Scalable Search Spaces for Scheduling Problems

Dirk C. Mattfeld
Dept. of Economics
University of Bremen
28334 Bremen, Germany

Abstract

This paper presents a tunable decoding procedure for Genetic Algorithm (GA) based schedule optimization. Tuning the decoding procedure of the algorithm scales the size of the search space of the underlying optimization problem. We show by experiment, that a tradeoff exists between the advantage of searching a small space and the impacts of excluding near-optimal solutions from being searched at the same time. Thereby we reveal the weakness of GAs when coping with large and difficult search spaces. Finally we discuss the deficiencies observed by tuning the decoding procedure auto-adaptively and leave this matter as an open issue.

1 Introduction

A common way to improve GA performance focuses on tuning the parameters of the algorithm, i.e. to enlarge the population size, to alter the selection pressure and so forth. If all the parameters are already within useful bounds, the promised progress for solving a single problem instance usually does not justify the costs of finding an even more appropriate fine tuning. More universal approaches are going by modifications of the population management. E.g. spatial distributions of the population can lead to a better performance. Although coming along with slight improvements, such techniques are certainly not able to eliminate the weakness of adaptation concerning a particular problem class. Substantial improvements of the adaptation process require a modification of the scope of search. This can be achieved either by modifying the problem representation or by altering the decoding procedure. This paper is devoted to the latter intention.

Throughout the last decade researchers have devoted a vast number of publications to the task of scheduling resources over time. Despite some remarkable approaches, GAs are still outperformed by OR algorithms for most academic benchmark problems. On the other hand, for real world scheduling applications GAs have been shown to work sufficiently well, often because they are the only applicable search method at hand. In order to cope with the constraints of the real world, recent GA research has focused on the encoding/decoding scheme of the algorithm. We follow this line of research by taking up well known GA components from previous research, all of them capable to deal with complex constraints of the real world. In the resulting algorithm we incorporate a tunable decoding procedure yielding serious improvements of solution quality.

Any encoding of schedules can be decoded into semi-active, active, or non-delay schedules (Giffler and Thompson, 1960). The three properties of schedules can be achieved by incorporating domain knowledge at a variable extent, causing a reduction of accessible schedules, see Fig. 1. Furthermore, there is strong empirical evidence that for regular measures[1] the smallest of these sets —consisting of non-delay schedules— show a better average performance compared to the set of active schedules. However, as indicated in Fig. 1, there is not necessarily an optimal schedule in the set of non-delay schedules. On the other hand at least one schedule of optimal quality is known to be in the set of active ones. Since the set of active schedules is a true subset of the set of semi-active schedules, we confine the search to the set of active schedules without loosing the condition for optimality.

[1] A regular measure of a schedule does not deteriorate whenever the completion time of any of its jobs changes to an earlier date. Prominent regular measures are the minimization of the maximal completion time of jobs and the minimization of the mean flow time of jobs.

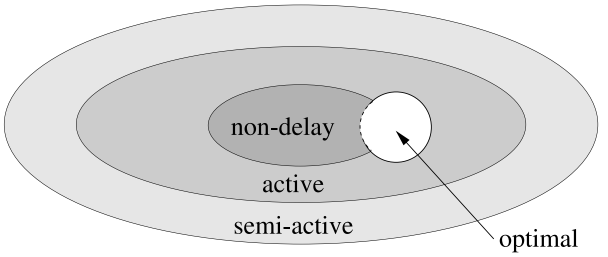

Figure 1: Relationships of schedule properties.

For this reason so far most GAs have incorporated an active schedule decoding, see e.g. Mattfeld (1996) for a comprehensive survey. Contrarily, a non-delay scheduler is used for decoding in the GA approach of Della Croce et al. (1995). They notice that a non-delay decoding procedure sometimes come up with a much better performance than an active one. These results motivate the idea to vary the scope of search for the decoding procedure. For this end we incorporate a tunable decoding procedure originally proposed by Storer et al. (1992) which scales the scope of the search in the range from non-delay to active schedules.

