Predicting the Outcome of Mutation in Genetic Algorithms

Sandeep Rajoria[1], Carlos Soares[2], Jorge Pinho de Sousa[3], and Joydip Dhar[4]

[1] ABV-Indian Institute of Information Technology and Management, Gwalior, India
[2] LIAAD-INESC Porto LA/Faculdade de Economia, Universidade do Porto, Porto, Portugal
[3] INESC Porto LA/Faculdade de Engenharia, Universidade do Porto, Porto, Portugal
[4] Faculty of Applied Sciences, ABV-Indian Institute of Information Technology and Management, Gwalior, India
sandeep.rajoria@iiitm.ac.in, csoares@fep.up.pt, jsousa@inescporto.pt, jdhar@iiitm.ac.in

Abstract. The general goal of our work is to develop *smart operators* that incorporate Machine Learning methods to improve the search of GAs (or EC, in general). As a first step in that direction, this paper investigates if is it possible to learn to predict the behavior of the swap mutation operator on Job-Shop Scheduling problems. For that purpose, we generate all possible operations and assess their effect on the quality of the solutions. We apply a learning algorithm to obtain a mapping between a description of the operations and that effect. We obtain good results even using relatively small training sets.

1 Introduction

Genetic Algorithms (GA) or, more generally, Evolutionary Computation (EC) methods, are widely used in Optimization [2]. Within the EC framework, the user may tackle an optimization problem using off-the-shelf methods, such as a binary GA. In this case, the user must design methods to translate problem solutions into binary strings (encode) and to translate these back into solutions (decode). Any implementation of a binary GA can then be used to carry out the optimization process. On the other hand, EC also enables a radically different approach, by tailoring part or the whole optimization process to the problem. In this case, the user may work with any kind of representation of the solutions (ranging from simple numerical vectors to very complex structures, such as graphs and trees) but appropriate operators must then be developed (or adapted) to carry out the search.

The flexibility of the EC framework encourages the development of new operators. These can be general operators, that depend only on the representation, or more specific ones, that are only suitable for a small set of problem types. This flexibility creates an appealing opportunity to employ Machine Learning methods [5].

The general goal behind this research is to develop *smart operators* that incorporate Machine Learning methods to improve the search of GAs (or EC, in general) so as to make it more efficient and faster. These operators should be able to decide how to behave based on previous experience. But before developing such an operator it is necessary to investigate the following question: *is it possible to learn to predict the behavior of GA operators?*.

In this paper, we perform one such study. Our goal is to test whether it is possible to predict the outcome of the swap mutation operator of a GA applied to the problem of Job-Shop Scheduling. We systematically generate mutation operations and evaluate their effect on the quality of the solutions. Then, we characterize operations using a set of measures and, finally, we use learning methods to obtain a mapping between the characteristics of operations and their effect.

We start by describing the Job-Shop Scheduling problem and the mutation operation which is investigated in this work (Section 2). Next, we describe our approach in detail, focusing on the process of generating the data to be used for learning (Section 3). In Section 4, we present some experimental results. Finally, we present some conclusions and future work, namely how a *smart operator* can be built based on the learning approach described in this paper (Section 5).

2 Job-Shop Scheduling Problem and Swap Mutation

We start by informally describing the Job-Shop Scheduling problem. More detailed descriptions can be easily found in the literature (e.g., [6]). Then, we describe the swap mutation operator, which is the object of this analysis.

2.1 Job Shop Scheduling problems

In summary, the Job Shop Scheduling (JSS) problem can be defined as follows:

- A finite set of n jobs
- Each job consists of a chain of operations
- A finite set of m machines
- Each machine can handle at most one operation at a time
- Each operation needs to be processed during an uninterrupted period of a given length on each machine

The goal is to find a *schedule*, that is, an allocation of the operations to time intervals on machines, that optimizes some criterion. In our experiments, the evaluation criterion is the *makespan*, i.e., the end time of the job finishing the latest.

Thus from the above definition we can figure that the most basic and simple JSS problem consists of:

Precedence matrix This basically consists of the precedence constraints, representing the order of the machines in which the jobs have to be processed.

Duration matrix The time required for the jobs to be processed in the machines.

Table 1 contains an example of a 3x3 JSS problem, i.e., 3 machines and 3 jobs.

