TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechanical Engineering

Development of Evolution Algorithm for Shop Scheduling Problem

Master Thesis

Liberec 2020

Pandiyaraj Gnanasekar

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechanical Engineering

Development of Evolution Algorithm for Shop Scheduling Problem

Master Thesis

Liberec 2020 Pandiyaraj Gnanasekar

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechanical Engineering

Development of Evolution Algorithm for Shop Scheduling Problem

Master Thesis

Study programme:	N2301 Mechanical Engineering
Study branch:	Manufacturing Systems and Processes
Author:	**Pandiyaraj Gnanasekar**
Thesis Supervisors:	Ing. František Koblasa, Ph.D. Department of Manufacturing Systems and Automation

Liberec 2020

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechanical Engineering

Master Thesis Assignment Form

Development of Evolution Algorithm for Shop Scheduling

Name and surname:	**Pandiyaraj Gnanasekar**
Identification number:	S18000448
Study programme:	N2301 Mechanical Engineering
Study branch:	Manufacturing Systems and Processes
Assigning department:	Department of Manufacturing Systems and Automation
Academic year:	**2019/2020**

Rules for Elaboration:

The aim of diploma thesis is to develop evolution algorithm to solve scheduling problems in mechanical engineering manufacturing. Thesis will include:
1/ Literature review of scheduling models ending with selecting one type.
2/ Literature review to map current approaches to solve selected model by evolutionary computing with goal to define nowadays approaches.
3/ Developing own evolution algorithm.
4/ Design testing of developed algorithm with comparison to Simple Genetic Algorithm and Dispatching rules.
5/ Evaluating efficiency of developed algorithm.

Scope of Graphic Work:	
Scope of Report:	50-60
Thesis Form:	printed/electronic
Thesis Language:	English

List of Specialised Literature:

[1] MICHALEWICZ, Z. Genetic algorithms+ data structures= evolution programs. Springer Science and Business Media, 2013.
[2] BACK, T. Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms. New York: Oxford University Press, 1996. ISBN 0-19-509971-0.
[3] EIBEN, A. E. a J. E. SMITH. Introduction to evolutionary computing. Second edition. Heidelberg: Springer, 2015. Natural computing series. ISBN 978-3-662-44873-1.
[4] DASH, S. S. a S. DAS, B. K. PANIGRAHI a K. VIJAYAKUMAR. Artificial Intelligence and Evolutionary Computations in Engineering Systems. Springer, 2017. Advances in Intelligent Systems and Computing, 517. ISBN 978-981-10-3173-1.
[5] SIMON, D. Evolutionary optimization algorithms: biologically-inspired and population-based approaches to computer intelligence. Hoboken: Wiley, 2013. ISBN 978-0-470-93741-9

Thesis Supervisors: Ing. František Koblasa, Ph.D.
Department of Manufacturing Systems and Automation

Date of Thesis Assignment: November 20, 2019
Date of Thesis Submission: May 20, 2021

L.S.

prof. Dr. Ing. Petr Lenfeld
Dean

Ing. Petr Zelený, Ph.D.
Head of Department

Liberec, November 20, 2019

Declaration

April 29, 2020 Pandiyaraj Gnanasekar

ACKNOWLEDGEMENT

Hereby, I would like to express my gratitude to the following people who were very supportive and encouraged me to make this thesis a reality. I am extremely thankful for dedicating their precious time to motivate me and guided towards the success of this project.

Ing. František Koblasa, Ph.D. – Supervisor

Department of Manufacturing systems and automation, for his experienced and expert guidance, support, motivation, patience and a great inspiration to complete the thesis.

Ing. Petr Zelený, Ph.D. – Head of Department, Department of manufacturing systems and automation, for his encouragement and support for the work.

Technical University of Liberec – For giving me a wonderful opportunity to study and to learn new things in my life.

For my family members, who made my studies possible with financial support and their love and continuous support in all difficult times.

For my friends who were there when I needed them the most and helped me gain confidence to complete the work.

For the support of Student Grant Competition of the Technical University of Liberec under the project Optimization of manufacturing systems, 3D technologies and automation No. SGS-2019-5011.

Liberec, 2020 Pandiyaraj Gnanasekar

ABSTRACT

This thesis is aimed at research of evolution algorithms (EA) in the field of the shop scheduling problems and to develop a new strategy in order to improve the performance. Job shop scheduling problem (JSSP) is one of the most complex scheduling problem and finding the optimal solution is very difficult due to their complexity. Existing evolution algorithms were reviewed and one of the best and widely used genetic algorithm is selected for solving job shop scheduling problem. Active schedules for JSSP were generated based on various dispatching rules with the help of most used problem instances to compare effectiveness of EA. Then the structure and the major parameters of simple genetic algorithm (SGA) is reviewed and based on that a new strategy for replacement (Reusable Replacement Strategy) is proposed and implemented in the SGA. The implementation of RRS in SGA improves the results and also its impact on two different type of chromosome representations were experimented. The developed MSGA_{JO} is concluded to be the best genetic algorithm among tested to give the best makespan values for the JSSP problem instances.

Keywords
Simple genetic algorithm, Job shop scheduling, Dispatching rules, Evolution Algorithms

ABSTRAKT

Tato diplomová práce je zaměřena na výzkum evolučních algoritmů (EA) v oblasti plánování zakázkové výroby a na vývoj nové strategie za účelem zlepšení výkonu. Sekvenční rovrhovací problem (JSSP) je jedním z nejsložitějších plánovacích problémů a nalezení optimálního řešení je vzhledem ke složitosti velmi obtížné. Byly přezkoumány existující evoluční algoritmy a pro řešení sekvenčního rozvrhovacího problému byl vybrán jeden z široce používaných genetických algoritmů. Pro porovnání efektivnosti EA jsou vygenerovány nejprve Aktivní plány pro pro eta lonové problémy JSSP na základě různých prioritních pravidel . Poté je přezkoumána struktura a hlavní parametry jednoduchého genetického algoritmu (SGA) a na základě toho je v SGA navržena a implementována nová strategie nahrazení (opakovaně použitelná substituční strategie - RSS). Implementace RRS v SGA zlepšuje výsledky a také byl experimentován její dopad na dva různé typy reprezentací chromozomů. Navržený MSGAJO je považován mezi testovanými za nejlepší genetický algoritmus, který dává nejlepší hodnoty promísení pro případy problému JSSP.

Klíčová slova
Jednoduchý genetický algoritmus, plánování pracovních obchodů, dispečerská pravidla, evoluční algoritmy

Contents

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

In production technology, there will be no permanent solution for any operations and technologies. The changes are inevitable and improves the production with their new and innovative ways of optimization. One of such kind is the classical Job shop scheduling. Researchers and engineers have been developing numerous ways to optimize the problems of the Job shop scheduling from the origin of this concept of JSSP.

Job shop scheduling is a typical problem prevail in the production environment, where the jobs assigned to machines depending upon their technology order. It may seem to be a solvable problem at first instance, but it is still under the category of NP-hard (Non-Polynomial time) problems. The difficulty of assigning few jobs like 3 to few machines like 3 is quite easily achievable, as the number of possibility is small. But, if the size of jobs or machines increases, the possibility will rise exponentially and leading to find an optimal solution a near impossible one.

The difficult in finding an optimal solution for the classical Job shop problems makes many researchers to find a way to solve this problem. The optimal solution is the solution with the minimum makespan of jobs in case of Job shop scheduling. Makespan is the maximum time required to complete all the jobs assigned to the machines in the production environment.

This aim of this paper is to develop an evolutionary algorithm to optimize the job shop scheduling problem by using Matlab Programming tool.

This work includes,

- Literature review of the papers related to job shop scheduling and genetic algorithm along with other evolutionary algorithms
- Creating a simple genetic algorithm (SGA) based on Giffler and Thompson active schedule
- Exploring options to improve the SGA
- Implementing new strategy to improve the SGA
- Experimenting and comparing the developed algorithm with the SGA

The outcome of the experiment is analysed and based on the results further options and future developments will be concluded.