The above classification scheme of schedules can be applied for a wide array of scheduling problems whenever a regular measure of performance is pursued, e.g. Kolisch (1995). Without loss of generality we confine this research to the job shop scheduling problem (JSP), because it is well known, it has a rich literature, and many benchmarks exist (Błażewicz et al., 1996).

2 Job Shop Scheduling

We consider a manufacturing system consisting of m machines $M_1, \dots M_m$. A production program consists of n jobs $J_1, \dots J_n$ where the number of operations of job J_i $(1 \le i \le n)$ is denoted by m_i. Since all operations have to be processed by dedicated machines, μ_i defines a production recipe for job J_i. The elements of μ_i point to the machines, hence for $\mu_i(k) = j$ $(1 \le k \le m_i)$ machine M_j is the k-th machine that processes J_i. The k-th operation of job J_i processed on machine $M_{\mu_i(k)}$ is denoted as o_{ik}. The required processing time of o_{ik} is known in advance and denoted by p_{ik}.

A schedule of a production program can be seen as a table of starting times t_{ik} for the operations o_{ik} with respect to the production recipe of jobs. The completion time of an operation is given by its starting time plus its processing time $(t_{ik} + p_{ik})$. The earliest possible starting time of an operation is determined by the maximum of the completion times of the predecessor operations of the same job and of the same machine. Therefore the starting time for operation o_{ik} is calculated by $t_{ik} = \max\Big(t_{i,k-1} + p_{i,k-1}\,,\, t_{hl} + p_{hl}\Big)$. Here o_{hl} refers to the operation of job J_h which precedes o_{ik} on its machine, i.e. $\mu_i(k) = \mu_h(l)$. For $k = 1$ we deal with the beginning operation of a job and in this case the first argument in the max-function is set to zero. If o_{ik} is the first operation processed on a machine it has no machine predecessor and the second argument in the max-function is set to zero.

The completion time C_i of a job J_i is given by the completion time of its final operation o_{i,m_i}, i.e. $C_i = t_{i,m_i} + p_{i,m_i}$. As a measure of performance the minimization of $C_{\max} = \max\{C_1, \dots C_n\}$ is pursued.

3 Genetic Algorithm Components

Literature reports a large number of GA approaches to the job shop scheduling problem. In the following a new GA is described, which combines well known components adopted from previous research in the fields of Operations Research and Evolutionary Computation into a very efficient algorithm. First, the solution encoding and the genetic operators, all of them already proven successful, are sketched. Next the decoding procedure is introduced in detail, before finally the parameterization of the algorithm is described.

3.1 Encoding and Operators

Encoding. We use a permutation of all operations involved in an instance of a scheduling problem. The decoding procedure scans the permutation from left to right and consecutively assembles a schedule from this sequence of operations. Note that not all permutations represent feasible schedules directly. An operation o_{ik} found in the permutation is only schedulable if its job predecessor $o_{i,k-1}$ has been scheduled already. Therefore in a feasible permutation $o_{i,k-1}$ occurs to the left of o_{ik}. In case of infeasible permutations the decoding procedure ensures feasibility of the resulting schedule by delaying an operation until its job predecessor has been scheduled.

Crossover. In the following we describe the Precedence Preservative Crossover (PPX) which was independently developed for vehicle routing problems by Blanton and Wainwright (1993) and for scheduling problems by Bierwirth et al. (1996). The operator passes on precedences among operations given in two parental permutations to one offspring at the same rate, while no new precedences between operations are introduced. PPX is illustrated in Figure 2 for an instance consisting of six operations A-F.

parent permutation 1	A	B	C	D	E	F
parent permutation 2	C	A	B	F	D	E
select parent no. (1/2)	1	2	1	1	2	2
offspring permutation	A	C	B	D	F	E

Figure 2: Precedence Preservative Crossover.