Table 1. An example of a 3x3 problem

	Duration			Precedence		
	Machine1	Machine2	Machine3	Machine1	Machine2	Machine3
Job1	10	24	11	3	2	1
Job2	64	31	95	2	3	1
Job3	9	12	30	1	3	2

2.2 Representation

A chromosome is an unpartitioned permutation with m repetitions of the n jobs [1]. An example of a solution for a 3x3 (3 Machines and 3 Jobs) JSS problem is

1,3,3,2,1,2,3,1,2

which actually will mean that:

– the first operation to be scheduled is the first of job 1 (which, according the precedence matrix, turns out to be on machine 3);
– the second operation to be scheduled is the first of job 3;
– the third operation to be scheduled is the second operation of job 3, as 3 appears for the second time;
– and so on...

A schedule is generated by decoding a solution. We use a simple algorithm which allocates time slots on the machines to operations, strictly following the order in the chromosome. The steps used in the whole process of decoding can be summarised as follows. For $i \in 1 \ldots n \times m$:

1. $j \leftarrow i^{\text{th}}$ element of the chomosome. The value of j represents the job to which the operation that we are scheduling belongs.
2. $o \leftarrow$ number of operations of j that have been already scheduled, i.e., the number of times j is in the chromosome before the i^{th}.
3. $p \leftarrow$ machine in which the o^{th} operation of job j should be executed (from the precedence matrix).
4. $d \leftarrow$ execution time required by the operation of j to be executed oon machine p (from the duration matrix).
5. $s \leftarrow$ max(end time of the latest operation of job j that was already scheduled, end time of the latest scheduled operation on machine p).
6. Schedule the operation of job j on machine p from time s to $s + d$.

This algorithm generates valid schedules, i.e., schedules which do not violate any of the constraints. However, it may generate schedules which unnecessarily delay operations. For instance, let us consider two operations on the same machine, where operation i follows operation j in the chromosome. Using this algorithm, i will be scheduled after j, even if it the corresponding machine has a free slot before operation j which is long enough to schedule i (and assuming that the remaining constraints are not violated). Other algorithms exist that do not have this shortcoming (e.g., generate *active schedules* [3]). However, the choice of decoding algorithm is not essential for the purpose of this work.

2.3 Swap Mutation

Our work is focused on the swap mutation operator, which is commonly used in permutation-based representations [2]. It simply consists of generating a new chromosome by randomly swapping two elements from an existing one. An example of the swap mutation operator is given in Table 2.

Table 2. An example of the swap mutation. The selected positions are 2 and 4

Chromosome	1,3,3,2,1,2,2,1,3
Mutated chromosome	1,**2**,3,**3**,1,2,2,1,3

3 Prediction of the Outcome of Swap Mutation

As stated earlier, the goal of this work is to test the hypothesis of whether it is possible to learn models that are able to predict the behavior of GA operators and, in particular, of swap mutation. Two important characteristics of this problem are:

- the function we are trying to learn is deterministic, unlike most learning problems. The outcome of an operation is always the same.
- the universe of examples is finite and can be systematically generated. The universe of solutions of a $n \times m$ problem is the set of all unpartitioned permutations of n elements with m repetitions. The universe of all operations is obtained by testing all possible pairwise swaps of elements for each solution in the universe.

The size of the universe of operations is, naturally, very large, except for very small n and m. But, in the latter case, it is possible to generate all the possible operations, and, thus, investigate our hypothesis thoroughly, as will be described.

The methodology proposed is the following:

1. Generate all possible operations and estimate their effect on the quality of the corresponding solutions, which is the *target variable* (Section 3.1).

2. Generate a description of each of the operations (i.e., a set of *features*) (Section 3.2).
3. Use learning methods to find a mapping between these features and the target value.

Here the target variable we are dealing with is the effect of the mutation operator on the makespan of the solution. So the function that we are looking for maps the inputs, i.e., the features of the operation (x_i) to the output, i.e., the variation in the makespan of the chromosome ($\Delta makespan$). It can be represented as:

$$\Delta makespan = f\left(x_1, \ldots, x_k\right) \tag{1}$$

where k is the number of attributes. The target variable is numeric, which means that this is a regression problem.

For the development of smart operators, it may be sufficient to know simply if the operation is going to improve the quality of the solution or not. In this case, the problem can be addressed as a classification problem. The class of each operation can be determined simply by determining the sign of the variation in makespan:

$$class = sign\left(\Delta makespan\right) \tag{2}$$

Only if it is possible to generate such a mapping successfully, can we move on to build a smart swap mutation, which can guide its operation based on a model of its past performance.

Experimental results of this approach are presented in Section 4. Before that, we describe in mode detail the process of generating the target values.