2. LITERATURE REVIEW

Many researches have tried numerous ways to find optimal solution, but one significant method which provides good result is the Genetic Algorithm(GA). Genetic algorithm is a biologically inspired concept in which the process is based on the principle of evolution. It imitates the biological genetic processes such as generations, crossover, mutations in order to find an optimal solution from number of generations. Genetic algorithms have been used widely after its discovery in the world of evolutionary computing in artificial intelligence. This concept is applied to wide variety of problems and helps to attain a reasonable result with justification. Problems like production scheduling, travelling sales man, graph theory are also uses genetic algorithm to get the results. In this chapter, basics of scheduling and its types, constructive algorithm, Giffler and Thompson algorithm and dispatching rules are briefly explained. These are the basic concept in which the genetic algorithm to be developed for job shop scheduling problems. This Thesis proceeds with simple genetic algorithm and then the experimentation of new strategies to improve the performance of SGA and then analysing the results.

2.1 SHOP SCHEDULING

Scheduling is the allocation of available resources over time of activities that are competing. Scheduling process answers the questions of when and where the job or work to be carried out during the production of a products. The question when comes to reveal the time at which the operation or activity going to be performed and the question where, reveals the place or area in which the operation is going to be executed. It is the most used subject in the field of operation research. The problems based on shop scheduling belongs to multi-stage scheduling problems, in which each job has a set of operations. In shop scheduling problems, there will be n number of jobs $J_1, J_2, J_3, \dots J_n$ which has to processed in a set of m number of machines $M_1, M_2, M_3, \dots M_m$ [1]. Each job consists of a number of operations denoted by O_{ij}, where (i, j) refers to the respected job and machine in which the operation is processing. The processing time p_{ij} of each operation will be given in advance for deterministic scheduling problems. The shop scheduling is classified into open shop, flow shop and job shop scheduling.

A. **Flow shop:** The problems associated with flow shop will have each job has operation number exactly equal to the number of machines m ($O_i = m$) and the technological route or the machine order will be same for jobs that passes through the machines. In other words, all jobs will have same technological order and no alteration of order is involved in flow shop scheduling.

B. **Open shop:** Open shop scheduling is typical and there is no technological order involved in job's operations. The job routes are not fixed before scheduling and each job has to be processed on any machine is usually assumed in solving the open shop problems. Like flow shop, each job will have exactly one operation on each machine.

C. **Job shop:** The technological routes for all jobs are fixed and can be different for each jobs. Unlike flow shops, the number of machines will not be equal to the number of operation of jobs, it may be equal, less or more than the number of operations, ($O_i < m$, $O_i = m$, $O_i > m$). These three possibilities are allowed in job shops and this is one of the criteria which makes the job shop problems are one of the most complicated problems. Job shops are employed in most of the manufacturing operations, which helps to solve more models than the other shop scheduling types. If the number of jobs n is greater than m ($n > m$), it can be arbitrarily large and will become extremely hard to solve and the difficulty increases exponentially with increase in number of jobs. The job shops are chosen in this paper in order to test the complexity of various problem sets. Job shop scheduling is explained in detail with its problem environment and methods to solve it.

2.2 JOB SHOP SCHEDULING

Job	: A piece of work, which has a series of operations.
Shop	: An area in which the manufacturing or modification of machineries.
Scheduling	: A decisive process with the aim to deduce the order of processing[2].

Job shop is one of the major classification in manufacturing processes. In this process type, small batches of wide range of custom products are produced. Each product in this process flow exhibits unique sequencing and set-up steps for its production. Some examples of job shops are paint shop, machining centre and a commercial printing shop. One of the major characteristics of job shop is its routing. Routing means the order in which the jobs are arrived to the production floor. Each job will use certain number of machines and no job will utilize every machines in the production floor. Jobs are characterized by its route, its requirement for processing and its priority. The key issue in job shops are the variety of mixed products and decision on how and when to schedule these jobs. The minimization of expensive machine set-ups and change-overs may not be achieved by the arrival pattern of jobs completed. The jobs or works can also be scheduled depending on their processing times, longest shortest processing time and shortest to longest processing time[2].

This problem of scheduling a job shop is considered as one of the most difficult problem in the production environment. Many researchers following this problem uses many unique approaches to solve this problem. The main objective of this problem is the minimization of makespan with the constraints prevailed in the allocation of jobs along with their unique job sequence. A typical job shop scheduling problem is the *n* x *m* minimum-makespan, which is commonly referred as JSSP. The JSSP consist of *n* number of jobs $\{J_k\}_{1 \leq k \leq n}$ which is to be processed on a set of *m* number of machines $\{M_l\}_{1 \leq l \leq m}$. The technological sequence of each job to be processed is unique and different from each other. Each job has a set of operations O_{kl} to be processed on each machines. These operation processing on machines requires a uninterrupted duration of processing time P_{kl}[3].

In Job shop scheduling, there are some constraints to be considered during the schedule generation. Those constraints are as follows,

1. Each job consists of a set of operations
2. Definition of machine sequence for each job.
3. The pre-defined machine sequence should be maintained to complete each job.
4. Processing time also includes the setup time for each operations
5. No operation cannot interrupt other operation.
6. Jobs should not be processed in the same machine twice.
7. Only one type of task can be dealt by each machine

These constraints determine and ensures the feasible solution in the process of generating the schedule for the defined problems. Apart from the above constraint many specific constraints can also be considered for the development of optimal solution for the typical JSSP problems.

2.3 CONSTRUCTIVE ALGORITHM

Constructive algorithm means a program which is constructed to find the possible set of solutions for a problem. These algorithms are widely used in combinatorial optimizations in which every solution *x* is a subset of the ground set *E* [4]. A constructive heuristic iteratively updates a subset $x^{(k)}$ as follows:

Step 1: The algorithm starts from an empty subset: $x^{(0)} = Ø$. It is a subset of the optimal solution.

Step 2: During each iteration *k*, selection of best element among the admissible elements, $i^{(k)} \in E$.

Step 3: It then inserts $i^{(k)}$ to the current subset $x^{(k)}$: $x^{(k+1)} = x^{(k)} \cup \{i^{(k)}\}$.

Step 4: Then the algorithm continues to step 2 until the solution becomes complete.

This is basic procedure for creating a constructive algorithm. Based on the same procedure, the Giffler and Thompson algorithm developed in order to solve some of production based problems.

2.4 GIFFLER AND THOMPSON ALGORITHM

Giffler and Thompson algorithm is always a good method for the shop scheduling problems in the production environment. It is a constructive algorithm which consider a machine scheduling problem, consists of assigning a set of jobs to a set of machines and fulfilling the given criteria’s optimization requirement. This algorithm also contains some limitations or constraints similar to the basic scheduling algorithm for JSSP and some more constraints, which make sure the feasibility of the solution.

1. At time zero, the set of all jobs should be available.
2. Each job in the set should have a series of linearly ordered operations which may be distinct, but each should have the same number of elements.
3. Only one job can be processed on each machine at a time or each machine can handle only one job at a time.
4. The set-up time of machines and transportation times between machines are not considered as they are negligible.
5. Failure of machines never occur. i.e. Machines are available always.
6. Pre-emption of operation are not allowed.
7. The specialization of each machine should be in the sense that it can perform only a certain type of operation[5].

The objective of this algorithm is minimizing the makespan, which is the maximum completion time. The difficulty of the JSSP is increased if the number of machine and jobs increases. The more the number of jobs and machines, the more it will require time as the complexity of problem increases exponentially.

2.4.1 GT algorithm

Giffler and Thompson developed a recursive algorithm to generate active schedules in a systematic way[6]. The algorithm of Giffler and Thompson (GT) uses the following symbols for the representation of its operation in each steps of the algorithm,

n – Number of jobs.

m – Number of machines.

o_j – operation j of each job ($1 \leq j \leq nm$).

p_j – processing time of each operation o_j.

P_t – partial schedule of the $(t - 1)$ schedule operations.

S_t - set of operation available to schedule at time t or at each iteration t.

ρ_j – It is the earliest possible time at which operation o_j from set S_t can start.

β_j – It is the earliest possible time at which operation o_j from set S_t can be finished. i.e. $\beta_j = \rho_j + p_j$.[6]

With the defined symbols, the algorithm of Giffler and Thompson as follows:

Step 1: Let $t = 1$ and $P_{t-1} = \{\}$. S_t is the set of all the available operations without any predecessors.

Step 2: Finding the minimum value $\beta^* = \min_{oj \in Sj} \{\beta_j\}$ and the corresponding machine M^* at which o_j with β^* is executed. If there are multiple machines M^* for the selected operation o_j, choose arbitrarily.

