# MULTI OBJECTIVE COMBINATORIAL OPTIMIZATION IN SCHEDULING APPLICATIONS

**by**

**Mohammed Bilal Ansari**

Supervisor:  Dr. Herve Kerivin

Master's Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Computer Science

in the
Institute of Computer Science - ISIMA
Clermont Augergne INP

## Abstract

Scheduling jobs on multiple machines in some applications like fabric dying is an NP hard problem where the goal is to find jobs sequence from combinatorial search space to process on each machine such that it optimizes multiple objectives like total time taken to finish the processing of all jobs, minimizing the overall shade difference, and minimizing the overall setup time. There are certain constraints that need to be followed to ensure that each job is processed only on one machine. In this research project, we proposed the approach based on genetic algorithm to perform multi-objective combinatorial optimizations for fabric dying scheduling application.

**Keywords:** Scheduling application, genetic algorithm, combinatorial search problem

# Table of Contents

## List of Figures

## List of Acronyms

| | |
|---|---|
| GA | Genetic Algorithm |
| MILP | Mixed Integer Linear Programming |
| OR-Tools | Google Optimization Tools |
| COP | Combinatorial Optimization Problem |

## Glossary

| | |
|---|---|
| Job sequence | Set of jobs in certain order which need to be schedule on machine. |
| Makespan | Total time required to finish the processing of all jobs |
| Induced Sub-schedule | Induced sub-schedule of a schedule is another schedule, formed from a subset of the jobs of the schedule and jobs are connected in induced sub-schedule if they are connected in schedule. |

# CHAPTER 1. INTRODUCTION

## 1.1 General Introduction

Scheduling is defined as the allocation of resources to jobs and sequencing of jobs in a decision-making process such that it allocates the limited resources to jobs or activities within the limited time frame. Typical resources for production applications are machines, jobs etc. and typical actions are processing, operate etc. The aim of scheduling is to fulfil one or more objectives at the same time [1]. More often, objectives are minimizing make span and maximizing machine utilization. Scheduling has direct impact on the production of the industry.

Researchers from different communities like Operations Research, Artificial Intelligence, and Computer Science have contributed to solve the scheduling problems related to process industry, operating system, cloud computing etc. in several different ways. Most of the scheduling problems are NP hard [2] that means no known solution methods exist that are of polynomial complexity in the problem size, and it is not possible to find the global optimum solution for the given problem. Therefore, it has been challenging for research community to develop the efficient optimization algorithm for process scheduling in production industry. There has been significant contribution in last decades that aim to develop either tailored algorithms for specific problem instances or efficient generic method. Following are the different approach opted by researcher's community to solve the scheduling problems.

Solving the scheduling problems based on heuristics is often considered efficient procedure, we can obtain the solution within acceptable amount of time. But it has disadvantage that we cannot make the judgement about the obtained solution whether it is high quality solution or not. Performing evaluation on solution achieved by heuristic approach is difficult.

Mixed Integer Linear Programming (MILP) either with linear constraints or a linear objective is also widely used to solve the scheduling problems. Google provides OR-tools to solve integer optimization problem using branch-and-bound algorithm. Using MILP to solve the scheduling problem can lead to either optimal solution or provide bound on the optimal solution in case of convex problem. Thus, it is possible to assess the quality of the solution. But in case of non-convex problem, it is not possible to ensure the global optimality of solution also it is hard to achieve the solution is reasonable amount of time.

Genetic algorithms are based on the metaheuristic inspired by the process of natural selection and rely on biologically inspired operators such as mutation, crossover, and selection. They belong to larger class of evolutionary algorithms. They are commonly used to generate high quality solution for optimization problems in reasonable amount of time. They are stochastic in nature that explore different regions of huge search space in relatively short times. Very few publications have been published to address the scheduling problems for process industry based on genetic algorithms. We can also introduce the multi and complex objectives depending on the problem. They are scalable and can be easily run-on distributed clusters in parallel. But evaluating the solution is challenging because cost functions are unbounded, and speed of convergence cannot be predicted.

## 1.2 Problem Statement

Sequencing and scheduling decisions play a vital role in today's industries, whether it is manufacturing, transportation, or other service industries. Effective scheduling and sequencing decisions are necessary for optimal use of resources and labor, reducing costs, maintaining quality etc. During the last decade, there has been immense progress in the area of scheduling and planning in the process industry. An important scheduling problem is one of sequencing jobs with sequence dependent setup times. Setups are sometimes required to ensure machines are ready to run the jobs.

In this research project, we address the problem of fabric dying scheduling with multiple objectives like reduce setup times, minimize make span, and maintaining the shade consistency. There are also some sequence constraints to ensure that certain jobs are not scheduled after certain others, often it is done to reduce defects and effort to setup the machine. For example, white fabrics should not be scheduled right after black fabrics on the same machine because it is very hard to remove all the black dye from the machine., resulting in spots on the white fabric.

