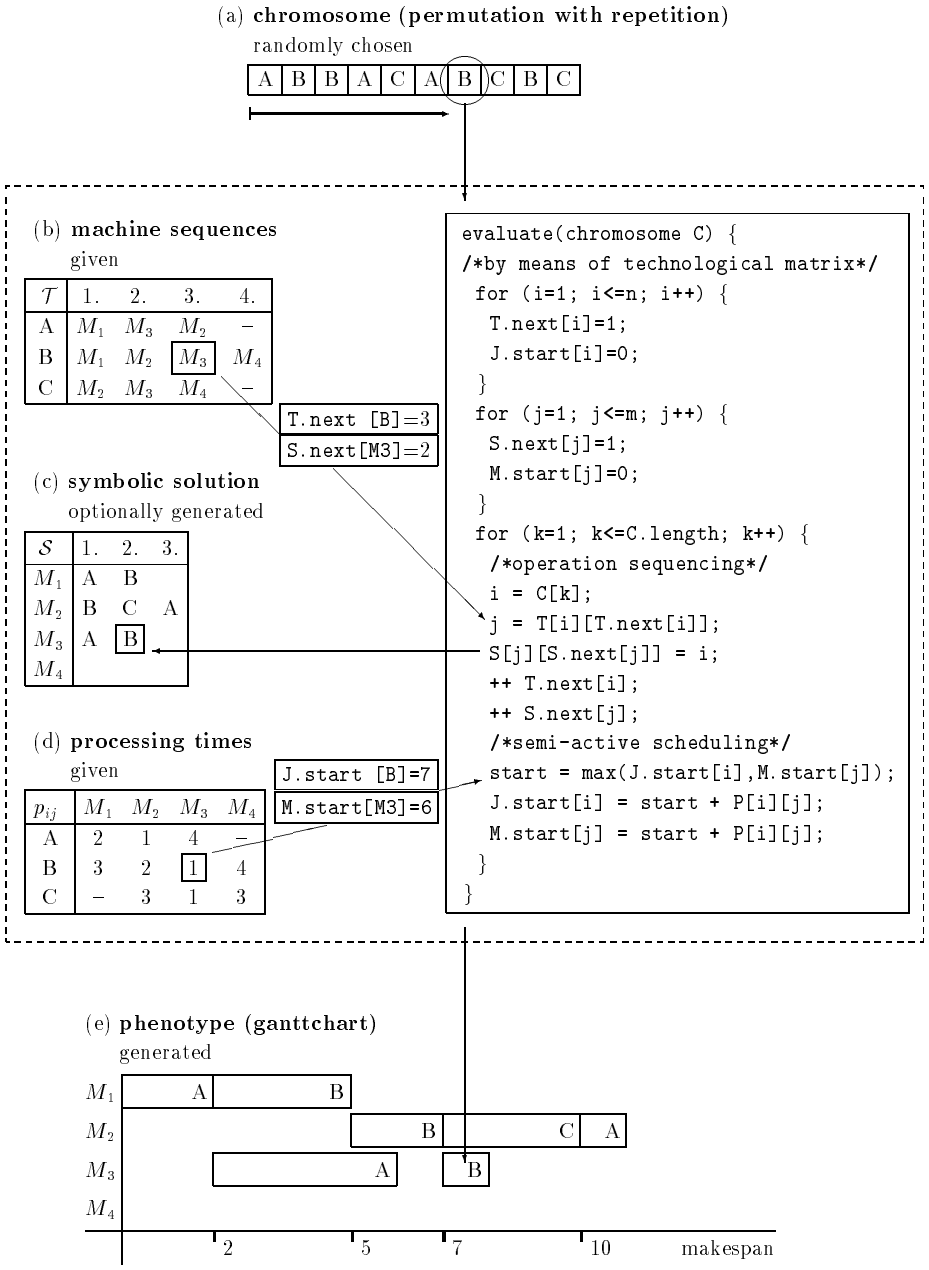

Fig. 1. Phenotype evaluation of a permutation with repetition.

Above that an active scheduler with forcing does the corresponding shift on the evaluated chromosome.

To illustrate forcing the ganttchart (e) of figure 1 is used. The first task that permits a left shift is the first operation of job C processed on M_2. The active scheduler detects a shift, which permits O_{C2} to be placed before O_{B2}. Forcing causes the corresponding shift in the chromosome (a).

$$\text{A B B A } \boxed{\text{C}} \text{ A } \ldots \quad \longmapsto \quad \text{A B } \boxed{\text{C}} \text{ B A A } \ldots$$

4 Crossover for Permutations with Repetition

Crossover can be regarded as the backbone of genetic search. It intends to inherit nearly half of the information of two parent solutions to one or more offspring solutions. Provided that the parents keep different aspects of high quality solutions, crossover induces a good chance to find still better offspring.