3.1 Target Values

Examples are generated as follows:

1. Take one of the instances and generate all the possible chromosomes, i.e., all unpartitioned permutations of n elements with m repetitions. In a 3x3 problem, this amounts to a total of $\frac{(3*3)!}{3!*3!*3!} = 1680$ chromosomes. Then, we compute the fitness of each of the chromosomes.
2. For each one of the chromosomes, find out all the operations that is possible to carry out with the swap mutation. For a 3x3 problem, there are $18 + 9 + 0 = 27$ possible swap mutations for each solution that exchange alleles with different values.
3. Determine the target value for each operation. This is computed as the difference in the fitness of the chromosome that is generated by the operation and the fitness of the original chromosome. In a 3x3 problem, we have a total of $1680 * 27 = 45360$ operations and, thus, of fitness variations. Table 3 illustrates the generation of the target value.

Table 3. An example of the process of generating the target value

	Chromosome	Fitness
Original	1,2,2,3,2,3,1,1,3	208
Mutated	**2**,2,2,3,**1**,3,1,1,3	129
Variation		-79

3.2 Features

To be able to predict the effect of an operation on the quality of the corresponding solution, we need to describe those operations using predictive features. This means that the features must contain information that is useful to determine that effect which we are trying to predict. Additionally, the features can only use information that is available before the operation is executed. Otherwise, there is no need to predict its outcome. The information that is available to be used in the development of the features is the following:

Problem :
- Precedence of the machine in the jobs (i.e., the precedence matrix)
- Execution time of the jobs on the machine (i.e., the duration matrix)

Individual :
- Order of the jobs for scheduling (i.e., the chromosome)
- Fitness of the individual

Operation : In the case of swap mutation, the information is can be used is
- Position of the swaps

Many features can be generated based one of these types of information or combinations. In this study we have generated the following set of features:

- J1 & J2 - Jobs to which the operations being swapped belong.
- D1 & D2 - Execution time of all the operations in the job before the current one, for J1 and J2.
- O1 & O1 - Order of the operation in the job, for J1 and J2.
- M1 & M2 - Machines in which the operation should be executed, for J1 and J2.
- MOJ & MAOJ - The job which is numerically smaller and bigger, respectively.
- NTJAB1 & NTJAB2 - Number of jobs that have occurred before the job, for J1 and J2.
- TMJ1 & TMJ2 - Total execution time of all jobs on the machine used by the operation, for J1 and J2.
- DBP - Difference in the position of the operations on their respective machines.
- TTBMTMJ1 & TTBMTMJ2 - Ratio of the total execution time of all jobs on the machine used by the operation (TMJ1 or TMJ2) and the maximum total execution time of all the machines, for J1 and J2.
- SM - Whether the operations share the same machine before and after swap.

- NM1 & NM2 - New machines in which the operation should be executed after the swap, for J1 and J2.
- SNM - Whether the new machines are same.
- TTOMBTTNMJ1 & TTOMBTTNMJ2 - Ratio of the total execution times of all jobs on the old (before the swap, TMJ1 or TMJ2) and new (after the swap) machine used by the operation, for J1 and J2.
- RJMPJ1 & RJMPJ2 - Ratio of the number of jobs on the machine used by the operation, before and after it, for J1 and J2.
- NSOB1 & NSOB2 - Number of operations on the same machine of the operation in between the swap positions, for J1 and J2.
- DPM - Difference between the positions of the jobs J1 and J2 on the machines of the corresponding positions.
- TPJMJ1 & TPJMJ2 - Total execution time of the jobs executed on the same machine used by the job, that are scheduled before it, for J1 and J2.
- MC - The mean correlation between the execution time of the jobs on the machines, for all the jobs.

4 Experimental Results

Given that we generate all possible operations, we need to work with small problems. We have generated ten problems with three jobs and three machines (3x3). We implemented a problem generation method which is commonly used in Operations Research [6].

The smart operators we plan to develop will generate models based on operations that were previously executed. The number of operations that is available for this purpose is necessarily small when compared to the universe of all possible operations. Although our goal at this stage of the work is simply to investigate whether it is possible to predict the effect of an operation, we should use a relatively small number of examples to be able to obtain more reliable conclusions.

Additionally, we wish to assess how small a dataset we can use and still obtain satisfactory models. Therefore, we did experiments using 20%, 10%, 9%, 8%, 7%, 6%, 5%, 4% and 3% percent of the data for training and the remaining data to estimate the generalization error (i.e., as test set).