Step 3: An operation o_j is chosen from S_t with the following conditions:

- Operation o_j requires machine M^*, and
- $\rho_j < \beta^*$

Step 4: After the selection of operation, proceed to next iteration by:

- adding o_j to P_t, which results in P_{t+1}
- Deleting o_j from S_t and forming S_{t+1} by adding the successor operation of o_j to S_{t+1} (not applicable for the last operation)
- $t = t + 1$

Step 5: If $S_t \neq \{\}$ go to Step 2. Or else stop.[6]

The above GT algorithm generates an active schedule with a feasible solution (may not be an optimal solution). Each iteration, GT algorithm deals with a conflict set S_t from which an operation is chosen based on some criteria or rules and then scheduled. This recursive procedure by Giffler and Thompson is executed until all operations are scheduled. The order of precedence is preserved. Steps 2 and 3 of this algorithm ensures that there is no operation of job that can start earlier and respects the precedence constraint, without delaying other operation.

The steps 2 and 3 of the above algorithm can also be modified to generate a non-delay schedule, in which a machine is never idle when there is an operation available to be executed on that machine.

The modification can be done as follows:

Step 2: Finding the minimum value $\rho^* = \min_{oj \in Sj} \{\rho_j\}$ and the corresponding machine M^* at which o_j with ρ^* is executed. If there are multiple machines M^* for the selected operation o_j, choose arbitrarily.

Step 3: An operation o_j is chosen from S_t with the following conditions:

- Operation o_j requires machine M^*, and
- $\rho_j = \rho^*$

The non-delay schedule improves the solution quality but the probability of not considering an optimum solution prevails with this schedule. For the search of optimal solution active schedules are more appropriate[6].

2.4.2 Dispatching rules

In scheduling and sequencing problems, there are some issues related to select which operation to be scheduled at a particular stage. There could be conflict among the operations to be scheduled. This conflict arises when two or more operations requires the same machine at the same time. These conflicts are generally solved through some rules known as priority or dispatching rules. These dispatching rules ensures a certain job from the conflict set is scheduled at the certain time based on some criteria. There are hundreds of dispatching rules that can be used based on the requirement of the scheduling problems. Some of the most common classification of dispatching rules are as follows,

I. **Simple priority rules:**

The characteristics of these jobs are usually dealing with the information processed by the jobs, like processing time, remaining number of operations, waiting time, etc. There is also certain rule which follows random selection which are not dependant on information with respect to a specific job. These rules are classified based on

- Processing times
- Setup times
- Number of operations
- Due dates
- Costs
- Arrival times
- Miscellaneous information.

II. **Combinations of simple priority rules:**

The above simple priority rules are sometimes combined to solve the conflicts among the jobs to be scheduled. The combination is applied by either to each set of conflict with different group of jobs or under the same queue with different circumstances.

III. **Heuristics scheduling rules:**

There are some rules which consider more complex criteria such as anticipated machine loading, effect of alternate routing, alternate operation scheduling, etc. These standards are typically utilized related to the principles in Simple Priority rules. In some cases, the heuristics may include non-mathematical parts of human insight, for example, inserting a job in an idle time availability by visual assessment of a schedule[7].

2.5 EVOLUTIONARY ALGORITHMS (EA)

Evolutionary algorithms are generally based on the nature's principle of evolution and are classified as genetic algorithms (GA), evolution strategies, genetic programming and evolutionary programming [1]. These categories are based on population of individuals and are widely used for problems like production, scheduling, distribution, inventory and location based problems.

Evolutionary algorithms is first used for shop scheduling problems in the year 1980. It was first applied to flow shop [8] and job shop scheduling problems [9]. The components and structure of genetic algorithm was discussed by Goldberg [10] or Beasley et al.[11].

Particle Swarm Optimization (PSO) is one of the evolutionary computation, which is proposed by Kennedy J, Eberhart R C [12]. PSO is popular in solving problems based on swarm intelligence paradigm. The idea of PSO is inspired from the social psychology and swarming theory and it simulates the real life swarms like flocks of birds and schools of fishes that search for food [13]. PSO evolves solution based on individual and group experiences, instead of using evolutionary operators like crossover and mutation. The basic idea is that the algorithm evolves based on the shared information among the population.

Ant colony algorithm is also based on evolutionary computation which mimics the pheromone trails used by ants to search food with their medium of communication and feedback. This algorithm was first proposed by Marco Dorigo in 1992 [14]. This algorithm uses the concept of pheromone trails to improve the solutions iteratively and similar to PSO, this algorithm uses shared information through communication and feedback among within the population and will not use evolutionary operators like Crossover and Mutation.

Memetic Algorithm is first proposed by Moscato and Norman [15], based on the individual enhancement of solutions of agent that interrelated to one another under a process of cooperation and population competition [16]. This algorithm has been widely applied for knapsack problems, scheduling problems, routing problems and spanning tree and based on evolutionary operators like crossover and mutation.

There are also some of the evolutionary programmes and strategies like simulated annealing [17], tabu search [18]and neural networks [19]. Among these evolutionary algorithms, Genetic algorithm has been widely used and applied in various fields of computation right from its discovery. The computation effectiveness and the ability evolve the solutions makes the genetic algorithm more reliable for the computation of maximization and minimization problems [20].

Genetic algorithms are similar to the algorithms researched from the year 1950's and became most popular by J.H. Holland in 1975 [21]. Many researchers preferred genetic algorithm for solving production scheduling related problems. This is proven by the data obtained from last two decades of researches [1]. This paper also based on the genetic algorithm and experiments some strategies and discovering the effect on its implementation and how it improves the solution during the whole computation.

3. GENETIC ALGORITHM

One of the widely used evolutionary computing technique is genetic algorithm. It is based on the Darwinian theory of evolution. This concept imitates the essential aspects of evolution of the organisms such as gene cross over, mutation, survival of fittest, development of chromosomes in each generation, etc. Genetic Algorithm is a powerful set of stochastic global search techniques, which are used to solve wide range of complex problems. The Genetic Algorithm will be referred as GA in the upcoming review.

A typical GA starts with the generation of populations of individuals(chromosomes), generating fitness function which evaluates each individual, selection of individuals, application of crossover and mutation techniques and finding the best individual through a number of iterations. The various stages of GA are shown in figure 1,

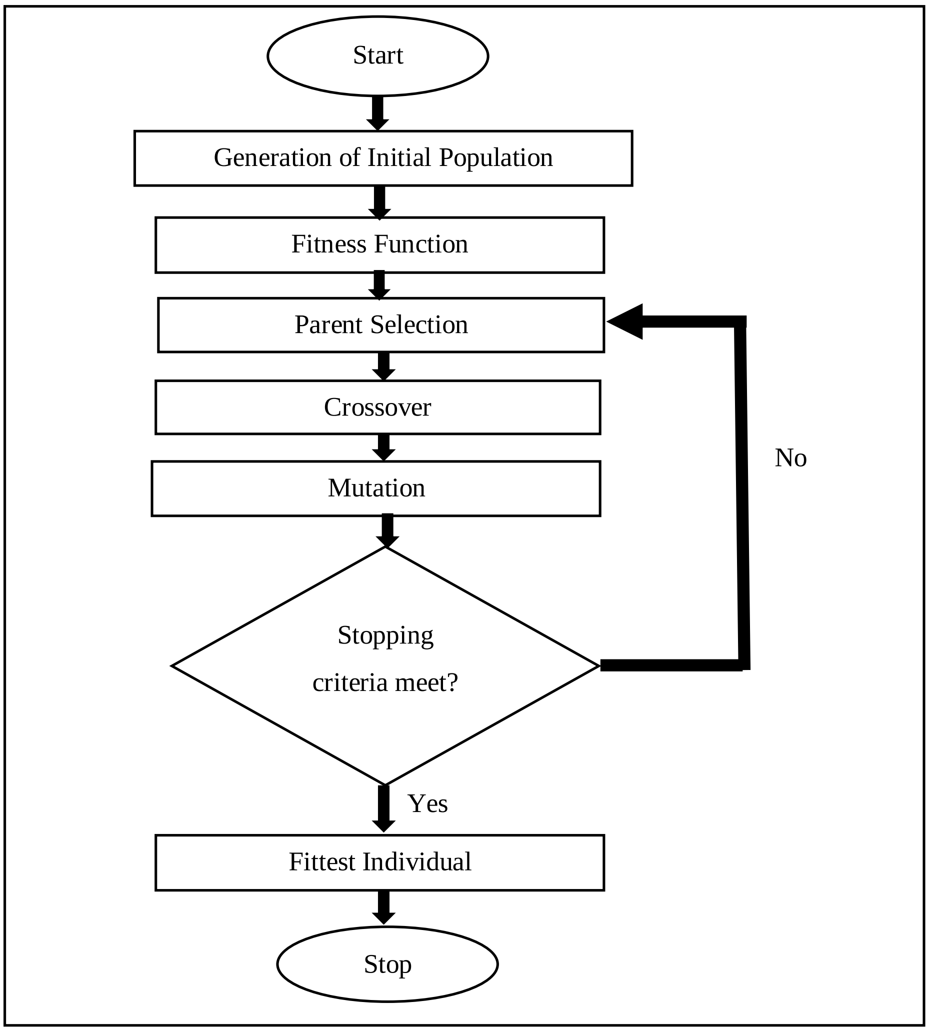

Figure 1. Evolution model of genetic algorithm [22]