Our problem has complex combinatorial search space and non-linear in nature. We decided to use Genetic algorithm to find out the optimal schedule for our problem. Two-dimensional encoding has been used to represent the schedule. We used most popular method ‘Roulette Wheel Selection’ to select the candidate from GA population. Crossover exchanges the subsequence of parents to generate offspring. We proposed two different approaches to perform mutation, one is mutating each job sequence of machine and another one is mutating across the job sequence of machines. In experimental results, we can observer that mutation always enhances the quality of candidate/schedule. But crossover does not guarantee to enhance the quality but it more likely generate better quality offspring.

# CHAPTER 2. PROBLEM FORMULATION

## 2.1 Decision Variables

Consider the finite set of jobs, $\phi$ which need to be scheduled on finite set of parallel, identical, and independent machines, $\pi$ while optimizing the multiple objective functions like minimizing makespan, $c_{max}$ of the schedule (i.e., total time taken to finish the processing of all jobs) over the machines where the size of finite sets $\phi$ and $\pi$ are n and m, respectively.

$$\phi \ = \ \{1, 2, \ldots.. n\} \, ; \quad \pi \ = \ \{1, 2, \ldots.. m\}$$

Let $i_1 \; and \; i_2$ are two jobs belongs to $\phi$ then shade difference and setup time required between them denoted by $s_{i_1,i_2}$ and $\beta_{i_1,i_2}$ respectively.

Let us formulate the decision variables for our scheduling problem

We need to consider one decision variable to store the distribution of finite set of jobs among the parallel, identical, and independent machines. Another variable to store the ordering of jobs sequence on each machine. Both these variables are associated with each other.

i) Decision variable, $\boldsymbol{M} \in \{0,1\}^{m \, x \, n}$ is the binary matrix, each row of it represents the jobs which need to be schedule on the particular machine.
   The row and column of $\boldsymbol{M}$ are represented as follows:

$$M_j' = col_j \, M^T \;\; \forall j \in \pi \, ; \quad M_i = col_i \, M \;\; \forall i \in \phi \, ;$$

$$M_{j,i} \ = \begin{cases} 1, & if \; machine \; j \; processes \; the \; job \; i \\ 0, & Otherwise \end{cases}$$

$$M_{i_1}^T M_{i_2} \ = \begin{cases} 1, & if \; both \; jobs \; i_1 \; and \; i_2 \; are \; processed \; on \; same \; machine \\ 0, & Otherwise \end{cases}$$

ii) Decision variable, $\boldsymbol{J} \epsilon \{0,1\}^{n \, x \, n}$ is the binary matrix which maintains the ordering of jobs.

The row and column of $\boldsymbol{J}$ are represented as follows:

$$J_i' = col_i \, J^T \;\; \forall i \, \epsilon \, \phi \, ; \quad J_i = col_i \, J \;\; \forall i \, \epsilon \, \phi$$

$$J_{i_1,i_2} = \begin{cases} 1, & if \; job \; i_1 \; is \; processed \; immediately \; before \; i_2 \\ 0, & Otherwise \end{cases}$$

## 2.2 Formulation as a Graph

We express our fabric dying scheduling problem based on graph theory where each job in finite set is represented as a node in the graph. Two different types of vectors i.e., machine vector and job vector are used for the notion of graph representation of our scheduling problem. Machine vector representation is an unordered sequence of jobs which need to be schedule on each machine and considered as isolated nodes in the graph or bag of jobs [fig.2.1]. This vector representation does not maintain information about ordering of processing jobs on each machine. We maintain the ordering between the jobs by representing each job using two 'one hot encoding' vector that is called job vector representation [fig.2.2].

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| M1: | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |

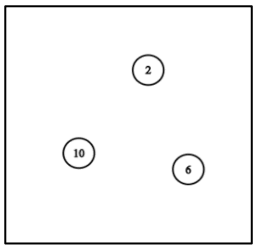


Fig 2.1. Machine Vector Representation

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| J2(In): | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| J2(Out): | *0* | *0* | *0* | *0* | *0* | *0* | *0* | *0* | *0* | *1* |

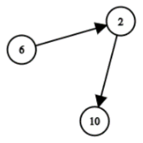


Fig 2.2. Job Vector Representation

## 2.3 Objective Functions:

Our problem is the multi objective combinatorial optimization problem (COP) that has following multiple objective function which we want to optimize.

1) Setup times, $f_{st}$

   Setup times is the time required to setup the job on the machine. Schedule jobs such that it minimizes the overall setup time, $\beta_{\boldsymbol{total}}$.

$$f_{st} : \boldsymbol{M},\boldsymbol{J} \rightarrow \beta_{\boldsymbol{total}}$$

$$\hat{\beta}_{\boldsymbol{total}} = \underset{\boldsymbol{M},\boldsymbol{J}}{min} \; f_{st}(\boldsymbol{M},\boldsymbol{J}) = \underset{\boldsymbol{M},\boldsymbol{J}}{min} \sum_{i_1,i_2 \,\epsilon\, \phi \mathrm{x} \phi} \left( \left( M_{i_1}^T M_{i_2} \right) \wedge \boldsymbol{J}_{i_1,i_2} \right) \beta_{i_1,i_2}$$

2) Shade Consistency, $f_{sc}$

   Difference between the shade value of two job should be consistent. Schedule jobs such that it minimizes the overall shade difference, $\alpha_{\boldsymbol{total}}$.

