Solving a Job-Shop Scheduling Problem with Genetic Algorithm

Nicolò Merzi, Francesco Battista

I. Abstract

Production scheduling is one of the most important aspects of many companies, as an inefficient solution can directly affect production capacity and, therefore, profitability. When considering applications in problems of medium and large size, finding the optimal scheduling solution is hard. In this paper, we analyze a Genetic Algorithm (GA) for optimizing the scheduling of production systems. The algorithm integrates ad hoc genetic operators to avoid creating unfeasible solutions. Computational results shows that a methodology with approximate solution can achieve satisfactory results, comparable with optimal solutions.

II. Introduction

There are many different Shop Scheduling problems where each job consists in a set of operations, such as *flow-shop*, *job-shop* and *open-shop*. The main difference between the problems is in the processing of operations. In a flow shop problem the machine order in which the job passes through is the same for any job. In a job shop problem, a specific order of operations is given for each job. In an open shop problem, no order is imposed on the jobs. In this paper we deal with the classical formulation of job shop [1].

Job shop scheduling (JSS) is a combinatorial NP-hard problem [2], where the main goal is to find a schedule with minimum makespan for processing n jobs on a set of m machines. Within each job there is a set of operations which need to be processed in a specific order. In the classic formulation of JSS the input data is:

- $\boldsymbol{J}$ set of jobs, $\boldsymbol{J}$ = 1, . . . , n,
- $\boldsymbol{M}$ set of machines, $\boldsymbol{M}$ = 1, . . . , m,
- o_r^i the machine that process the $\boldsymbol{r}$-th operation of job $\boldsymbol{j}$, the sequence without repetition $O^j = (o_1^j, o_2^j, \ldots, o_m^j)$ is the processing order of job $\boldsymbol{j}$,
- p_{ij} processing time of job $\boldsymbol{j}$ in machine $\boldsymbol{i}$.

A JSSP solution must respect the following constraints:

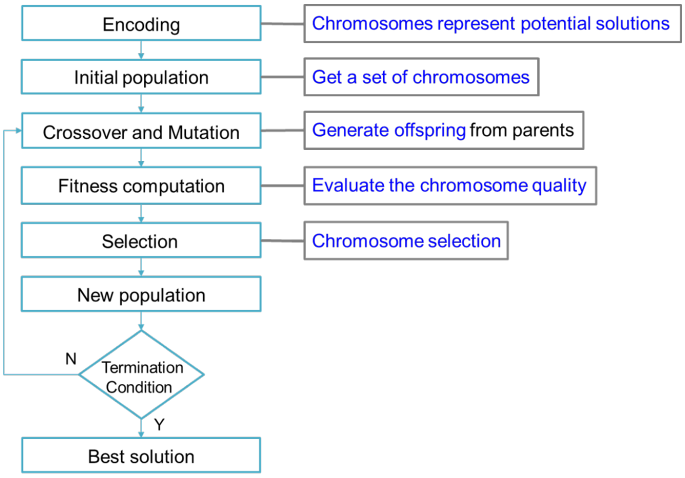
- Only one operation in a job can be processed at a given time,
- Operations of a job must be processed in a certain order,
- Exemption of operations is forbidden,
- An operation cannot be interrupted,
- A machine can process only one job at a time,
- A job consists of at most m operations.

The benchmark used in this paper is the 10x10 instance proposed by Muth and Thompson in 1963 [3] with an optimal solution of 930.

III. Genetic Algorithm

Genetic Algorithms belong to the class of *evolutionary algorithms*. These algorithms draw inspiration from the principles of natural evolution. In nature, only the fittest individuals, namely those who can better adapt to the environment, survive and are able to reproduce. The reproduction of organisms happens through mating and mutation, which generate new offspring and so on. The architecture of a genetic algorithm is based upon these concepts.

An initial population of potential solutions is generated. Each solution is represented by a chromosome, which is an encoded string. Through mating (crossover) and mutation, the next generations is created. Solutions are evaluated against an objective function and only the best individuals are selected to be part of the new generation. This process is repeated until some stopping conditions are reached. The figure below shows the flow chart of GA.