A binary vector of the same length as the permutation is filled at random. This vector defines the order in which the operations are successively drawn from parent 1 and parent 2. We now consider the parent and offspring permutations and the binary vector as lists, for which the operations 'append' and 'delete' are defined. We start by initializing an empty offspring. Then the leftmost operation in one of the two parents is selected in accordance to the leftmost entry in the binary vector. After an operation is selected it is deleted in both parents and appended to the offspring. Finally the leftmost entry of the binary vector is also deleted. This procedure is repeated until the parent lists are empty and the offspring list contains all operations involved.

Mutation. In our GA we alter a permutation by first picking (and deleting) an operation before reinserting this operation at a randomly chosen position of the permutation. At the extreme all precedence relations of one operation to all other operations are affected. Typically a mutation has a much smaller effect and often it does not even change the fitness value.

3.2 Decoding Procedure

Schedules are produced in a flexible way by introducing a tunable parameter $\delta \in [0.0, 1.0]$. The setting of δ can be thought of as defining a bound on the span of time a machine is allowed to remain idle. At the extremes $\delta = 0.0$ produces non-delay schedules while $\delta = 1.0$ produces active schedules.

In each step of the decoding procedure the set A contains all schedulable operations, i.e. those operations whose job-predecessors have been scheduled already.

1. Determine an operation o' from A with the earliest possible completion time $c' = t' + p'$ with $c' \leq t_{ik} + p_{ik}$ for all $o_{ik} \in A$.
2. Determine the machine M' of o' and build the set B consisting of all operations in A which are processed on M', $B := \{o_{ik} \in A \mid \mu_{i(k)} = M'\}$.
3. Determine an operation o'' from B with the earliest possible starting time, $t'' = t_{ik}$ for all $o_{ik} \in B$.
4. Delete operations in B in accordance to parameter δ such that $B := \{o_{ik} \in B \mid t_{ik} < t'' + \delta(c' - t'')\}$.
5. Select operation o^*_{ik} from B which occurs leftmost in the permutation and delete it from A, $A := A \backslash \{o^*_{ik}\}$.

This tunable decoding procedure produces schedules based on a look-ahead technique. All operations which could start within the maximal allowed idle time of a machine have equal chance for being performed next by that machine. This degree of freedom is used by the GA representation. Whenever there is more than one operation available for processing, the one is selected which occurs at the left-most position of the permutation. In this way the permutation memorizes priorities among operations.

3.3 Genetic Algorithm Parameters

The components described above are now embedded into the framework of a simple GA. Since we minimize $C_{\max}$, selection is based on inverse proportional fitness. The PPX operator is applied with probability 0.8, whereas the mutation operator is applied with the probability of 0.05. A fixed population size of 150 individuals is used.

After a permutation has been decoded, the permutation is rewritten with the sequence of operations as they have been actually scheduled in the decoding step. In order to take advantage from a fast convergence, a flexible termination criterion is used. The algorithm terminates after a T generations which are carried out without gaining any further improvement. We confine T to half of the number of operations contained in a problem instance. In this way larger instances are given a longer time to converge.

The appropriate setting of the parameter δ remains as an open issue. Storer et. al (1992) propose values of 0.0 and 0.1 in the context of another search method. In a recent publication (Bierwirth and Mattfeld, 1999) we have found that on average $\delta = 0.5$ works best for a set of instances of a related scheduling problem. Furthermore we have found that different optimal parameter settings $\delta^\star$ exist for different problem instances. The spread of $\delta^\star$ as well as the improvements we can expect from tuning δ are subject of the following investigation for the JSP.