$$f_{sc} : \boldsymbol{M},\boldsymbol{J} \rightarrow \alpha_{\boldsymbol{total}}$$

$$\underset{\boldsymbol{M},\boldsymbol{J}}{min} \; f_{sc}(\boldsymbol{M},\boldsymbol{J}) = \underset{\boldsymbol{M},\boldsymbol{J}}{min} \left( \sum_{i_1,i_2 \,\epsilon\, \phi \mathrm{x} \phi} \left( \left( M_{i_1}^T M_{i_2} \right) \wedge \boldsymbol{J}_{i_1,i_2} \right) \left| s_{i_1} - s_{i_2} \right| + \sum_{j \,\epsilon\, \pi} \left| \sum_{i \,\epsilon\, \phi} M_{ji} \left(1 - M_j' \, J_i \right) s_i^g - \sum_{i \,\epsilon\, \phi} M_{ji} \left(1 - M_j' J_i' \right) s_i^g \right| \right)$$

   Simplified form:

$$\hat{\alpha}_{\boldsymbol{total}} = \underset{\boldsymbol{M},\boldsymbol{J}}{min} \; f_{sc}(\boldsymbol{M},\boldsymbol{J})$$

$$= \underset{\boldsymbol{M},\boldsymbol{J}}{min} \left( \sum_{i_1,i_2 \,\epsilon\, \phi \mathrm{x} \phi} \left( \left( M_{i_1}^T M_{i_2} \right) \wedge \boldsymbol{J}_{i_1,i_2} \right) \left| s_{i_1} - s_{i_2} \right| + \sum_{j \,\epsilon\, \pi} \sum_{i \,\epsilon\, \phi} M_{ji} \left| M_j' J_i' - M_j' \, J_i \right| s_i^g \right)$$

3) Makespan, $c_{max}$

   Makespan is the total time taken to finish the processing of all jobs. Schedule jobs such that it minimizes makespan, $c_{max}$ of the schedule.

$$minimize \; c_{max}$$

# CHAPTER 3. GENETIC ALGORITHM

Genetic algorithms are efficient global methods to solve the nonlinear optimization problem. GAs are adaptive meta heuristic search algorithms which classified as an evolutionary computing algorithm, which mimic the process of natural selection. GA is a discrete and non-linear process that is not a mathematically guided algorithm where optima evolved from one generation to another without mathematical formulation. They share the favorable characteristics of random Monte Carlo over local optimization methods in that they do not require linearizing assumptions nor the calculation of partial derivatives, are independent of the misfit criterion, and avoid numerical instabilities associated with matrix inversion. The additional advantages over conventional methods such as iterative least squares is that the sampling is global, rather than local, thereby reducing the tendency to become entrapped in local minima and avoiding a dependency on an assumed starting model.

GAs can run on distributed clusters environment to make convergence faster because parallel search to randomly select individual candidate from populations is possible. They perform following operation: crossover on two candidates to exchange information between them, mutate the candidate to perturb information, and selection of candidate until system reaches convergence. Convergence can very problem to problem it can be user defined or until all candidates become identical in the system. Every run of selection, crossover and mutation processes is called generation. Speed of convergence of the system can be estimated by number of generations [3]. The set of candidates in the current generation is called “population”.

GAs are widely used to provide solutions to very complex optimization problems. Researchers have proposed several approaches in many areas such as speech recognition, healthcare, software engineering, cloud computing, image processing, machine learning, computer games, computer networks, Vehicle Routing Problem [6].

Scheduling problems belong to the NP-hard complexity class, using exact methods to solve them will take either enormous amount of computation time or may be result into infinite situation. Due to this high complex nature of problem, providing solutions using heuristics will be helpful in order to address larger solution space in less computational time. GA is one of the efficient metaheuristics applicable to a larger set of combinatorial problems and well suited to solve multi objective optimization problems [11].

Using genetic algorithms to solve a problem involves first defining a representation of chromosomes that describes the problem states. Most previous studies have adopted one-dimensional representation. Some real problems are, however, naturally suitable to two-dimensional representation [8]. But there is relative a smaller number of publications which address GAs implementation based on matrix. Our problem of scheduling has strong reason to use two-dimensional encoding representation of chromosome where each chromosome is represented by two binary matrices.