The algorithms that we used for the purpose of model generation are given in Table 4. More information about these algorithms can be found in textbooks on Machine Learning and Statistics (e.g., [5, 4]).

Table 4. Learning algorithms

Regression		Classification	
LR	Linear Regression	LD	Linear Discriminant
RT	Regression Tree	DT	Decision Tree
SVM-R	Support Vector Machines	SVM-C	Support Vector Machines
RF-R	Random Forest	RF-C	Random Forest

The results are presented in Table 5 and Figure 1. Several interesting observations can be made. Firstly, these results indicate that the hypothesis underlying this study is true: it is possible to predict the effect of swap mutation. The best result in the regression problem is an NMSE of 0.16 and in the classification problem is an error rate 5%.

Table 5. Results of predicting the effect of swap mutation, varying the amount of training data. The evaluation measure for classification is the error rate and for regression is NMSE

	Regression				Classification			
%	LR	RT	SVM-R	RF-R	LD	DT	SVM-C	RF-C
20	0.59	0.55	0.29	0.16	0.18	0.16	0.10	0.05
10	0.59	0.56	0.33	0.23	0.19	0.16	0.12	0.07
9	0.59	0.55	0.35	0.24	0.19	0.16	0.12	0.08
8	0.59	0.57	0.35	0.26	0.19	0.16	0.12	0.08
7	0.59	0.55	0.36	0.27	0.19	0.16	0.13	0.09
6	0.59	0.56	0.38	0.29	0.19	0.17	0.13	0.09
5	0.59	0.56	0.39	0.31	0.19	0.16	0.13	0.10
4	0.60	0.57	0.41	0.33	0.19	0.16	0.14	0.10
3	0.60	0.57	0.44	0.36	0.19	0.17	0.15	0.11

Although the numbers cannot be compared directly, we observe that the results obtained in the classification problem seem to be much better than the ones obtained in the regression problem. This means that we are better able to predict whether an operation is going to improve the quality of the solution or not, than the value of the variation (i.e., by how much the solution improves or worsens).

Additionally, the best results are obtained with the Random Forest algorithm. The results clearly show that Support Vector Machines are also doing pretty well. However, we note that both of these algorithms are computational more expensive than the other ones. This cost may be relevant in the context of the optimization algorithm, although we have not investigated this issue yet.

The lines in the plot are almost horizontal, which means that the effect of reducing the size of the training set is not very big. However, on closer inspection, we observe that it is stronger on the algorithms that achieve the best results. In fact, for the Random Forests the error doubles both in the classification and the regression problems. However, it is still low (10% and 0.36, respectively). One interesting result is that the performance of the Discriminant Analysis methods and simple Tree-based models are almost not affected by the training set size.

The variance of the results is also quite low, which indicates that the approach is robust (Figure 2).

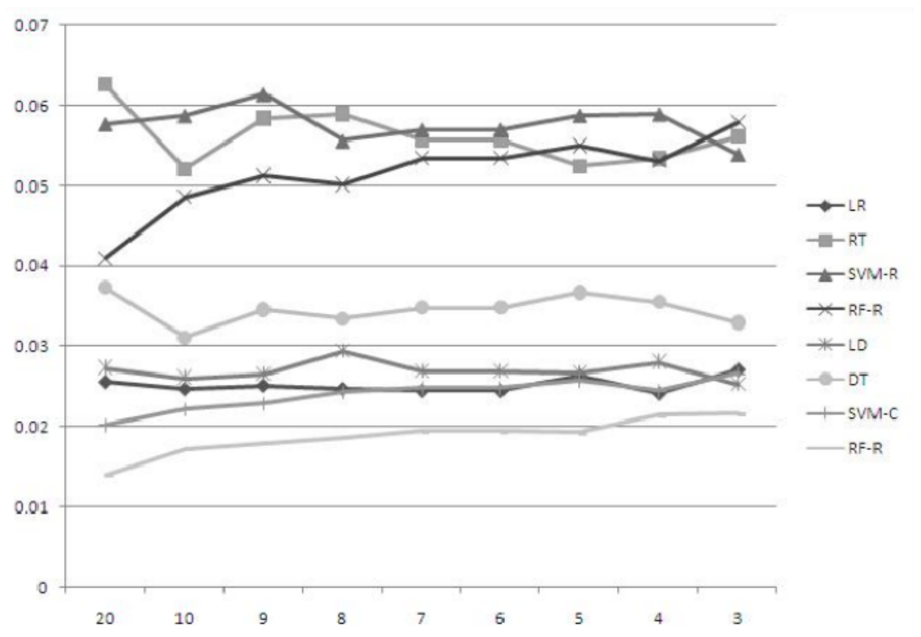

Fig. 1. Results of predicting the effect of swap mutation, varying the amount of training data

5 Conclusions

In this paper, we present a first set of experiments to test whether it is possible to predict the effect of the swap mutation operator in optimization with Genetic Algorithms. The optimization problem addressed is the Job-Shop Scheduling problem.