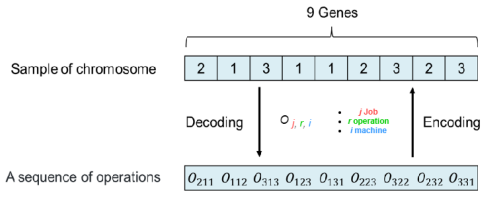
A. GA – Encoding

For Job shop problems, there are a number of possible encoding strategies that can be generally divided in *direct* and *indirect* representation. In a direct representation, a solution is directly encoded into the genotype, while an indirect representation endcodes the instructions to a schedule builder. Independently from the type of representation used, it is necessary, in order to apply GA, to represent the possible solutions in such a way that genetic operators (crossover and mutation) can be applied.

The method used in this paper is a direct encoding proposed by Gen-Tsujimura-Kubota [4]. The concept of this method is to express a solution as a sequence of operations. According to

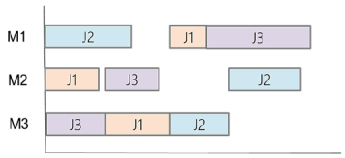
the position and the number of times an operation appears on a chromosome we can encode/decode the resulting schedule. For example, the encoded string $[2, 1, 3, 1, 1, 2, 3, 2, 3]$ of a 3×3 job shop problem represents a possible schedule. The schedule starts with the first operation of job 2, followed by the first operation of job 1 and then the first operation of job 3. After that the second and third operation of job 1 are processed, effectively concluding job 1. We then have the second operation of job 2 and job 3, followed by the third and last operation of again job 2 and job 3. As can be seen, the encoding of a solution is a string where the order represent the sequence of how operations are processed, and the number represents the job. The number of times each job appear on the string represents the ***r***-th operation. There is no need to have information about machines in the encoded string, since each operation of any job can only be processed with a specific machine.

The following image shows the 3x3 example of encoding.



- Processing time

Operation	O1	O2	O3
Job 1	8	10	5
Job 2	15	9	12
Job 3	9	7	19



- Machine sequence

Operation	O1	O2	O3
Job 1	M2	M3	M1
Job 2	M1	M3	M2
Job 3	M3	M2	M1

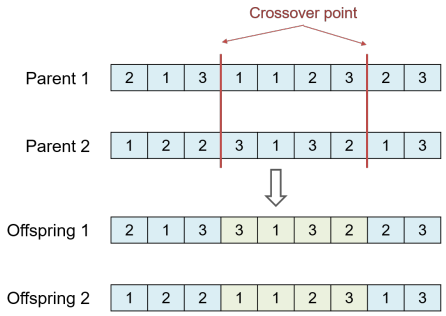
O_{jri} means that ***operation r*** uses the ***machine i-th*** for ***job j***.

The chromosome of each possible solution will be composed by $n \times m$ genes because each job will only be processed once on each machine, so each job will appear exactly ***m times*** in the chromosome. The ***i***-th gene specifies the ***r***-th operation of Job ***J***. In the 10×10 instance, each chromosome is represented by $10 \times 10 = 100$ genes.

B. GA – Crossover

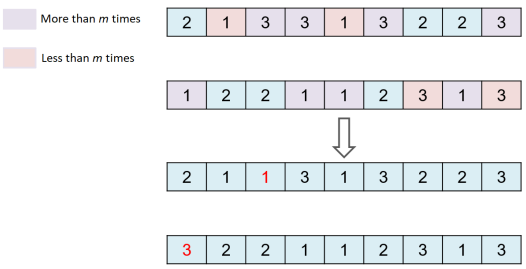
With crossover, the genetic structure of two selected individuals is exchanged too generate new offspring. The parameter *crossover rate* denotes the probability of applying crossover when generating new individuals from the current population. In a scheduling problem, a solution consists of a string of length $n \times m$, where n is the number of jobs and m is the number of machines. If we apply a classical crossover operator on two individuals, the resulting offspring are often unfeasible solutions. Due to the nature of this operator, the number of occurrences of each job in a chromosome can be greater or less than the actual number of operations m. For this reason, a classical crossover operator cannot be used, but instead a specific crossover for permutation problem is needed. The crossover operator can be divided in two main part. The first part is the standard two-point crossover: given two parents and two random cut-points in the chromosome, two children are generated by exchanging the genetic material in between the points of the two parents.