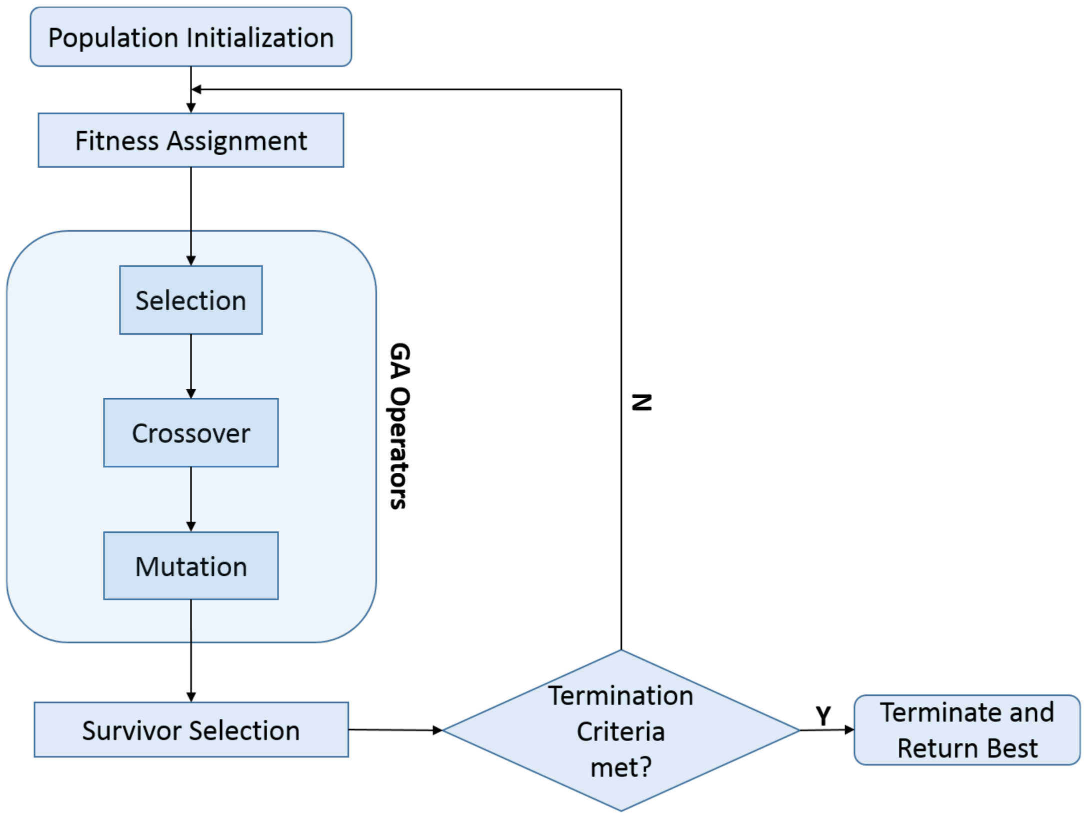


*Fig 3.1. Flow Chart of Genetic Algorithm*

## 3.1 Initial Population

There are two primary methods to initialize a population in GA. One is random initialization (i.e., populating the initial population with completely random solutions) and another one is heuristic initialization (i.e., populate the initial population using known heuristic for the problem). Most of the researchers have used stochastic process to generate initial population of models, in similar fashion to random Monte Carlo simulation [5]. It has been believed that random solutions are the ones to drive the population to optimality.

We decide to use the combination of both approaches random initialization and heuristic initialization. A set of N schedules are constructed as an initial population for Genetic algorithm. 70% of schedules are constructed based on below heuristic approach and 30% are generated stochastically.

**Algorithm 1:** Heuristic Approach to construct Chromosome/Schedule, X

```
Step 1. Initialize binary matrices M, and J for the schedule, X
Step 2. Assign single job to each machine, $j \epsilon \pi$ randomly.
Step 3. Assign remaining jobs to machines based on the job-pair strength. Perform the following steps for each remaining job, k
Step 3.1  Find the job, l from already assigned jobs which has highest pair strength with job, k
Step 3.2  Assign the job, k to the same machine as job, l
Step 3.3  If job, l is already connected to any job, q then, connect job, k to job, q
          $J[k][q] = 1; J[l][q] = 0$
Step 3.4 Finally, connect job, l to job, q
          $J[l][q] = 1$
```

## 3.2 Selection Schema

Selection operation is first operation in every generation. Before performing the crossover, individuals from population in the current generation are selected into a pool. The purpose of this operator is to discard low quality chromosomes and high-quality chromosomes have high chance to survive and propagate their good genes to the next generation. There are several schemas based on either fitness-proportional or ranking-based to perform selection operations. Fitness-proportional selection selects chromosomes according to their probability which depends on the value of fitness itself. Ranking-based selection selects the chromosomes according to their ranking and chromosome of given rank always has fixed probability to be selected no matter how much it is better than others [12].

We decided to use the Roulette Wheel Selection (i.e., fitness-proportional selection) for our system which creates a discrete probability distribution from which we identify the schedule for crossover. It is stochastic approach to perform the selection operation, where the chance of selection of a schedule is proportional to its fitness value.

Consider a population of N schedules with a corresponding fitness value $f_x$ then, corresponding probability $p_x$ of selection is defined as:

$$p_x = \frac{f_x}{\sum_{i=1}^{N} f_i}$$

Higher the probability of schedule greater the chance of selection for a GA population.

## 3.3 Crossover

The purpose of crossover operator in our system is to exchange the subsequence between two parent schedules. Suppose there are K schedules in the population that are represented as $X^k, k \epsilon \{1,2, \ldots, K\}$ where each schedule $X^k$ has n jobs and m sequence of jobs which need to schedule on m machines. Crossover split the job-sequences of pair of selected schedules into two parts and exchange between them to generate offspring. Reason for applying the crossover is that new offspring schedules may be better in quality than both parent schedules, if it considers best subsequence from both parents [4].