Table 1: Mean relative error observed for different δ parameterizations.

problem instance	value of parameter δ 0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
la02	5.1	4.7	5.0	4.0	4.2	3.6	4.0	3.8	4.3	4.7	5.0
la04	3.6	3.6	3.6	3.6	3.6	3.6	3.6	1.3	1.6	1.7	2.0
la16	5.3	5.6	5.3	5.1	5.1	4.5	4.3	4.4	4.3	4.1	4.1
la17	1.3	1.2	1.8	1.5	1.7	1.5	1.5	1.6	1.6	2.1	1.9
la19	4.1	3.8	3.6	3.3	2.9	2.8	2.6	2.4	2.2	2.2	2.7
la20	4.2	2.5	2.1	2.1	2.2	2.0	1.9	2.4	2.8	2.6	3.4
la21	8.6	8.1	8.3	8.0	7.8	8.6	8.8	8.9	9.4	9.8	10.6
la27	8.2	8.0	8.0	7.7	7.8	8.4	8.9	9.2	9.1	9.8	10.4
la30	2.2	2.1	1.4	1.5	1.9	1.9	1.9	2.3	2.5	3.1	3.4

4 Scaling the Scope of Search

This investigation is performed on a set of benchmarks proposed in 1984. It consists of 40 instances ranging from 50 to 300 operations. Since meanwhile all instances have been solved to optimality, we are able to report the relative error in % calculated by 100(fitness − optimum)/ optimum in order to obtain a comparable measure of performance. For each of the 40 instances and 11 values of $\delta = \{0.0, 0.1, \ldots, 1.0\}$ we calculate the mean relative error. In order to produce a sound mean we have performed 50 GA runs for each of the resulting 440 cases.

4.1 Gain of Performance

We confine the following discussion to the nine instances listed in Tab. 1. These problems are selected because the instances show a different $\delta^\star$ in the range of $[0.1, 0.9]$ (depicted in gray shade). For non of the 40 instances an optimal parameter setting of $\delta^\star = 0.0$ or $\delta^\star = 1.0$ has been observed. In other words, tuning the decoding procedure has improved the solution quality in every case. Moreover, for certain instances (e.g. 'la20') non-delay and active schedules show the worst mean performance of all δ values investigated. A reduction of the relative error of merely 1% is already significant for the JSP. Hence, we can state that tuning δ to appropriate values is worthwhile the effort spent.

By looking closer at Tab. 1 we notice that —although we must admit a few exceptions— the performance increases with increasing delta starting from $\delta = 0.0$ up to $\delta^\star$. By increasing δ further on we then observe a continuous loss of performance up to the extreme of $\delta = 1.0$. This phenomenon can be nicely observed at the example of instance 'la21': Starting from a mean relative error of 8.6% for non-delay scheduling an increase of δ causes a widening of the look-ahead interval considered in the decoding procedure. A large look-ahead interval in turn leads to a wider scope of search. Schedules of potentially better performance are included in the scope of search and therefore the schedule quality achieved increases (for 'la21' up to 7.8% at $\delta = 0.4$).

4.2 Loss of Performance

By increasing δ beyond $\delta^\star$ for 'la21' we observe a continuous deterioration of GA performance towards 10.6% for $\delta = 1.0$. This finding strikes because non-delay schedules form a true subset of the active schedules. Thus, by scaling up the scope of the search, it is guaranteed that no schedule is excluded from the search. Moreover, a larger search space may include schedules of even better quality. Whenever a decrease of performance is registered in connection with an increase of the scope of search, the algorithm obviously has failed to explore the larger space. In this way $\delta^\star$ determines the borderline of effective genetic search for the particular problem instance under consideration.

Now recall that a flexible termination criterion is used and that the set of active schedules is much larger than the set of non-delay schedules. We expect much shorter GA runs —in terms of generations performed— for the latter case. Indeed, we have observed that the number of generations performed depends almost linearly on the δ parameter used. On average, a GA run with $\delta = 0.0$ takes only 1/2 of the generations needed for a run with $\delta = 1.0$. By taking into account that a GA parameterized with $\delta < 1.0$ improves its performance significantly while saving up to one half of the generations needed, an automatic tuning of δ seems highly desirable.

