# The Use Of Genetic Algorithm In Job Shop Scheduling Problem

Yuanjie Shi

Junjie Chen

1. **Abstract**

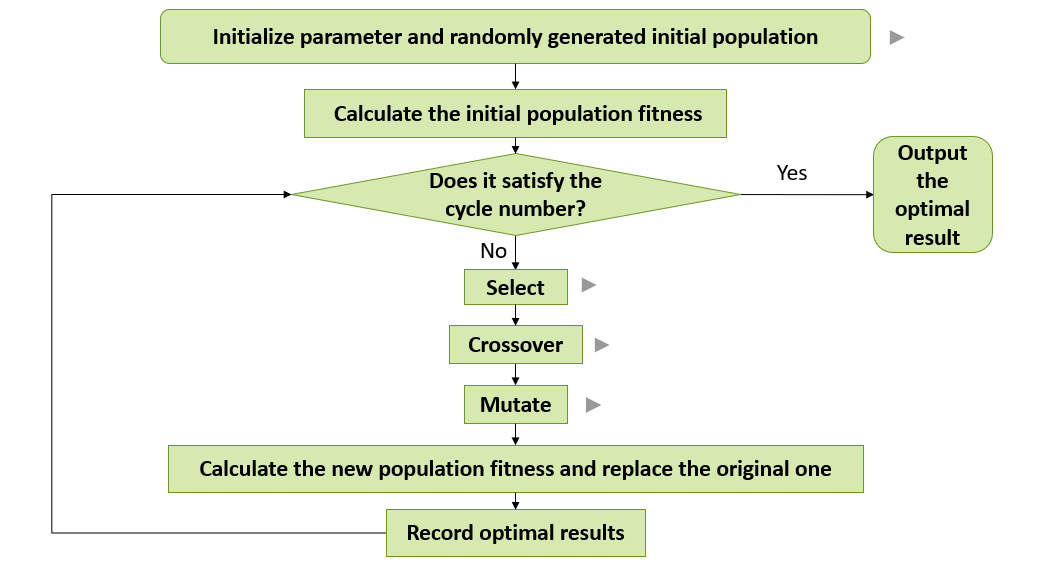
In manufacture, we always face a dilemma: There are several operations O1, O2…that need to be processed in a specific order (known as Precedence constraints). Each operation needs to be processed on a specific machine and only one operation can be processing at one time. How can we arrange the production sequence? Which production sequence is the best? These problems is called the job shop scheduling or the job-shop problem. JSP is an optimization problem in computer science and operations research in which jobs are assigned to resources at particular times.

In this paper, we build a Genetic Algorithm that is designed according to the mixed-model assemble line to solve the JSP problem. The Genetic Algorithm we build includes Population Fitness Function, Select Function, Crossover Function and Mutate Function. We test the model in MATLAB with simulated then compare the change of solution and population mean and the Gantt charts of the optimum results of two selection method to find a more efficient way to solve the JSP problem.

1. **Implementation**

In computer science and operations research, genetic algorithm (GA) is inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions of optimization and search problems by relying on bio-inspired operators such as mutation, crossover and selection.

We build a multiply layer Genetic Algorithm to solve JSP problem. The structure of this Algorithm is shown in the figure below:



There are several modules in this model that implement different functions. The Population Initialization module initializes the initial solution set of population initializing problem. The Fitness Calculation module calculates the initial Population fitness. The selection Module picks outstanding individuals from the test data. The Mutate module gets outstanding individuals. We will discuss this modules specifically at below.

1. **Population Fitness Function**

A fitness function is a particular type of objective function that is used to summarize, as a single figure of merit, how close a given design solution is to achieving the set aims. Fitness functions are used in genetic programming and genetic algorithms to guide simulations towards optimal design solutions.

In particular, in the fields of genetic programming and genetic algorithms, each design solution is commonly represented as a string of numbers (referred to as a chromosome). After each round of testing, or simulation, the idea is to delete the n worst design solutions, and to breed n new ones from the best design solutions. Each design solution, therefore, needs to be awarded a figure of merit, to indicate how close it came to meeting the overall specification, and this is generated by applying the fitness function to the test, or simulation, results obtained from that solution.

1. **Crossover Function**

In genetic algorithms, crossover is a genetic operator used to vary the programming of a chromosome or chromosomes from one generation to the next. It is analogous to reproduction and biological crossover, upon which genetic algorithms are based. Crossover is a process of taking more than one parent solution and producing a child solution from them.

1. **Mutate Function**

Mutation is a genetic operator used to maintain genetic diversity from one generation of a population of genetic algorithm chromosomes to the next. It is analogous to biological mutation. Mutation alters one or more gene values in a chromosome from its initial state. In mutation, the solution may change entirely from the previous solution. Hence GA can come to a better solution by using mutation. Mutation occurs during evolution according to a user-definable mutation probability. This probability should be set low. If it is set too high, the search will turn into a primitive random search.