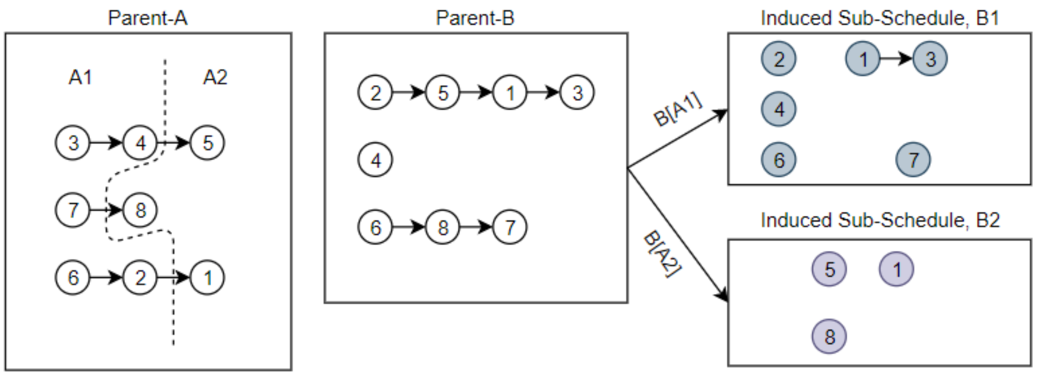


*Fig 3.2 Two parents schedule before Crossover*

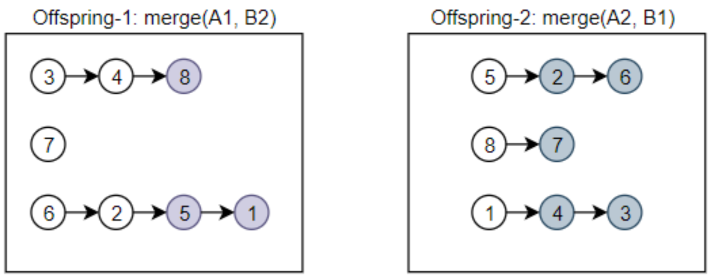


*Fig 3.3 Two offspring schedules after Crossover*

**Algorithm 2:** Crossover on Schedules A and B

*Let $\lambda_{i_1,i_2}$ be the inverse averaged value of setup time and shade difference between $i_1$ and $i_2$, which we called strength of job pair $(i_1, i_2)$.*

*Step 1. Split any one of the schedules into two partitions such that $A^1 \cap A^2 = \emptyset$ where $A^1$ and $A^2$ are the partition of schedule, A. To create the partitions, break weakest job-pair (a, b) of each job sequence of the schedule into two sub-sequence if it satisfies the following criterion:*

$$\lambda_{a,b}^{A_j} < \alpha$$

*Where, (a, b) is the weakest job-pair of job sequence, j of schedule, A and $\alpha$ is the threshold of job-pair weakness.*

*Step 2. Split another schedule, B into two partitions $B^1$ and $B^2$ such that they are the induced sub-schedules of schedule, B as defined below:*

*In case of partition $B^1$, $Jobs(B^1) = Jobs(A^1)$ and $Jobs(B^1) \subset Jobs(B)$*
*Jobs in partition $B^1$ are connected if there are connected in schedule B.*

*Similarly, in case of partition $B^2$, $Jobs(B^2) = Jobs(A^2)$ and $Jobs(B^2) \subset Jobs(B)$*
*Jobs in partition $B^2$ are connected if there are connected in schedule B.*

*Step 3. To generate the offspring from schedule A and B, perform the merge operation as follows:*

*Offspring-1 = $merge(A^1, B^2)$*

*Offspring-2 = $merge(A^2, B^1)$*

*Step 4. Merge operation on any two partitions X and Y is defined as:*

*Perform the below procedure until we process all the components (i.e., sequence of jobs) of sub-schedule, Y.*

```
If number of components in X is less than number of machines, m then
        add the largest component from Y to X
Else, connect the component from Y to the component of X, such that last job of X and first job of Y
has highest pair strength.
```

## 3.3 Mutation

Mutation is a genetic operator used to maintain genetic diversity from one generation of a population of chromosome to the next. It alters one or more gene values in a chromosome from its initial state and avoid local optima [7]. 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.

In our problem, schedule is considered as a chromosome and ordered sequence of jobs as gene. We proposed two different types of mutation for our problem, one is within the job sequence of a machine and another is across the job sequence of machines. One out of two approach is selected randomly to perform the mutation on schedule.