The following image shows an example of the standard two-point crossover:



The second part deals with repairing unfeasible solutions. After the standard crossover operator has been applied and we have a new generation of individuals we need to check and repair the unfeasible solutions. There are different operators that repair illegal solutions[1]. The technique used in this paper checks how many times each job appears in a chromosome and replace the jobs that appear more than *m* times with those that appears less than *m* times, where *m* is the number of operations in a job ***j***. For example, The two new children generated with crossover in the example above are both unfeasible solutions. The repair mechanism checks the occurrences of each job and see that in the first children job 1 appears two times, while job 3 appears four times. In order to convert that solution into a feasible schedule, the repairment procedure replace a 3 with a 1 in the chromosome, so that each job appears exactly *m* times.

The following image is an example of the repairment procedure for the previous crossover operation.

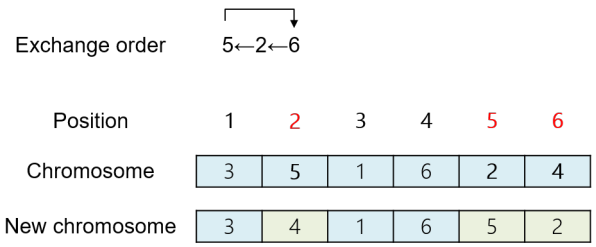


C. GA – Mutation

Mutation is a genetic operator that simulates biological mutation, it allows to maintain genetic diversity between generations, by altering ore or more gene values in a chromosome [5]. Mutation is controlled by two parameters: *mutation rate* and *mutation selection rate*. The mutation rate determines the probability of changing a particular solution, while mutation selection rate determines the percentage of genes in a chromosome to be mutated.

Again due to the nature of the problem, a random mutation strategy can create unfeasible solutions. In order for a solution to be feasible, the number of occurrences of each job in a chromosome has to be constant, for this reason mutation cannot randomly change genes in a chromosome. The method used in this paper is a modified version of *Displacement Mutation*, where two random points are selected and the genes between them are reinserted into the chromosome at a random position [6]. Instead of shifting genes between two random points, a random sequence is generated based on the parameter mutation selection rate, which determines the length of the sequence. This sequence indicates the corresponding position of the genes to be shifted. In this way, only the order of operations changes.

The method is illustrated in the figure below:



D. GA – Fitness

The fitness function is a function which takes a solution as input and returns as output a measure of how "good" the solution is with respect to the problem in consideration. Calculation of fitness value is done for every individual in the population and at each generation. The fitness function also represents the function that the algorithm is trying to optimize (*objective function*).

In this formulation of the problem, the objective of the algorithm is to produce a schedule with minimum makespan (completion time of every job). For each solution, the processing time of every operation is summed to obtain the total competition time for that schedule. This number determines the quality of the generated solution: the lower the makespan the higher the probability that the solution will survive in the next generation.

The fitness *FIT(i)* of an individual *i* is defined as:

$$FIT(i) = M_i(S_i)$$

where $M_i(S_i)$ denotes the makespan value of the schedule S_i resulting from the individual i

E. GA – Selection

Selection happens in two places:

1) Selecting parents from the current generation (***Parent Selection***),
2) Selection from parents + offspring to go into the next generation (***Survivor Selection or Replacement***).

Parent Selection is the process of choosing individuals to be parents for the next generation, based on their fitness level. In order to select parents, a fitness proportionate selection method called *Roulette Wheel Selection* has been used: individuals can become parents with a probability proportionate to their fitness. The fitness of each solution is used to associate a probability of selection proportionate to its fitness level. This means that solutions with a better fitness (lower makespan) will have a greater chance of being selected to be parents of the new generation. This allows to evolve better individuals over time. The probability $P(i)$ of selecting the i-th individual is given by:

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

where p is the population size.