5 Auto-Adaptive Scaling

As we have seen in Tab. 1, no parameterization exists which fits the needs of all problem instances. Typically the parameterization of GAs is done a priori with respect to the problem difficulty. Since the tractability of JSP instances cannot be determined sufficiently on behalf of the static problem data, we are going to parameterize the GA dynamically over the course of its run.

5.1 Co-Evolutionary Approach

Therefore the GA already described above can be modified in the following way: Instead of setting a fixed δ for each call of the decoding procedure, δ is drawn from a population of co-evolving individuals. This additional population consists of 150 Gray-coded strings in 6-bit precision. After the strings have been initialized at random, they undergo an identical selection and recombination cycle —with the same operator probabilities— as described above for the schedule permutations. Different to those, a standard two-point crossover and a bit-flip mutation operator are used. Whenever a permutation is decoded, the corresponding δ is determined from a string by normalizing its fitness to $[0.0, 1.0]$. The objective function value of the resulting schedule is assigned to both, the permutation and the δ-string. Two variants are proposed.

coupled Every δ string is tied to a certain permutation over the entire runtime of the algorithm. Therefore both, the permutation and the string always carry the same fitness and consequently they are selected as a union in order to enter the population of the next generation. Whenever a couple is chosen for recombination, the appropriate operators are applied to both, the permutation as well as the string.

decouple In this variant both populations evolve almost independently of each other. Selection and recombination cycles are carried out separately for both populations. For the purpose of decoding one member of each of the two populations are combined to generate a new schedule. After the schedule is generated, the fitness is assigned to both, the permutation and the string.

The 'coupled' variant has the advantage, that above average fit couples have a reasonable chance to enter the next population. Moreover elitist strategies can be implemented straightforward. The 'decouple' variant allows to set operator probabilities and fitness-scalings for the two populations independently of each other. Although many different parameterizations of the two variants were tested, the search did not benefit significantly from these modifications. Therefore we report the results of the genuine version of both variants in the following. Again the mean relative error is based on 50 runs for each instance.

Table 2: Performance with auto-adaptive δ tuning.

instance	$\delta^\star$	best	decouple	coupled
la02	0.5	3.6	4.3	4.4
la04	0.7	1.3	3.5	3.4
la16	0.9	4.1	5.4	5.0
la17	0.1	1.2	1.3	1.3
la19	0.8	2.2	3.7	3.6
la20	0.6	1.9	1.8	1.7
la21	0.4	7.8	8.0	7.9
la27	0.3	7.7	7.9	8.0
la30	0.2	1.4	2.0	2.2

The third column 'best' of Tab. 2 reports the best performance taken from Tab. 1, while the second column lists the $\delta^\star$, at which the best performance was observed. The results obtained for both co-evolutionary variants are of almost identical, but inferior quality. Merely for the instances 'la17', 'la21', 'la27' and 'la30' an acceptable performance has been achieved. Consider that all four instances show a $\delta^\star \leq 0.4$. In order to explain the results, program traces of the mean $\overline{\delta}$ of the string population have been taken for all nine instances. In all cases $\overline{\delta}$ rapidly decrease from initial 0.5 to $\overline{\delta} < 0.25$ in later generations. Thus, the acceptable performance observed for four out of the nine instances has been achieved merely by accident. A proper auto-adaptation towards $\delta^\star$ has not taken place.

5.2 Deceived by Stochastic Sampling

The failure of the auto-adaptive tuning of δ can be explained by arguments of plausibility. Recall that the size of the set of active schedules is much larger while the average performance is inferior compared to the properties of the set of non-delay schedules. Therefore the decoding procedure returns a much better performance in the vast majority of cases if called with a small δ. As a consequence, selection tends to favor non-delay scheduling because of its seemingly better potentials. Auto-adaptation of δ fails because the stochastic sampling of the GA is deceived.