3.1 CHROMOSOME REPRESENTATION

A chromosome is a set of parameters which characterizes a solution for the problem that the GA is attempting to solve. The population, chromosome and gene are illustrated in the figure 2. The chromosome can be represented either by direct or indirect methods. Each chromosome consists of data for each activity or operation which are called as gene. These solutions or individuals are cumulatively called as population. By traditional methods, the chromosomes are represented in binary as series of 0s and 1s. The chromosomes can also have encoded through other possible methods also. In fact, any series or strings that allows the solution should be represented as a finite length can be used. The chromosome is called as parent before application of genetic operators and chromosome formed after the application of genetic operators are called as offspring. The chromosomes are decoded in two ways, direct and indirect. Direct encoding contains the information of the solution by itself, but in indirect encoding a set of rules or constructing the solution. The chromosomes representation is categorized as two ways,

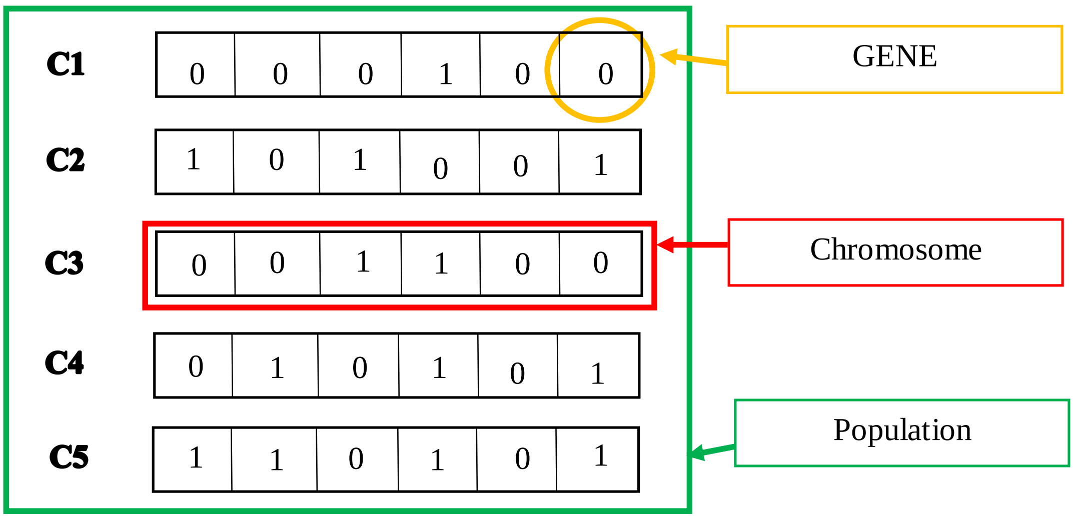

Figure 2. Gene, Chromosome, Population [23]

3.1.1 Binary representation

In this representation, the genes in the chromosome is represented as 0s and 1s. the solution is converted in to binary values in order to employ the genetic operators like crossover and mutation and generate a solution for the problem to be solved. The length of the chromosome will be depending on the size of the problem given. A typical example for a binary representation is 10011011 is the binary string for the number 155. Similarly, each chromosome possesses a certain value which helps to solve the problem.

3.1.2 String representation

This representation is commonly used for order based problems like production scheduling problem, travelling salesman problem where the problem arises when and where the schedule should occur and how it should proceed. The representation can be in any form. It can be a set of alphabets, set of numbers in which each letter or number carries the necessary data for the solution set. The decoding of the chromosomes should also be considered for an uncommon chromosome representation. The representation should allow the algorithm to generate a feasible solution. A typical example of string representation may be set of permutation numbers (654312987). This representation can be the order in which a salesman travel around the cities or a job which visits the machines for its operations. Some of the commonly used representations for JSSP in genetic algorithm are priority-based representation, job based representation, machine based representation, algorithm based representation permutation representation, matrix representation[24].

i. Operation based representation

This representation is also known as permutation with job repetition, in which the sequence of operations and each gene in the chromosome represents one operation [25]. All operations of jobs are represented by their respective job number and the number of occurrence of their job number in the chromosome is the order of the operation to be scheduled for the respective job number. This interpretation of the order of occurrence in the chromosome sequence produces a feasible solution.

2	1	3	3	1	1	3	2	2

Figure 3. Operation based representation [25]

From the figure 3, the chromosome can be encoded as follows, the first operation of job 2 is scheduled followed by first operation of job 1, then first operation of job 3 and followed by second operation of job 3 and so on. The occurrence of each job defines the order of execution of each operation of a job.

ii. Job order based representation

This is one of the permutation representation, in which the chromosome is set of multiple string of job order for machines instead of one complete string. The chromosome id divided in to substrings of machines and each sub string consist of the order in which jobs are scheduled. This representation are often breaks the technological constraints, which results in infeasible solutions. A separate algorithm is required for converting the infeasible solutions to feasible one.

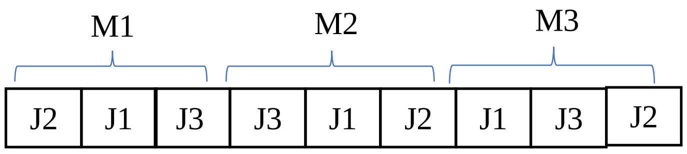

Figure 4. Job-order representation [28]

Figure 4, is a typical example of job-order representation, the jobs *J1*, *J2* and *J3* are scheduled in the machine 1 in the sequence order of 2,1,3.

iii. Random key representation

In this method, the representation [26] is similar to operation based representation, except that each gene in chromosome is filled with random numbers between 0 and 1. The random numbers in the chromosomes are then sorted out and the order resulting from the operation is replaced with the integers (the order). Then each operation in the chromosome is assigned with an integer value and the resultant string will be in the length of operations. A scheduling algorithm can also be used to decode the chromosomes or schedule the operations based on the information in the chromosome.

iv. Machine based representation

This representation [27] is based on the string of genes with a total length equivalent to the number of machines. The machine sequence in the chromosome represents the order in which the machines are scheduled and the bottleneck machines can be identified with some heuristics like shifting bottle heuristics [28].

2	1	3	3	1	2	1	3	2

Figure 5. Machine based representation[29]

In figure 5. An example of machine based representation is given, where each number represents a machine and schedule of those machines is based on the order in which they are arranged.

v. Priority rule based representation

It is one of the algorithm based representation, where chromosome is represented as a string of *n*-1 entries (R_1, R_2, …. R_{n-1}), where *n*-1 is the number of operation in the problem instance and R refers to the rule from a set of priority rules which is chosen beforehand. GT algorithm is used to solve the conflicts set with the help of rules in the chromosome[30]. This representation is used mostly in solving the job shop scheduling problems.

vi. Preference list-based representation

In this representation, instead of using a single string for all the operations, a string of operation for each machine is used. This makes it to be a direct representation of the processing sequence based on PIAN model [28]. However, this type of representation are frequently violates the technological constraints and requires additional computation for repairing the infeasible chromosome.

vii. Matrix representation

Matrix representation is applied to the job shop scheduling, where the matrix contains the number of jobs and its sequence within it [31]. This type of representation requires some additional computation to solve the problem.