### a) Within the job sequence of machine

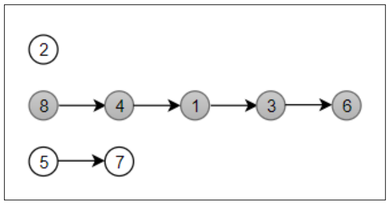
We perform mutation within the job sequence of every machine such that mutated schedule has better quality than before.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |

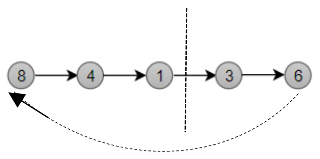
a) Matrix, M

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

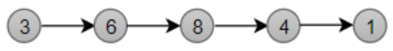
b) Matrix, J



c) Graphical representation of chromosome/schedule



d) Before mutation



e) After mutation

Fig 3.4. Mutation within the scheduling job sequence of machine, $m_2$

**Algorithm 3:** Mutation Within the job sequence of machine

*Step 1. Let* $\lambda_{i_1,i_2}$ *be the inverse averaged value of setup time and shade difference between* $i_1$ *and* $i_2$*, which we called strength of job pair* $(i_1, i_2)$ *and* $\lambda_{total}$ *be the total strength of the schedule.*

$$\lambda_{i_1,i_2} = Inv\left(mean\left(\beta_{i_1,i_2}, \left|s_{i_1} - s_{i_2}\right|\right)\right); \quad \lambda_{total} = \sum_{i_1,i_2 \, \epsilon \, \epsilon \, \phi \mathrm{x} \phi} \boldsymbol{J}_{i_1,i_2} \lambda_{i_1,i_2}$$

*Step 3. Perform the mutation operation on scheduling job sequence of each machine, j in* $\pi$

*Step 2. Compute the size of scheduling job sequence of machine, j*

$$\sigma_j = \sum_{i \, \epsilon \, \phi} M_{ji}$$

*Step 3. If* $\sigma_j$ $>=$ 2 *Then, execute the following steps Otherwise do not perform mutation.*

*Step 4. Find first and last job on machine, j*

$$i_{first} = \underset{i \, \epsilon \, \phi}{argmax} \; M_{ji}(1 - M_j' J_i); \quad i_{last} = \underset{i \, \epsilon \, \phi}{argmax} \; M_{ji}(1 - M_j' J_i')$$

*Step 5. Find the jobs pair* $(i_1, i_2)$ *having lowest value of strength of job pair i.e.,* $\lambda_{i_1,i_2}$

*Step 6. If the* $\lambda_{i_1,i_2}$*, strength of job pair* $(i_1, i_2)$ *weaker than the* $\lambda_{i_{last}, i_{first}}$ *strength of job pair (*$i_{last}$*,* $i_{first}$*) then, update the matrix, J.*

$$J[i_{last}][i_{first}] = 1$$
$$J[i_1][i_2] = 0$$

## b) Across the job sequence of machines

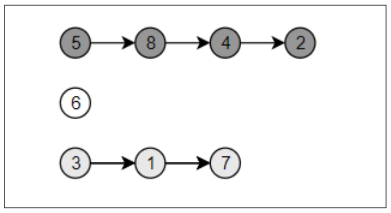
Job sequences of two machines are selected randomly to perform mutation across the machines such that mutation leads to the better-quality schedule than before.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |

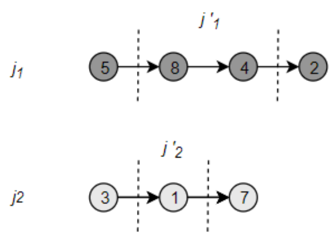
*a) Matrix, M*

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

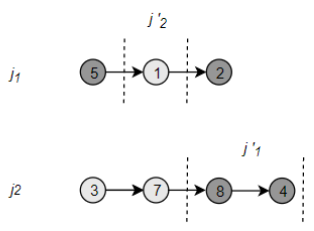
*b) Matrix, J*



*c) Graphical representation of chromosome/schedule*



*d) Before mutation*



*d) After mutation*

*Fig 3.5. Mutation across the scheduling job sequence of machines, m1 and m3*

**Algorithm 4:** Mutation Across the job sequence of machines

*Step 1. Let* $\lambda_{i_1,i_2}$ *be the inverse averaged value of setup time and shade difference between* $i_1$ *and* $i_2$*, which we called strength of job pair* $(i_1, i_2)$ *and* $\lambda_{total}$ *be the total strength of the schedule.*

$$\lambda_{i_1,i_2} = Inv\left(mean\left(\beta_{i_1,i_2}, |s_{i_1} - s_{i_2}|\right)\right); \quad \lambda_{total} = \sum_{i_1,i_2 \,\epsilon\, \epsilon\, \phi \text{x} \phi} \boldsymbol{J}_{i_1,i_2} \lambda_{i_1,i_2}$$

*Step 2. Select two machines randomly* $j_1$ *and* $j_2$ *such that* $j_1, j_2 \,\epsilon\, U(1, m)$ *and* $j_1 \neq j_2$

*Step 3. Compute the size of scheduling job sequence of machines,* $j_1$ *and* $j_2$

$$\sigma_{j_1} = \sum_{i \,\epsilon\, \phi} M_{j_1 i}\ ; \quad \sigma_{j_2} = \sum_{i \,\epsilon\, \phi} M_{j_1 i}$$

*Step 3. If* $\sigma_{j_1} >= 2$ *and* $\sigma_{j_2} >= 2$ *then, execute the following steps Otherwise do not perform mutation.*

*Step 4. Generate two random numbers* $\sigma'_{j_1}$ *and* $\sigma'_{j_2}$ *such that* $\sigma'_{j_1} \,\epsilon\, U(1, \sigma_{j_1})$ *and* $\sigma'_{j_2} \,\epsilon\, U(1, \sigma_{j_2})$