The classic example of a mutation operator involves a probability that an arbitrary bit in a genetic sequence will be changed from its original state. A common method of implementing the mutation operator involves generating a random variable for each bit in a sequence. This random variable tells whether or not a particular bit will be modified. This mutation procedure, based on the biological point mutation, is called single point mutation. Other types are inversion and floating point mutation. When the gene encoding is restrictive as in permutation problems, mutations are swaps, inversions, and scrambles.

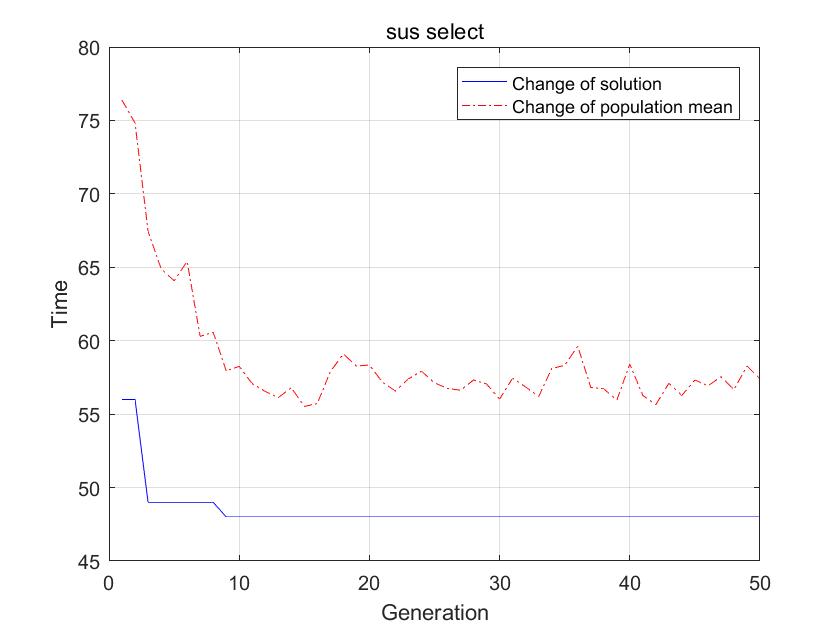
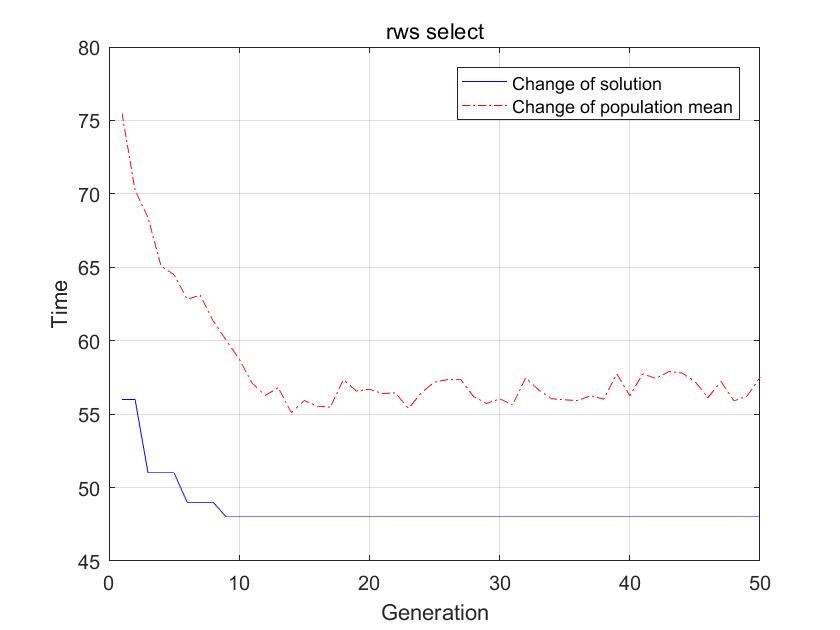
The purpose of mutation in GAs is preserving and introducing diversity. Mutation should allow the algorithm to avoid local minima by preventing the population of chromosomes from becoming too similar to each other, thus slowing or even stopping evolution. This reasoning also explains the fact that most GA systems avoid only taking the fittest of the population in generating the next but rather a random (or semi-random) selection with a weighting toward those that are fitter.