3.2 INITIAL POPULATION

Population is the set of chromosomes or individuals which is a group of solution which offers the algorithm to search the best solution. Defining initial population is crucial in genetic algorithm, as the solution will converge to the local optimum if the population size is small or the solution will converge to the global optimum if the size of population is bigger, which eventually requires more time for processing. So the right population size should be defined based on the problem requirement. The initial population are generated either randomly or using some creation functions. However, the creation functions should satisfy all bounds and linear constraints. These limitations will help the program to generate feasible solutions. The encoding method of each individual in the initial population differs from problem to problem.

3.3 FITNESS FUNCTION

The fitness function is the function that evaluates the chromosome fitness based on the objective function for its phonotype[32]. The chromosomes in the initial population are first decoded and then evaluated based on some heuristics function. This fitness function is in evaluation phase of the genetic algorithm, which not only ensures the fitness of the chromosome, but also the capacity of the chromosome to produce the feasible solution. The computation time of the real-life objective function can be very [35]. An intelligent fitness function, which ensures that if there is a chromosome with same fitness, it will not be tested multiple times. Alternate possible approach is that usage of approximation of the fitness function [33].

An effective and famous method for generating an active feasible schedule for classical job shop scheduling is Giffler and Thompson (GT) algorithm. Optimal solutions are always active and for generating active schedules GT algorithm has been used. The GT algorithm provides the makespan (objective value) value of each chromosome and a small modification is used to repair the chromosome if it is not feasible.

3.4 SELECTION

This is the process of choosing chromosomes to produce new offspring from reproduction. The selection of individuals is carried out in several ways. Many researchers have framed unique way of selections and few general methods. Here, the commonly used methods are explained below.

3.4.1 Roulette Wheel Selection(RWS)

This is a stochastic method involves in proportionate selection as it selects the chromosome or individuals based on probability rate using the basic roulette-wheel procedure. In roulette-wheel, a ball is thrown in to a wheel of numbers and the ball acts as a pointer. Each time the ball settles in the wheel when the wheel stops. The number where the ball settles is the selected one. The same procedure is followed in selecting the parents from the population [33]. The probability of selection is aligned with the fitness, where there is a chance of reproducing the same individual with the low. This selection method will not produce population with the best fitness but also individuals with worst fitness as the selection is based on the probability. This ensures that the population will always have mixed individuals and finding the optimum solution.

All individuals in the population are placed on the roulette wheel based on their fitness value and each one is assembled in the roulette wheel. Each segment of the roulette wheel is distributed based on the fitness value of the chromosomes. If the probability value of the individual is bigger, then it will occupy a big space in the wheel and if it is small then it will occupy only a small space and has less probability of getting selected. The individuals responsible for the segment is selected after the wheel stops. The same process is repeated for the desired number of parents to be in the mating pool. By this method, there is no guarantee that the good individuals with good fitness will be selected in the mating pool.

There is also a probability in which the best individual can also be missed from the population.

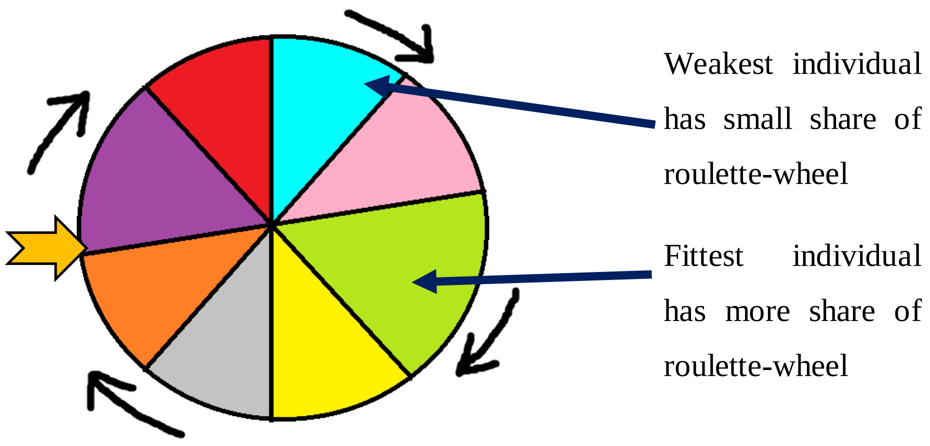
Weakest individual has small share of roulette-wheel
Fittest individual has more share of roulette-wheel

Figure 6. Roulette wheel fitness distribution [34]

The roulette wheel algorithm can be explained as follow; The expected number of individuals in the mating pool should be determined as the initial step. Then the probability value of each individual is computed by dividing each individual's fitness value with the sum of all individual's fitness value in the population. For minimization problems, the minimum fitness value is divided with the fitness of each individual and this whole value is divided with sum of all minimum fitness value divided by the individual fitness value. The probability of selection $\boldsymbol{P_S}$ can be expressed mathematically by (1),

$$\boldsymbol{P_S} = \frac{\frac{\min(f(x))}{f(x)}}{sum\left(\frac{\min(f(x))}{f(x)}\right)} \qquad \textbf{(1)}$$

Where, $\boldsymbol{P_S}$ = Probability of selection,

$\boldsymbol{f(x)}$ = makespan (Objective function).

Then, the cumulative probability is computed by adding the probability of an individual with the partial total value of the probability of previous individuals. After that, a random number between the interval 0 and 1. A loop will execute continuously until it finds the cumulative probability number equal to the random number. The index of that individual is selected and the individual is added to the mating pool for the reproduction. This process runs repeatedly up to the number of expected count of the population.

3.4.2 Tournament selection

In this method, the individuals are not selected based on their fitness values. The individuals are selected randomly i.e., two individuals are selected randomly from the population and their fitness value is evaluated and the individual with the best fitness is added to the mating pool. The process will be repeated to desired number of times to get the required population in the mating pool. This method of selection is very time-efficient than some other methods. Also, there is a less chance of strong individuals to get selected as the selection is random[33]. A selection mechanism in GA's is the process of selecting the better individuals in the population which favours the mating pool to be filled with good individuals, as the genes in the individual are inherited by the next generation[35].

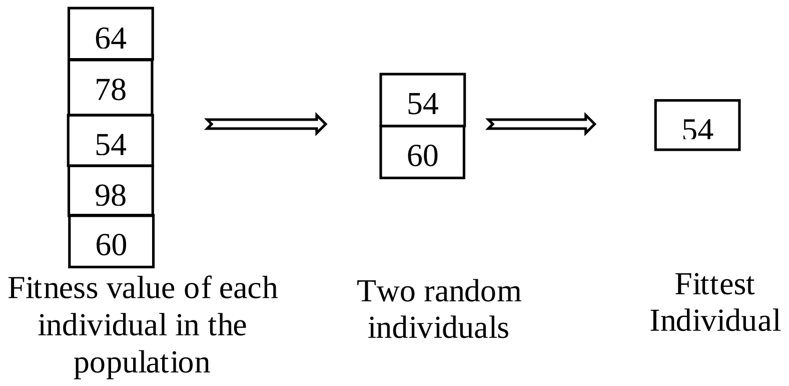

Figure 7. Tournament selection [37]

The figure 7 illustrates the tournament selection for the minimization problems and the fittest individual may be changed for the maximization problems, as the fitness value might be taken as bigger value. For problems like job shop scheduling the selection is usually based on the smallest fitness value, as it deals with the optimization of makespan as one of its objective criteria.

3.4.3 Rank selection

This selection method is simple and requires few steps to select the individual from the population. The individuals in the population are assigned with a number based on their fitness value. Then, the individuals are sorted in ascending or descending order based upon the requirement of the problem to be solved. After that, the top most individual is selected and included in the mating pool for reproduction. The numbering or scaling of each individual in the population can be done by using the linear fitness scaling, Boltzmann fitness scaling, sigma scaling, linear rank scaling, non-linear scaling, transform ranking [33]. This method allows the algorithm to explore the whole search space which prevents the premature convergence.