Survivor Selection or Replacement is the process of selecting and discarding individuals of the current generation to be part of the next generation. The parameter *P Population Size* determines the size of the population, which remain constant between generation. After the crossover operator has been applied we are left with a population consisting of parents + children, which is exactly double the population size. Each solution (parent + children) is evaluated and only p solutions are kept to be part of the new generation, the rest are discarded. For example, if we start with a population size of $p = 10$ individuals, after the crossover operator has been applied we now have 20 individuals (10 parents + 10 children). From this 20 solutions we only keep p best individuals to be part of the new generation.

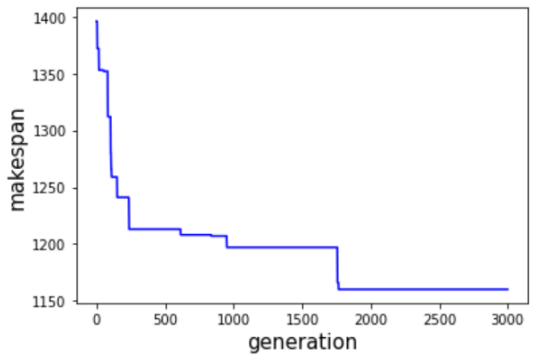
IV. Results

In order to assess the optimal values for parameters, GA has been cross validated ten times with different parameters values. The best solution was found using a population of 60 individuals with a crossover probability of 0.8, a mutation rate and mutation selection rate of 0.2, for 3.000 generations. The algorithm took 80 seconds to run and the best solution found had a makespan of 1155 (930 being the optimal solution).

From the table below, we can see that mutation-only and crossover-only strategies had the worse average competition time, suggesting that a combination of both operators is the best strategy for this kind of problem.

Population	Crossover rate	Mutation rate	Mutation selection rate	Generations	Average competition time	Best time
30	0.8	0.2	0.2	1000	1190	1176
30	0	1	0.5	1000	1222	1197
30	1	0	0	1000	1211	1183
60	0.8	0.2	0.2	3000	1162	1183
200	0.8	0.2	0.2	500	1182	1183

The images below show the makespan of the best solution at each generation and the resulting Gantt chart of final best solution.



V. Conclusion

A Genetic Algorithm was applied to a Job Shop Scheduling Problem. A particular encoding, suited for permutation problems, was used to represent possible schedules. Ad hoc genetic operators were used in order to create feasible solutions for this kind of problem. The result is a Gantt chart representing the best solution found by the algorithm on the 10×10 instance of Muth and Thompson. The computational results shows that, even though GA is a heuristic methodology, it can find good solutions that are close to the optimal.

For further research, the methodology used in this paper could be applied to different instances or with a greater population sized and number of generations. Another interesting possibility could be to generate the initial population following some strategy, instead of generating random schedules as the initial population, resulting in a faster convergence time and possibly better solutions.

VI. Contribution

The whole work is the result of the cooperation between the two authors.

References

[1] F. Werner, *Genetic Algorithm for Shop Scheduling Problems: A Survey*. Otto-von-Guericke-Universität, Fakultät für Mathematik, 39106 Magdeburg, Germany, 2018.

[2] R. Graham, *Bounds for certain multiprocessing anomalies*. Bell System technical journal, 1966.

[3] J.F. Muth and G.L. Thompson, *Industrial Scheduling*, Prentice-Hall, 1963.

[4] M. Gen, Y. Tsujimura, E. Kubota, *Solving job-shop scheduling problems by genetic algorithm*. IEEE International conference on SMC, 1994.

[5] M. Obitko, *Crossover and Mutation*. http://www.obitko.com, 2011.

[6] P. Larranaga, C. Kuijpers, R.H. Murga, S. Dizdarevic, *Genetic Algorithms for the Travelling Salesman Problem: A review of Representations and Operators*. Artificial Intelligence Review, 1999.