In order to design a crossover for permutations with repetition it is important to focus on two properties of the new operator. First it has to respect the repetition-structure of a certain permutation. Secondly it has to preserve the coded characteristics of solutions as far as possible to guarantee the inheritance-feature. Because the assignment of jobs to machines happens during the phenotype evaluation, the second property cannot be realized by a simple transmission of subsequences from parent-chromosomes to offspring alone. As an additional desirable feature the crossover operator should enable inheritance of counting-indices of genes within subsequences whenever possible.

For the first reason the new operator was modeled on the Order-Crossover (OX) for simple permutation schemes [9]. OX is known to transmit the relative node positions of two permutations quite well to offspring. A straight-forward **G**eneralisation of **OX** to permutations with repetition allows the integration of a technique, which includes the inheritance-feature into the operator **GOX**. To illustrate GOX the chromosome of figure 1 is used. It functions as parent1 in a recombination process.

Parent1:	B A B B C A C C B A
Index:	1 1 2 3 1 2 2 3 4 3

The index line shows a counter for the occurrence of all genes in the chromosome. The second chromosome functions as donator (parent2) of a crossover-string. To determine the crossover-string an offset position is randomly chosen within the donator chromosome. Implanting a crossover-string into the receiver chromosome (parent1) requires a preparation which usually causes some loss of information. Therefore the length of the crossover-string is randomly chosen in between one half and one third of the total length of a chromosome. For that both parents can inherit nearly the same amount of information to the offspring. The determined crossover-string either lies inside the donator chromosome or it is wrapped around.

If the crossover-string lies inside the donator chromosome its indices have to be calculated.

Parent2: A B B A C A B C B C
Index: 2 1 3 3

Afterwards all genes in the receiver that occur in the crossover-string with the same index are marked. To determine a position where to implant the string, a method known from OX is used. The receiver chromosome gets a cut after that gene, which is equal to the first gene (and its index) of the crossover-string. Now the string can be implanted into parent1's chromosome.

Parent1: B A B [B] [C] [A] A C A B C C B [A]
Index: 1 1 2 3 1 2 2 3 4 3

After deleting the marked genes in parent1, a new offspring chromosome results. Using this technique it may happen that the indices of some genes get displaced. Imagine a constellation in which gene "C" occurs two times before the chromosome of parent1 is cut after the second A-gene. Now the C-gene from the crossover-string cannot be realized with index 1 in the offspring, actually it will get the index 2. Hence GOX introduces an implicit mutation into the offspring chromosome. But the choice of the crossover-cut takes care of this to be rare. In our example the information of both parents is totally inherited to the offspring.

Offspring: B A B A C A B C C B
Index: 1 1 2 2 1 3 3 2 3 4

If the crossover-string wraps around the boundary points of the donator chromosome the whole procedure is simplified very much. Again the indices of all genes in the chosen string have to be calculated.

Parent2: A B B A C A B C B C
Index: 1 1 4 3

After marking the genes of the crossover-string in parent1, it gets implanted into the receiver chromosome at the same position as it has in the donator chromosome.

Parent1: A B [B] [A] B B C A C [C] [B] A B C
Index: 1 1 2 3 1 2 2 3 4 3

This guarantees the indices of the implanted string to remain valid in the new chromosome, which is easy to verify after deleting the marked genes in parent1.

Offspring: A B B B C A C A B C
Index: 1 1 2 3 1 2 2 3 4 3

Choosing the crossover-cut in case of an inner crossover-string in the same way, i. e. keeping the donator position of the string with respect to the receiver, will usually lead to a higher rate of implicit mutations. Computational experiments with an operator of this kind showed a weaker performance than the presented GOX operator.

As we remarked GOX arises from a generalization of the OX operator. That is why the application of GOX to simple permutations (all counting-indices are 1) generates the same offspring than OX does.