*Step 5. Find the job sub-sequence* $j'_1$ *of length* $\sigma'_{j_1}$ *from sequence* $j_1$ *such that sum of breaking pairs strength is minimum among all the possible sub-sequences. Similarly find the job sub-sequence* $j'_2$ *of length* $\sigma'_2$ *from sequence* $j_2$

*Step 6. Interchange the job sub-sequences* $j'_1$ *and* $j'_2$ *such that total strength of the schedule,* $\lambda_{total}$ *is maximized and update the matrices J and M accordingly.*

# CHAPTER 4. EXPERIMENTAL RESULTS

For this experiment, we considered 20 machines and 80 jobs. Parameters like setup time, $\beta$ and shade difference, s generated randomly from uniform distribution $U(10,50)$. We generated our initial population of size 500 based on our heuristic method.

Two experiments have been performed on our system. One is to understand the behavior of our proposed mutation operator and another one is for crossover operator.

1. **Mutation Operator:** Change in strength of schedule after applying mutation operator.

We applied mutation operators on 60 different schedules to observe its impact on the quality of schedule. In below Fig 4.1 we can clearly observe that curve depicting strength of schedule before mutation is bounded by curve depicting strength of schedule after mutation. Hence, we can conclude that mutation always enhances the quality of schedule and it is deterministic in nature.

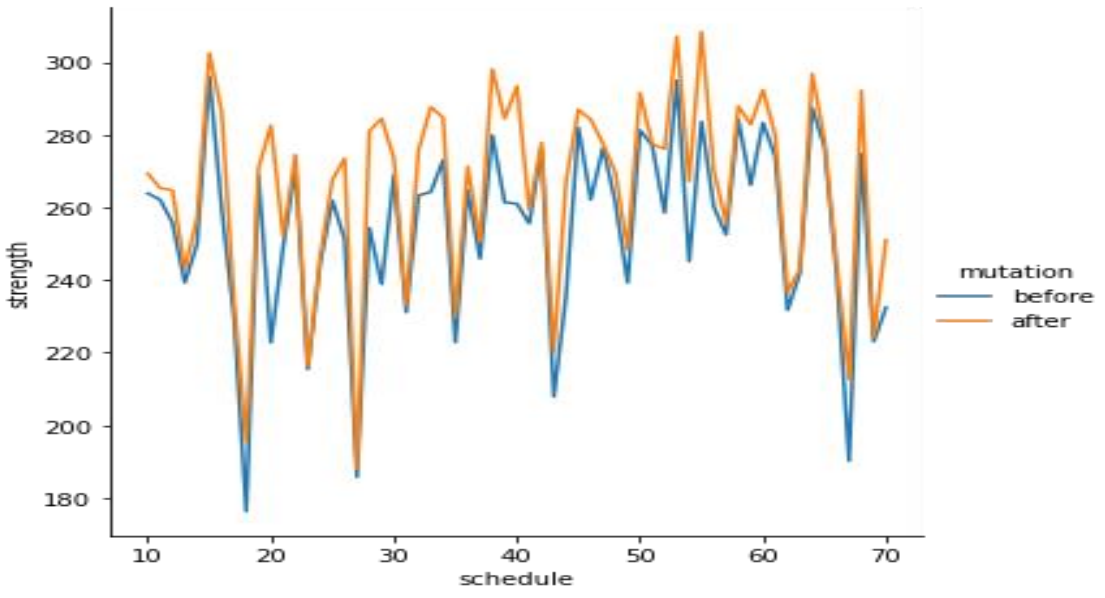


*Fig 4.1. Change in strength of schedule after mutation*

2. **Crossover Operator:** Change in strength of schedule after applying crossover operator

We applied crossover operator on 60 schedule pairs to analysis its behavior. In below Fig.4.2. we can observe that in some cases offspring strength is higher than parent schedules but in some other cases it worse than parent schedules. Hence, we can conclude that behavior of crossover operator is non-deterministic, but it is more likely to generate better quality offspring.