3.5 CROSSOVER

Crossover is the process of producing new chromosome or individual in the hope of finding a better individual than the existing individual. From the perspective of optimization, crossover helps to exploit the search by recombination of genes in the individuals. Crossover method can differ in the way that they traverse the search space. The parameter P_C denotes crossover rate, which gives the probability for the application of crossover function to the generated offspring from the parents selected. Most papers, recommends to use $P_C \geq 0.6$. The classical crossover operators like one-point, two-point and uniform crossover can be used for the binary represented chromosomes. Application of these crossover methods to the non-binary or permutation representation may result in infeasible solutions or offspring. So, for such kind of representation operators like PMX (Partial Mapping Crossover), JOX (Job Order Crossover), OX (Order based Crossover), LOX (Linear Order Crossover), CX (Cycle crossover), OBX (Order based Crossover), PBX (Position based crossover) operator[1]. Almost every operator follows the principle of two-point crossover, the only criteria differs in each operator was the heuristics used to produce a feasible offspring after application. Some operators used in production scheduling are explained as follows.

3.5.1 One-point crossover

This method is one of most commonly used crossover method, in which two parents are selected from the mating pool and parts of each parent is exchanged and new offspring produced. The process starts with randomly locating a gene in the parent individuals which is called as “cross over point”. The part from start to the cross over point is taken from the parent 1 and from cross over point to the end is taken from the parent 2 and merged to form a new offspring (child 1). Similarly, the left over parts of each parents are merged to form second offspring (child 2). Thus, two offspring can be produced by mixing two parents. This method has some limitations, if the individual has a bigger defined length then the individual might get damaged with one-point crossover. In order to manage this situation, two-point crossover can be used for individuals with long defining length. An illustration of one-point crossover on individuals with binary representation is shown in figure 8.

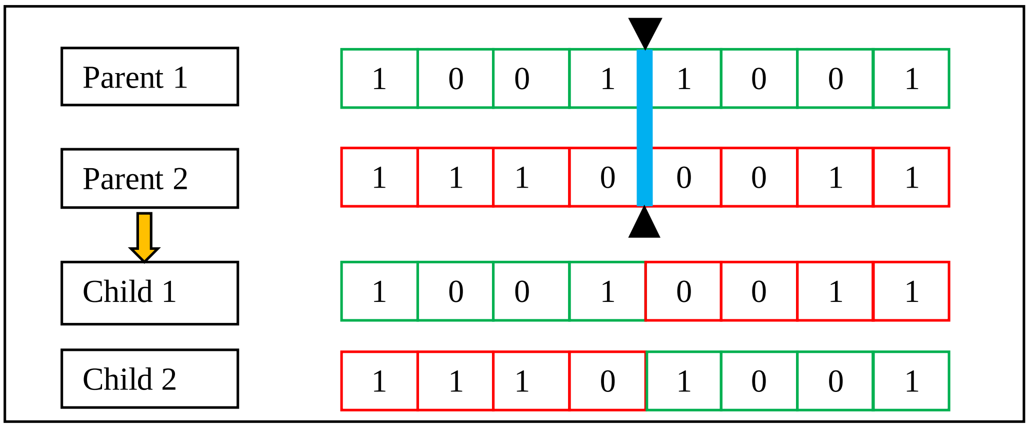

Figure 8. One-point crossover[36]

3.5.2 Two-point crossover

The working principle is same as the on-point crossover for this method. The only difference is that this method has two points to cut the parents and interchanged to form children. The bits or genes in between two points are taken from one parent and remaining genes are taken from the other parent. This operator is used for individuals with high definitive length. An illustration of two-point crossover on individuals with binary representation is shown in figure 9.

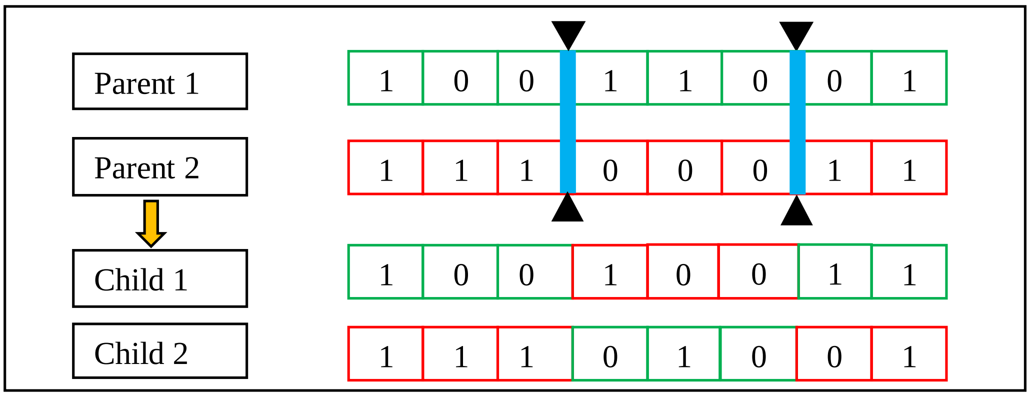

Figure 9. Two-point crossover [1]

3.5.3 Uniform crossover

In this operator, the offspring are produced with the help randomly generated sequence, which consist of numbers 0s and 1s. This sequence is called as bit mask [1]. The principle of this operator is that the genes from one parent is selected if the corresponding number in the bit mask is 0, if there is different number (1) then the gene from the other parent is taken and a new offspring is produced. For child 2, the same procedure is followed where the genes from one parent is taken if the corresponding number is 1 and if it is 0 then the gene from other parent is taken and forms the child 2. This principle is shown with an illustrative example through binary represented chromosomes in figure 10.

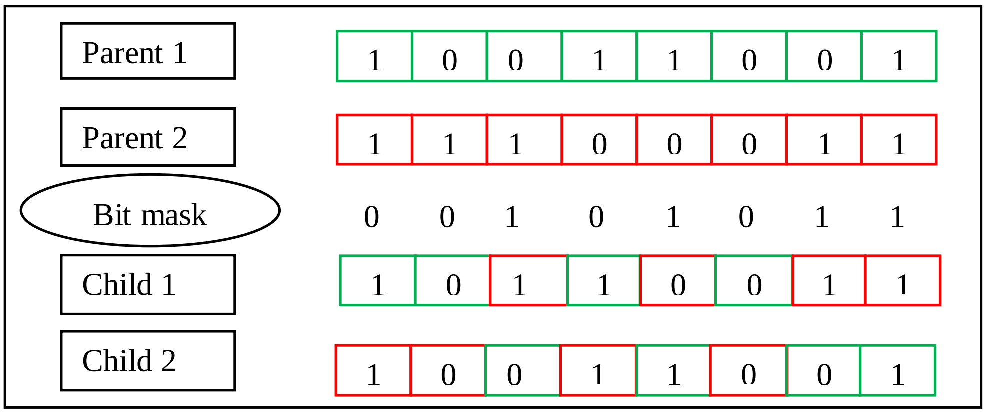

Figure 10. Uniform crossover [1]

3.5.4 Partial mapping crossover

The PMX operator is based on the two-point crossover principle, where the chromosome is divided with two cut points and the genes between them are interchanged among the selected parents from the mating pool. The genes between the two cut points from the first parent is replaced with the genes between the two points of second parent. This will result in an infeasible child and this child should be repaired to get the feasible outcome. This operator stores the position of the genes between the points and will use those positions to repair the child. The crossover operation and the repairing by the PMX operator can be shown in the figure 11. In the shown illustration, the genes with number 4 and 6 are repeated in the proto child and hence the gene are mapped to their respective numbers. The gene with number 4 is mapped to 7 and 7 mapped to 2, so the gene with number 4 is replaced with 2 and similarly, the gene with number 6 mapped to 1, so it is replaced with 1 [1]. Thus, the child becomes feasible.