The set of active schedules contains at least one schedule of optimal performance. Thus, a relatively large δ is a prerequisite for the generation of schedules of near-optimal or even optimal quality. The generation of one of these scarce schedules is subject to search. Hence an appropriate decoding procedure has to carry out a search process on its own in order to assess the potentials of a certain δ. A GA which is able to tune δ properly requires another GA (called GA$'$) inside its decoding procedure. This GA$'$ optimizes random permutations by using a fixed δ, i.e. a δ which is prescribed by the decoding procedure of GA. The only task of GA$'$ is to assess the potentials of the δ value.

In detail, the decoding function of the GA takes two inputs, a permutation and a δ-string. It builds the schedule and assigns the resulting $C_{\max}$ as fitness to the permutation. In order to assign a proper fitness to the δ-string, GA$'$ is called. After the GA$'$ run finishes, the decoding procedure scales the $C_{\max}$ according to the GA$'$ assessment, before it is assigned as fitness to the δ-string. Unfortunately, at the current stage of computer technology this two-stage optimization procedure seems computational prohibitive.

In the sequel we suggest a simplified two stage procedure. First, we consider the task of GA$'$ as much more difficult to perform than the actual search among the δ values — provided that GA$'$ reports δ assessments correctly. Therefore the GA may be replaced by a hill-climber starting e.g. from $\delta = 1.0$. Fortunately, we do not have to perform additional experiments in order to obtain results, since we can simulate the search manually by means of Tab. 1. Every figure in the table (actually presenting the mean of 50 runs) may go for an assessment of a certain δ achieved by GA$'$.

Now let us start from the most right column of the table ($\delta = 1.0$) stepping to the left as long as the mean relative error does not increase. The performance obtained from GA is now estimated by simply taking the entry of Tab. 1 at the column where the hill-climb has stopped. By performing this hill-climb $\delta^\star$ would have been found for six of the nine problem instances. The suggested procedure performs much better compared to the co-evolving δ.

6 Conclusion

In this paper we have shown that scaling the scope of search will be useful whenever the search space is large and/or the method of search is weak. To scale the scope of the search auto-adaptively is highly desirable, but difficult to achieve. We must admit that this research has not opened a way to an auto-adaptive scaling of the search space within the GA framework.

The issue actually goes far beyond job shop scheduling, since narrowing the scope of search heuristically is a widely used concept. As in the JSP case, in many search spaces which have been narrowed heuristically, the mean performance observed is superior to the mean performance observed for the original space. Also another obstacle considered in this paper can be met for other optimization problems: Narrowing the scope of search often excludes near-optimal or even optimal solutions from being considered.

An appropriate scaling at which heuristic knowledge is incorporated adequately is highly desirable. Adaptation should keep the search space small without excluding near-optimal solutions from the scope of search. A practical way to control the process auto-adaptively is left as an open issue for further research.

References

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

Bierwirth, C., Mattfeld, D.C., and Kopfer, H. (1996). On permutation representations for scheduling problems. In Voigt, H.-M., et al., editors, *Proceedings of Parallel Problem Solving from Nature IV*, pages 310–318, Berlin: Springer.

Bierwirth, C. and Mattfeld, D. C. (1999) Production Scheduling and Rescheduling with Genetic Algorithms. *Evolutionary Computation*, to appear 7(1)

Blanton, J. L. and Wainwright, R. L. (1993). Multiple vehicle routing with time and capacity constraints using genetic algorithms. In Forrest, S., editor, *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 452–459. San Mateo, CA: Morgan Kaufmann.

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

Giffler, B. and Thompson, G. (1960). Algorithms for solving production scheduling problems. *Operations Research*, 8:487–503.

Kolisch, R. (1995) *Project Scheduling under Resource Constraints*. Heidelberg. Physica/Springer.

Mattfeld, D. (1996). *Evolutionary Search and the Job Shop*. Heidelberg. Physica/Springer.

Storer, R., Wu, S., and Vaccari, R. (1992). New search spaces for sequencing problems with application to job shop scheduling. *Management Science*, 38:1495–1509.