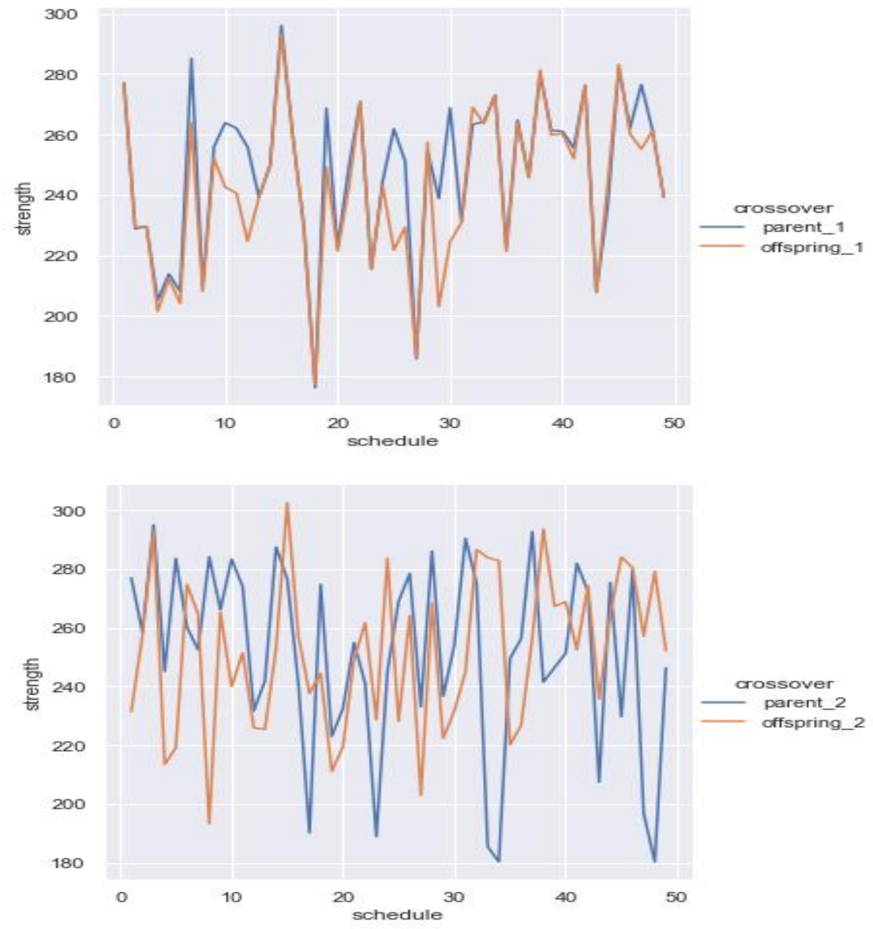


*Fig 4.2. Change in strength of schedules after crossover*

## CONCLUSION

In this research project, we proposed the genetic algorithm-based approach for our multiple-objectives combinatorial search problem (i.e., scheduling problem for fabric dying application). Proposed algorithm used a two-dimensional encoding representation.

In experimental analysis, we observed that our proposed approach for mutation always generates better quality schedule and offspring generated from crossover operation is more likely better than parent schedules.

Further research can be conducted to improve the crossover algorithm which can guarantee that quality of offspring will always be better than parent schedules and considering other objective function like latency etc.

## REFERENCES

[1] Josef Kallrath (August 2002). Planning and scheduling in the process industry. OR Spectrum (Vol. 24, No. 3, pp. 219-250).

[2] Gabow, H.N. (June 1983). On the Design and Analysis of Efficient Algorithms for Deterministic Scheduling. In Proceedings of the 2nd International Conference Foundations of Computer-Aided Process Design, Snowmass, CO, USA (pp. 473–528).

[3] Chen,Y., Elliot M., and Sakshaug, J. (October 2018). Genetic Algorithms in Matrix Representation and Its Application in Synthetic Data. UNECE Work Session on Statistical Data Confidentiality. Ljubljana. https://www.unece.org/fileadmin/DAM/stats/documents/ece/ces/ge.46/2017/2_Genetic_a lgorithms.pdf.Lastaccess20/12/2017.

[4] Hakimi, D., D. Oyewola, Y. Yahaya and G. Bolarin. (November 2016). Comparative analysis of genetic crossover operators in knapsack problem. Journal of Applied Sciences and Environmental Management. (Vol. 20 No. 3 pp. 593-596).

[5] Kerry Gallagher, Malcolm Sambridge (August 1994). Genetic algorithms: A powerful tool for large-scale nonlinear optimization problems. Computers & Geosciences (Vol 20 Issues 7–8 pp. 1229-1236).

[6] V. Kumar S G and R. Panneerselvam (June 2017). A Study of Crossover Operators for Genetic Algorithms to Solve VRP and its Variants and New Sinusoidal Motion Crossover Operator. International Journal of Computational Intelligence Research (Vol. 13 pp. 1717–1733).

[7] Hassanat, A.; Almohammadi, K.; Alkafaween, E.; Abunawas, E.; Hammouri, A.; Prasath, V.B.S. (December 2019). Choosing Mutation and Cross- over Ratios for Genetic Algorithms—A Review with a New Dynamic Approach. Information. (Vol 10 Issue 12 p 390).

[8] Tsai, Ming-Wen, T. Hong and Woo-Tsong Lin. (March 2015). A Two-Dimensional Genetic Algorithm and Its Application to Aircraft Scheduling Problem. Mathematical Problems in Engineering (Vol 2015 pp. 1-12).

[10] Guo, Yifeng, H. Su, Dakai Zhu and H. Aydin. (January 2015). Preference-oriented real-time scheduling and its application in fault-tolerant systems. Journal of Systems Architecture. (Vol 61 Issue 2 pp. 127-139).

[11] Konak, A., Coit, D. W., & Smith, A. E. (September 2006). Multi-objective optimization using genetic algorithms: A tutorial. Reliability Engineering & System Safety. (Vol 91 Issue 9 pp. 992-1007).

[12] A. Thengade and R. Dondal, (January 2012). Genetic Algorithm – Survey Paper. International Journal of Computer Applications (IJCA).