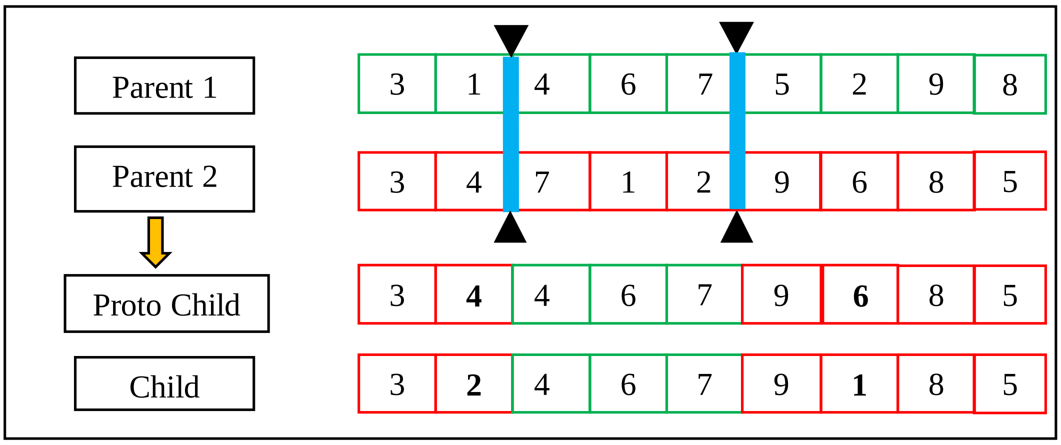

Figure 11. Partial mapping crossover [8]

3.5.5 Order crossover

The order crossover is first designed for the Travelling salesman problem, in which two cut points are located randomly and the genes between the cut points are chosen from the first parent and the remaining gene with indices not equal to the selected genes between the two cut points are taken from the second parent and filled in to the child in a relative order, starting from the last gene of the genes taken from the first parent [1]. A typical example of this principle is shown in figure 12.

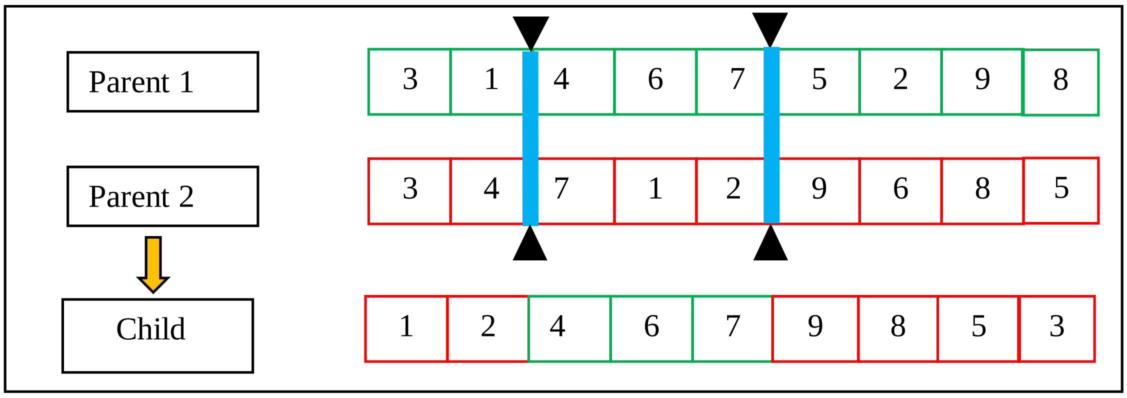

Figure 12. Order crossover [20]

The genes 3,4 and 5 with indices 4,6 and 7 present between the two cut points in the parent 1 is first extracted and filled the child. The next step is that filling the remaining gene position of child with the gene indices not equal to the indices 4, 6 and 7. The genes are filled in the relative order of the parent 2 and the filling starts from the last gene index 7. The remaining gene are filled in the order 1,2,9,8,5,3.

3.5.6 Job order crossover

This is one of the operator used for chromosomes with permutation representation. In this operator, the principle is based on randomly selecting a job number and taking all the genes with the same job number from the first parent along with their position and filling the child chromosome. The remaining genes with other jobs are taken from the second parent and the child is filled with these genes with the order in their parent. Similarly, the same randomly selected job is taken the parent and the process carried for forming the child 1 is followed to form child 2. The crossover mechanism of job order can be shown in the figure 13.

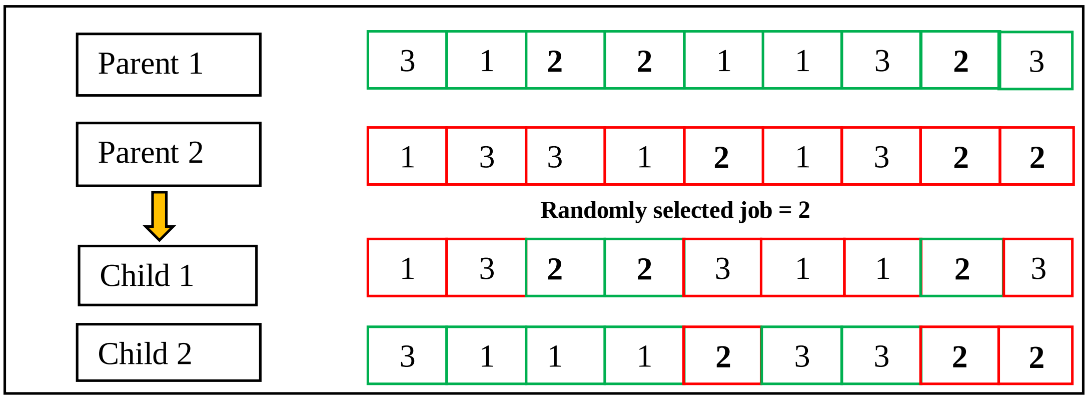

Figure 13. Job order crossover [37]

From the illustration, the randomly selected job number is 2 and was taken from the parent 1 and filled in the same position in the child 1. Then, the remaining gene not equal to the selected job number are taken from the parent 2 and filled the child with the relative order from the parent 2. Likewise, the child 2 also formed by taking the gene with selected job number from parent 2 and remaining genes are taken from the parent 1.

3.6 MUTATION

Mutation is the process of inducing extra variability to the individuals in the population, which prevents the premature convergence. This mutation disturbs the solution by altering the genes in the chromosome or individual. The children produced from the crossing operation might be subjected to the mutation operation if it is associated with the mutation probability. The mutation probability is usually ranges between 0 and 1. There will a designated parameter at the start of the algorithm and this mutation probability is compared to the designated parameter and if the mutation probability is less than the parameter, then the mutation takes place in that child. After the mutation operation, the offspring produced form the mutation is added to the population[16].

For Mutation operator, three main requirements exist. The first condition is that each point in the individual must be reachable from an arbitrary point. Not every mutation operator has this guarantee this condition of reachability. For instance, there might be difficulties in decoding approaches covering the overall solution space. The second requirement for the good design of mutation operator is unbiasedness. The mutation operator does not cause a search drift to a specific path, at least in spaces with no plateaus in unconstrained solution. In constrained solution spaces bias can be advantageous. The third principle requirement of mutation operator is the scalability. Every mutation operator should provide the degree of freedom for a adaptable strength [36]. This adaptable strength is usually possible for mutation operator based on probability distribution.

3.6.1 Bit flip mutation

This is the simple mutation operator which is applied to the individuals with binary representation. This operator works based on selecting a random bit from the individual and inverse the number in that bit. That is the number is flipped. For example, if there 0 in that bit, then it will be flipped to 1 and the individual mutated to a new child. An illustrative example is shown in figure 14.

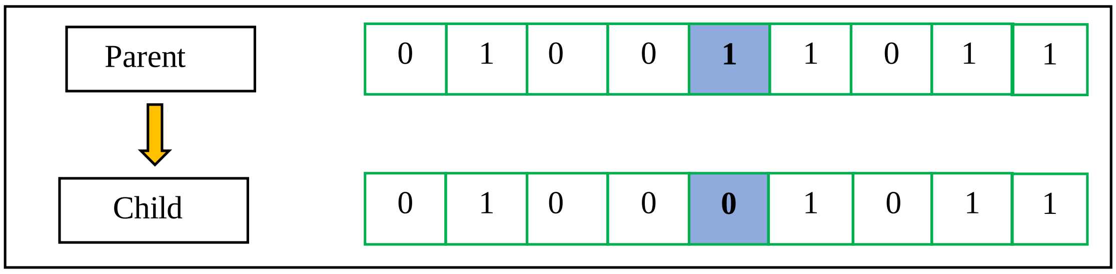

Figure 14. Bit flip mutation [26]

The bit in the position 5 has a binary value is chosen randomly and the value is flipped to 0 in order to mutate the parent to form a new child. This is a simple mutation operation in the genetic algorithm.

3.6.2 Swap mutation

In this mutation, two positions on the individual is selected randomly and then the number in that bit is swapped between the two positions of the individual. This type of operator is common in permutation based representations. A typical example of this mutation operator is shown in the figure 15.