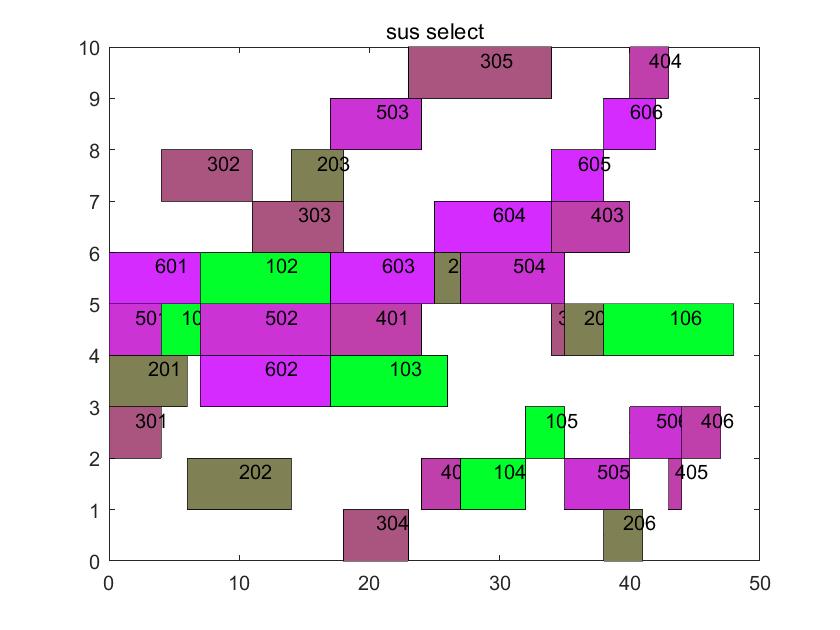
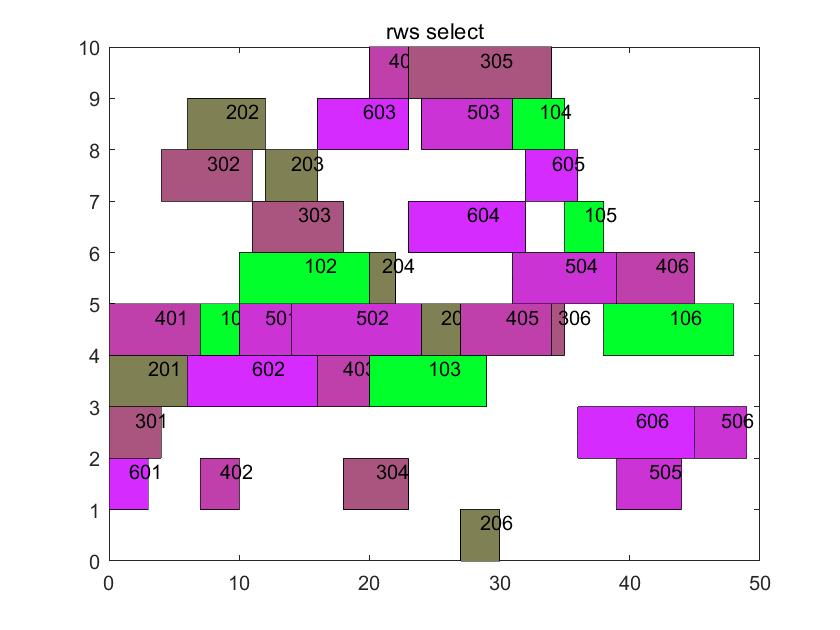
1. **Result analyzing**

The rws is a method selected based on the probability of a chromosome's fitness value. Increasing the probability of dominant chromosomes with high fitness values makes it easier for excellent chromosomes to be selected, reducing the probability of recessive chromosomes with low fitness values, making it easier to be eliminated.

SUS uses a single random value to sample all of the solutions by choosing them at even Intervals. This gives weaker members of the population (according to their fitness) a chance to be chosen and thus reduces the unfair nature of fitness-proportional selection methods.

The figures below shows the result of the change of population mean and the optimum result with generations growing:





According to the figures, the population mean and the optimum result tends to be stable while the number of generations increase. Besides, the fitness value of the optimum result has no significant difference in RWS and SUS. But the optimum result of SUS decrease rapidly in the early stage and then keep steady until several generations past. This means the convergence speed of SUS is too fast that may lead to a dilemma of local optimum. For this reason, RWS is a better option to solve the JSP problem.

1. **Code use direction**

After unzip the codes, the path of codes files should be added in MATLAB. Then, run main.m, the results is showed.

Due to the wide application of GA algorithm, there are many MATLAB-based genetic algorithm toolboxes, such as the GATBX toolbox designed by Sheffield University. To reduce the complexity of programing, we use the functions ranking.m, resins.m, select.m, sus.m and rws.m from GATBX. Others are programmed by ourselves. And the book *Genetic Algorithm Toolbox: for use with MATLAB,* which is edited by Sheffield University for using GATBX, help us understand GA more clearly.

1. **Work division**

In this term project, Yuanjie Shi program the main, the aberranceJm, the across function. Also, she designs the PPT and does the presentation.

Junjie Chen sets mathematical model for JSP problem and program the cal and the plotRec function. He also writes the report.

**6. Reference**

[1] Fox B R, McMahon M B. genetic Operators for Sequencing Problems: Foundation of Genetic Algorithms[M]. Gregory J E. 3th. San Francisco: Morgan Kaufmann Publishers, 1991.

[2] Jog P, Suh J Y, Gucht D V. The Effects of Population Size, Heuristic Crossover, and

Local Improvement on a Genetic Algorithm for the Traveling Salesman Problem [J]. Proceeding of the Third International Conference on Genetic Algorithms, 1989，110-115.

[3] Starkweather T, McDaniel S, Mathias K, Whitley C, Whitley D. A Comparison of Genetic Sequencing Operators [J]. Proceedings of the Fourth International Conference on Genetic Algorithms, 1991: 69-76.

[4] Andrew Chipperfield, Peter Fleming, Hartmut Pohlheim, Carlos Fonseca. Genetic Algorithm Toolbox: for use with MATLAB