Job Scheduling problem using genetic algorithms

Patil Avdhoot Ravindra

Computer science and engineer student

28th October 2020

I. Introduction:

Production scheduling is one of the most important problems for large industries of our time since an inefficient solution can directly affect the production capacity of companies and, therefore, their profitability. With this, the methodologies for solving the problem have evolved over time.

From optimization, the issue can be approached from the search for exact or approximate solutions. However, since production scheduling is an NP-hard problem is advisable to propose methodologies with solutions approximate, how the heuristics and metaheuristics, since these have applications in medium and large problems.

One of the most used methodologies for this type of problem is the genetic algorithm (GA).

What the genetic algorithm seeks is to simulate the growth of a population of living beings through genetics with its different factors: an initial population, a mechanism for mating, mutations and the survival of the stronger individuals to pass from one generation to the next. Then, during the process, an initial population of solutions is generated that, over time, combine with each other to generate new solutions (mating), are modified according to a probability and a mutation function, and the best individuals are selected to survive until the next generation.

For this reason, the following work presents the implementation of a genetic algorithm to a production programming problem type Job Shop where they have M number of machines and J number of jobs, which are made up of n i operations that must be done in a predetermined order and where each operation OR ij can only be performed on a subset of machines M ij.

Within the implementation, a strategy is presented to select the different parameters (size of initial population, number of generations and mutation probability) and then the performance of the algorithm is validated with different

Problems generate randomly.

II. Algorithm design:

For the design of the algorithm, the structure of the individual must be established, the function to create a random individual, the function fitness (which is responsible for measuring the affinity of an individual against the entire population), the function crossover

and the mutation function.
Structure of the individual

The best way to see a production schedule is to have a schedule per operation where the start time of the operation is recorded S ij and the machine X ij
(Bierwirth and Mattfeld, 1999). Therefore, having n number of jobs and m number of machines, and to make implementation easier, he
individual I know
represent with a list of 2 * ∑ • • elements (sum of all operations for each job) where, for each operation there is an integer variable S ij Y
an integer variable X ij, which moves between 1 and m:
$I = [S11, X11, S12, X12, \dots , Snni , Xij]$
With this, having 2 jobs, each with two operations (that is, 4 operations in total) and 2 machines, an individual [0,1,2,1,4,1,6,2] means operation 1 of job 1 starts at time 0 on machine 1, operation 2 of job 1 starts at time 2 on machine 1, operation 1 of job 2 starts at time 4 on machine 1 and job 2 operation 2 are started at time 6 on the machine.

- Function to create random individual:

To have a diverse initial population, feasible and infeasible solutions are generated through three methodologies: 1. Feasible
two. from the front, 2. Feasible from behind and 3. Random (unfeasible).
Feasible from ahead: a solution is generated from the first operation in the dataset (operation 1 job 1) to the last, starting from time zero. To select the machine for each operation, a random number is generated among the eligible machines M ij

Doable from behind: It is the same as the previous one, only it does not start from the first operation but from the last one.
Random: random start times are generated for the variable S between 0 and the maximum processing time for all operations, and random numbers between 1 and the number of machines are generated for all operations in the dataset.

In this implementation, 40% of the initial population was used with feasible from the front, 30% with feasible from behind, and 30% with random.

- Mutation function

For the mutation, two random numbers are generated between 0 and the length of the individual that represent the indices to exchange their values within the individual. If the two numbers are even After several experiments with a 4x2 problem (4 total operations and two machines), one 6x2, one 7x3, one 9x5 and one 11x6, an initial size of 200 individuals and 500 generations is determined, with which a feasible solution is found in all cases. However, when a workable solution is not found at the end of 500 generations, the generations are increased by 100 and the algorithm is rerun. This process is capped at 10 generations increments.
On the other hand, for the probability

of mutation, a value of 30% was taken, taking into account that the initial population has a large proportion of infeasible individuals and that the mutation will help these individuals improve their performance. fitness of others.

- Genetic algorithm:

Genetic algorithm (GA) is the one of the most popular evolutionary or metaheuristic algorithms. The main idea of algorithm is in a population of individuals or chromosomes, where each individual represents candidate solution. In case of scheduling, each individual or solution represents a schedule. Evolution process of GA consists of a number of evolution steps – generations. At each generation of algorithm, individuals inside the population are updated by using several mechanisms: mutation (random change inside one individual); crossover (generation of child solution from two parent solutions); selection (the extinction of the least adapted individuals). Fitness function is used to evaluate individuals and determine the best and worst solutions.

In our proposed GA, we use chromosomes, which are encoded as a set of tasks, allocated on specified nodes $\{(aaii, nnjj)\}$. The example of a topology, computing environment and its possible chromosome are shown on the figure 1.

In this example, streaming topology consist of 3 processing levels or bolts (spout, processing, output) with 3 executors at each level. Computing environment include 2 resources with allocated on them 4 slots in summary.