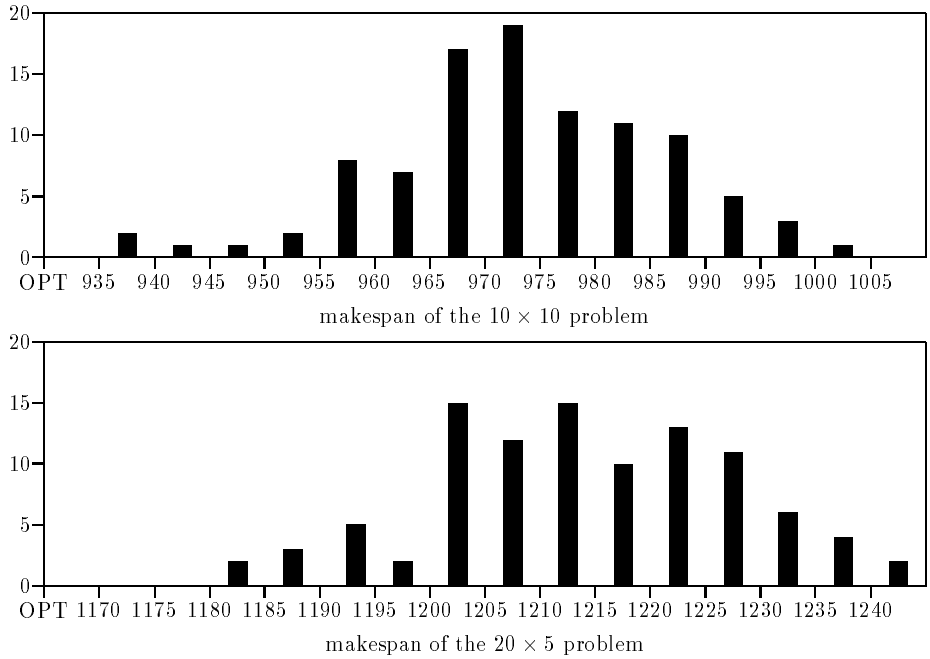

Fig. 2. Distribution of 100 solutions of the Muth-Thompson benchmarks.

5 Computational Results

The described representation and GOX were embedded into a distributed genetic optimization environment called Parnet. The distribution model of Parnet bases on "isolation-by-distance" within a structured population (for details see Bierwirth/Kopfer/Mattfeld/Utecht [2]). In this population model individuals reside on a ring. Mate-selection is done by neighborhood ranking and offspring-acceptance is done locally. The recombination strategy works as follows: Crossover via GOX is only performed if the integer valued fitness of mating individuals differs, otherwise a position-based mutation takes place.

The derived Generalized-Permutation GA (GP-GA) was tested on a suite of 12 well-known benchmark problems provided by Applegate and Cook [1]. In all runs the population size was set to 100. Selection is done by the ranking scheme "40%, 30%, 20%, 10%" in a 4-element neighborhood. The accepted worsening of offspring was set to 1%. Fitness evaluation of solutions is done via active scheduling plus forcing as sketched in section 3. For the moderate Muth-Thompson problems termination was set to 100 generations and for ten larger problems it was set 150 generations.

The first test of GP-GA was run for a total of 100 iterations on the Muth-Thompson $10/10/G/C_{max}$ and $5/20/G/C_{max}$ problems. The $6/6/G/C_{max}$ problem was always solved to optimality ($C_{max} = 55$) and is therefore not docu-

1963	Muth-Thompson	Test Problems	10x10	20x5
1991	Nakano/Yamada	Conventional GA	965	1215
1992	Yamada/Nakano	Giffler-Thompson GT-GA	930	1184
	Dorndorf/Pesch	Priority-Rule based P-GA	960	1249
	Dorndorf/Pesch	Shifting-Bottleneck SB-GA	938	1178
1993	Pesch	Job-Pair based 2J-GA	937	1193
	Pesch	Job-Constraint propagation JC-GA	937	1175
	Pesch	Machine-Constraint propagation MC-GA	930	1165
	Storer/Wu/Park	Problem-Space GA	954	1180
1994		Generalized-Permutation GP-GA	936	1181

Table 1. Comparison of results (makespan) on the Muth-Thompson problems.

Test Problems			GP-GA		Comparison				
No.	Size	Known	Best	Average	SB-GA	2J-GA	JC-GA	MC-GA	P-GA
la26		*1218	1232	1252.5	1219	1445	1218	1218	1278
la27	20	1236	1269	1298.7	1272	1271	1268	1268	1378
la28	×	*1216	1256	1271.9	1240	1271	1240	1234	1327
la29	10	1180	1233	1264.9	1204	1226	1209	1204	1336
la30		*1355	1355	1365.0	1355	1359	1355	1355	1411
la36		*1268	1315	1327.1	1317	1289	1311	1273	1373
la37	15	1402	1447	1481.2	1484	1507	1454	1414	1489
la38	×	1201	1251	1287.3	1251	1325	1292	1204	1296
la39	15	*1233	1251	1286.8	1282	1287	1275	1233	1351
la40		*1222	1252	1271.1	1274	1301	1291	1242	1321

*optimal

Table 2. GP-GA results and comparison on 10 large-scale job shop problems.

mented. In all tests GP-GA were run for a total of 10.000 calls to the schedule-builder. The best GP-GA solutions and a comparison to other approaches are shown in table 1. As shown in figure 2, all generated solutions fall approximately into an interval within a deviation of 1% to 7% from optimum, which is inside the range of comparable GAs.