Our results show that the problem is learnable, even using relatively small training sets. This indicates that it is possible to develop a smart swap mutation operator that decides how to mutate a chromosome based on the effect of previous operations. This smart operator must start with a model based on a small set of operations but it could periodically (say, every 100 operations) update the model. Here we have used larger samples, so it remains to be evaluated the effect of using such small training sets on the quality of the results.

Such a smart operator can be regarded as a greedy operator. Therefore, we note that although it is possible to build it, this does not necessarily mean, when integrated in the search process of a GA, it will contribute to better solutions. Additionally, it must be taken into account that the smart operator described will only be useful if it is not computationally very expensive relative to the optimization process. These issues must be verified empirically. Additionally, it is possible to develop hybrid operators, which sometimes make decisions based on the learning model and sometimes will choose randomly.

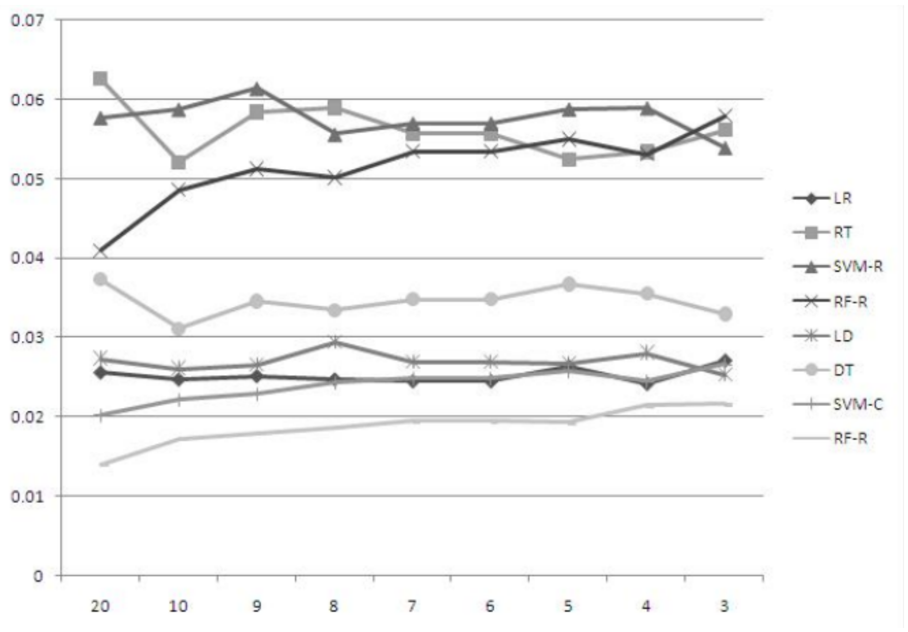

Fig. 2. Variance of the results of predicting the effect of swap mutation, varying the amount of training data

Despite the good results obtained, we believe that it is necessary to make further tests before using this approach to develop smart operators. One important issue is the size of the optimization problems. Our experiments so far were on very small problems. We need to test this approach on larger problems. Naturally, it will not be possible to generate all solutions, so we will work with samples.

Finally, the simplicity of the swap mutation makes it a good candidate to test our hypothesis. It would be interesting to test whether it is possible to use this approach to other operators, in particular to more complex ones, such as crossover. The challenge lies in the design of the features used to describe operations.

References

1. Christian Bierwirth, Dirk C. Mattfeld, and Herbert Kopfer. On permutation representations for scheduling problems. In Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature – PPSN IV*, pages 310–318, Berlin, 1996. Springer.
2. A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing (Natural Computing Series).* Springer, November 2003.
3. B. Giffler and G.L. Thompson. Algorithms for solving production-scheduling problems. *Operations Research*, 8(4):487–503, 1960.

4. Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The Elements of Statistical Learning.* Springer, August 2001.
5. T.M. Mitchell. *Machine Learning.* McGraw-Hill, 1997.
6. E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64:278–285, 1993.