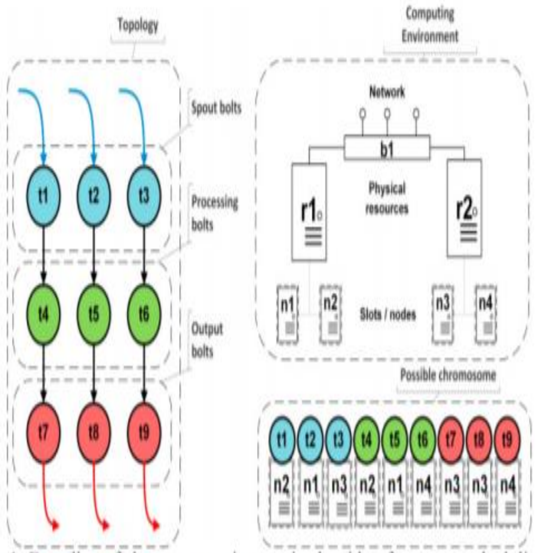

Figure 1: Encoding of chromosomes in genetic algorithm for stream scheduling problem

This coding allows to build schedule for streaming topology. Mutation operator in the developed GA is a random change of one of executor's node in chromosome or swap of nodes for two randomly taken executors. During the crossover operator, child individual receives nodes for a task from two parents randomly. We used tournament mechanism as a selection operator. However, the main part of GA is a fitness function. The aim of fitness function is to build the schedule from chromosome and evaluate the performance of obtained schedule. For this purpose, we require performance models of executors on specified nodes to estimate their throughput. Developed performance models are built on statistical data, which is gathered by developed monitoring system during the run or from history data, if the same applications were already executed. The example of fitness estimation is shown in figure 2.

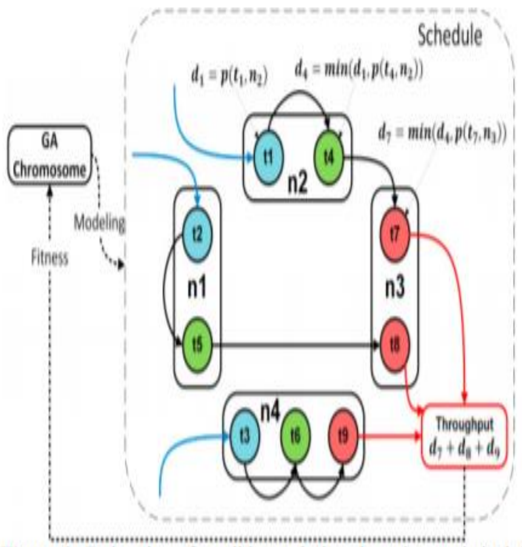

Figure 2: Estimation of candidate solutions by using statistical data

Presented in figure 2 chromosome corresponds to chromosome from previous example. To estimate its fitness function, firstly, the schedule should be constructed. Then, by using performance models $ddii = pp(ttii, nnjj)$ we determine the approximate amount of processed tuples per second. However, performance of executor $ttii$ depends not only on its assigned node, but also on a parent executor. Therefore, the performance of child executors is determined as minimum value between its possible performance and amount of input tuples, taken from parent executor (shown for $tt1$,$tt4$,$tt7$). Therefore, it is crucial to balance performance of all executors to avoid downtimes and floods on particular
executors. The result fitness function is a total throughput of all output executors ($tt7$,$tt8$,$tt9$ in figure 2)

III. Quality of solutions:

By having as the stopping criterion of this algorithm the feasibility of the best individual of the last generation and the number of generations of the input parameter, all the solutions found are feasible and serve as a solution applicable to the input problems For the validation of the algorithm, the same programming of production, problems with which the parameters were established were used and for each of them the number of generations and the execution time, in seconds, were taken. It should be noted that, since it is a random process, different solutions can be produced for the same problem, so the algorithm was run 3 times per problem and the best solution was extracted:

issue	Generations	
500	Time (s)	
4x2		13.66
6x2	500	16.33
7x3	500	15.87
9x5	500	21.23
11x6	500	27.76

It can be seen that, for the best solution, it was not necessary to increase the number of generations and the execution times are below 60 seconds, which, taking into account that the production scheduling problem is tactical, these times are applicable to a real production line.

Comparing it with high-performance hybrid algorithms such as that of Goncalvez, Mendes and Resend (2005), it is found that the developed algorithm has an execution time of 27.76 seconds for an 11x6 problem and the algorithm with which it is compared solved a problem 10x5 in 32 seconds, which, although it cannot be directly compared since they are different problems, does give a good picture of their efficiency in relation to the size of the problem.

IV. Conclusions:

In the present work, a genetic algorithm was implemented to a problem of scheduling and rescheduling with genetic algorithms. specifically, a Flexible Job Shop. During the implementation, an individual structure was used based on the start time of the operation and the machine, a diverse initial population where there are feasible and infeasible individuals, and a function fitness which minimizes the C max (makes pan) and punishes the no compliance of restrictions increasing the value of it. After its implementation in Python, the algorithm was validated with 5 different problems of different dimension operations-machines. This resulted in good performance with solution times of less than 60 seconds for the 11x6 problem. However, the performance of the algorithm should be explored with larger problems and, if possible, compared with implementations known in the literature.

On the other hand, with the completion of this work, it can be concluded that the most critical parts of a process of implementation of a genetic algorithm are the design of the individual's structure and function fitness since, you should always seek that the structure is simple and understandable but that it does not require many calculations for the fitness because otherwise it would affect the execution times. Also, a diverse starting population can bring better results.

GitHub Link: https://github.com/ApLife1827/INT246