In order to explore how the Generalized-Permutation approach "scales up" to larger problems, GP-GA were now run for a total of 25 iterations on five $20/10/G/C_{max}$ and five $15/15/G/C_{max}$ benchmark problems. In each test run GP-GA made a total of 15.000 calls to the schedule-builder. Computational results appear in table 2. The column "Known" refers to the best-known or optimal solutions of the ten problems. Seven values are taken from a recent paper of Pesch [10]. Meanwhile la29 was solved by Tabu-Search with a makespan of 1180. This result was achieved by Taillard and is – as far as we know – still unpublished. For la27 and la38 new best solutions were recently found by a GA. This algorithm uses the Generalized-Permutation approach presented in this paper. But in contrast to GP-GA it is hybridized by more powerful local-search techniques [7].

Table 2 shows the best-of-all and the average makespan generated by GP-GA. The average solutions of all problems are within 0.7% to 7.2% of the best-known values whereas the best-found solutions differ by only 0% to 4.5%. Thus the algorithm scales performance from previously treated moderate problems to larger ones in a promising fashion. Notice that GP-GA results are achieved in combination with the relatively weak base heuristic of active scheduling. The comparison in table 2 reports on the performance of five other GAs (Dorndorf/Pesch and Pesch) in the same test-suite. At least SB-GA, JC-GA and MC-GA incorporate stronger base-heuristics into genetic search than GP-GA does. Nevertheless, the performance of GP-GA is still inside the range of SB-GA and JC-GA.

To summarize, the coding of scheduling applications by permutations with repetition leads to efficient local genetic search. Apart from that, successful genetic search in large-scale optimization still requires support from strong problem-specific heuristics.

6 Conclusion

This paper develops a genetic representation for scheduling problems, that is closely related to recent GA applications to sequencing problems. Together with a new crossover operator, our representation demonstrates sustained performance on a platform of difficult benchmark problems. Without tuning and weak hybridization only, the algorithm reaches a quality level of optimization within the range of other GA approaches. But in contrast to these methods the Generalized-Permutation approach offers a high-level of abstraction which enables its diversification to a broad range of related problems.

References

1. Applegate, D., Cook, W.: A Computational Study of the Job-Shop Scheduling Problem. ORSA Journal on Computing **2** (1991) 149–156
2. Bierwirth, C., Kopfer, H., Mattfeld, D., Utecht, T.: PARNET: Distributed Realization of Genetic Algorithms in a Workstation Cluster. Int. Symp. on Applied Mathematical Programming, Preprints (1993) 41–48
3. Davis, L.: Job Shop Scheduling with Genetic Algorithms., Proc. of 1st Int. Conf. on Genetic Algorithms, Lawrence Erlbaum Associates (1985) 136–140
4. Dorndorf, U., Pesch, E.: Evolution based Learning in a Job Shop Scheduling Environment. Research Memorandum **92-019**, University of Limburg (1992)
5. French, S.: Sequencing and Scheduling - An Introduction to the Mathematics of the Job-Shop. John Wiley (1982)
6. Giffler, B., Thompson, G.: Algorithms for solving production-scheduling problems. Operations Research **8** (1960) 487–503
7. Mattfeld, D., Kopfer, H., Bierwirth, C.: Control of Parallel Population Dynamics by Social-like Behavior of GA-Individuals. Proc. of 3rd Int. Workshop on Parallel Problem Solving from Nature, Springer-Verlag (to appear)
8. Nakano, R., Yamada, T.: Conventional Genetic Algorithm for Job Shop Problems. Proc. of 4th Int. Conf. on Genetic Algorithms, Morgan Kaufmann Publishers (1991) 474–479

9. Oliver, I., Smith, D., Holland J.: A Study of Permutation Crossover Operators on the Traveling Salesman Problem. Proc. of 2nd Int. Conf. on Genetic Algorithms, Lawrence Erlbaum Associates (1987) 224–230
10. Pesch E.: Machine Learning by Schedule Decomposition. Working Paper, University of Limburg (1993)
11. Storer, R., Wu, D., Vaccari, R.: New Search Spaces for sequencing Problems with Application to Job Shop Scheduling. Management Science **38** (1992) 1495–1509
12. Storer, R., Wu, D., Park, I.: Genetic Algorithms in Problem Space for Sequencing Problems, Operations Research in Production Planning and Control, Springer-Verlag (1993) 584–598
13. Whitley, D., Starkweather, T., Fuquay, A.: Scheduling Problems and Traveling Salesman: The Genetic Edge Recombination Operator. Proc. of 3rd Int. Conf. on Genetic Algorithms, Morgan Kaufmann Publishers (1989) 133–140
14. Yamada, T., Nakano, R.: A Genetic Algorithm applicable to large-scale Job Shop problems. Proc. of 2nd Int. Workshop on Parallel Problem Solving from Nature, North-Holland (1992) 281–290

This article was processed using the LaTeX macro package with LLNCS style