The positions 2 and 6 are selected randomly and the numbers in the genes of the individual is interchanged. That is, the number 8 in the bit 6 is interchanged with the number 1 in the bit 2 and this number 1 is interchanged to the bit 6 and the changes are highlighted and shown in the example.

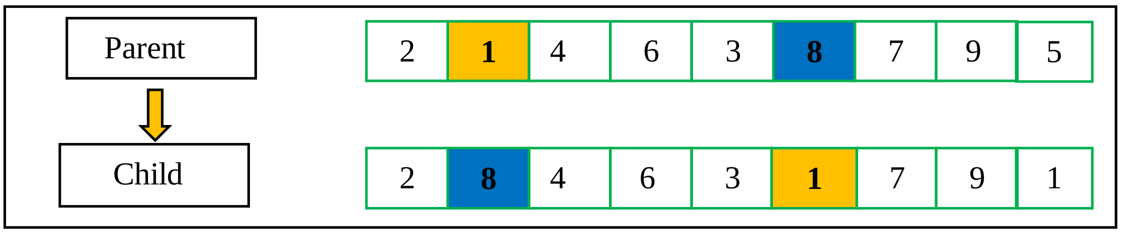

Figure 15. Swap mutation [38]

3.6.3 Inversion mutation

This mutation operator deals with the inversion of a subset within the chromosome to create a new child from the selected parent for the mutation operation. A subset of chromosome is selected either randomly or with some heuristics and then rearrange the genes in the subset in the reverse order [33]. The original order of the genes in the subset is reversed to form the child. An example of this mutation operator is shown in figure 16.

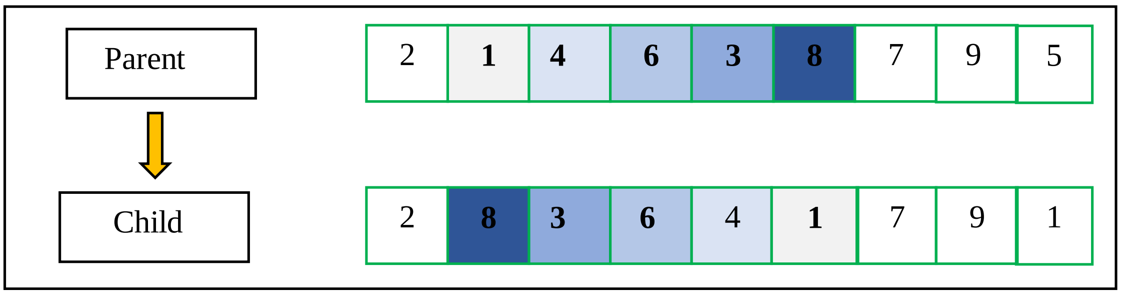

Figure 16. Inversion mutation [2]

The figure 17 shows the working of the inversion mutation, the subset of genes with numbers 1,4,6,3,8 are inversed in the order 8,3,6,4,1 and highlighted in the illustration. This mutation can be applied for individuals with large definitive length, which enables those individuals to prevent premature convergence.

3.6.4 Scramble mutation

This mutation follows the same principle of the inversion mutation, in which a subset of genes is selected from the individual. Instead of inversing the order of those genes in the subset, the genes are shuffled or scrambled [33]. So that the individual will create a child with different order of genes in the chromosome. Similar to the inversion mutation operator, this operator can also be applied for the large chromosomes and chromosomes with the permutation representation. Most of the chromosomes with the permutation representation uses the scramble mutation operator and known as a widely used mutation operator among them.

3.6.5 Shift mutation

The mutation is carried out by randomly selecting a gene in the chromosome and a random position is selected. The chosen random gene is inserted in the random position and rest of the genes in the chromosome is shifted towards right or left [7]. This mutation is also called as insert mutation. The shifting of genes within the chromosome impacts a great change in the chromosome.

3.7 REPLACEMENT SCHEME

The last step in the process of breeding in every genetic algorithm is the replacement of the new offspring to the current population. This replacement is carried through various methods and techniques. The decision of choosing the individuals to remain and the individuals to get replaced or deleted from the current population is made with the help of these replacement techniques. This scheme is classified in two major categories, (1) Generational replacement, (2) Steady state replacement.

3.7.1 Generational replacement

In generational replacement or full replacement, the entire population of individuals is replaced at each generation. This means, the new offspring population generated at each generation will proceed forward to the next generation by replacing all the individuals or chromosomes in the current generation. Generational replacement is classified into two derived forms. (μ+λ) replacement and (μ, λ) replacement[39], where μ is the current population and λ is the newly generated population.

i. (μ+λ) Replacement

In this strategy, both offspring and parent population are grouped into one population. Both the population of parents and children compete for the survival. After combining both population, they are sorted based on their fitness value. After sorting the set of best chromosomes equivalent to the population size are selected to proceed for the next generation of population.

ii. (μ, λ) Replacement

In this replacement strategy, the offspring generated may be far greater than the number of parents. The children created in each generation are ranked based on their fitness value and best children or offspring are selected to replace the individuals in the current population. Unlike μ+λ replacement strategy, the offspring generated is only being used to replace the parent population.

3.7.2 Steady state replacement

Steady State Replacement involves in overlapping strategy in which a small fraction of individuals from the offspring population is selected and replace the current population during each iteration. The new individuals from offspring population are inserted in to the current population as soon as they created. The advantage of this strategy is that best fit offspring can participate in the genetic operations in each iteration within generation, instead of waiting for a generation to complete. The worst individual from the current population is replaced with the best fit offspring from each iteration.

Whitley, proposed GENITOR concept, in which the worst individuals from the parent or current population were replaced deterministically during each iteration. This makes rapid improvements in the mean value of population fitness. On the other hand, in some cases, it may lead to premature convergence due to its focus on individuals with best fitness.

The steady state replacement uses different strategies for replacing the individuals,

- Replace worst
- Replace random
- Replace Parent
- Replace most similar(crowding)
- Replace weak parent.

i. Replace worst

This strategy is the most common and one of the elitist scheme of replacing the individuals in the current population. The worst individual is replaced with the individual with the best fitness value at each iteration after the application of reproduction and mutation operators. This is one of the effective optimizing strategy for finding the optimum solution quickly.

ii. Replace random

In this strategy, random individuals from the current population is selected and replaced with children produced from each iteration. The parents of the children are also included as candidates for the selection of random individual, which may lead to introduce weak offspring to the population. The disadvantage of this strategy is that, the probability of replacing the best individual in current population is quite high and there is a risk of replacing the solution that can produce optimum results.

iii. Replace parent

In Replace parent strategy, the parents involved in the crossover and mutation is replaced with the offspring they produced. This may result in losing best individuals in population during every iteration of each generation. There is also a chance of replacing almost every individual in the current population.

iv. Replace most similar (Crowding)

This technique involves, replacing the individuals with similar fitness value. The offspring produced is kept, one offspring is first compared with the current population and replace the individual which is most similar and the same principle is followed by the other offspring produced. There is minor possibility for the second offspring to replace the first offspring from the population.

v. Replace weak parent

After the operations of crossover and mutation, the best individual either parents or offspring is selected and replace the worst parent of offspring. The offspring produced and parents involved in genetic operation is compared among themselves and evaluated. Based on their fitness value, the population gets replaced. Either one parent and one child get replace or both parents gets replaced or it can delete both children and the parents will remain in the population

vi. Correlative family based replacement

Among four individuals (parents and children), the best single individual is selected as first survivor and the second individual is selected from the remaining individuals which has the highest distance from the best individual

vii. Other related strategies

Replace random n individuals – In this method, n individuals chosen arbitrarily from the current population and replace by n best individuals from the new generation of offspring. This kind of strategy corresponds to generational replacement scheme as the replacement is done for every generation and not for each iteration in a generation.

3.8 TERMINATION CRITERIA

In Genetic Algorithm, it is practically difficult to set a stopping criteria if the exact optimum is unknown or how long the algorithm should run to get the required solution or on which generation the optimum will be found. If the optimum value is known, then the termination rule can be made and the algorithm can be stopped at a generation where it finds the optimum value. This helps the GA to prevent wasting time in searching the whole generations. But, the goal of any GA is to determine the optimum value and so the termination criteria are most probably set based on assumptions and experimentations. It will require to run the program to certain number of times in order to figure out the good termination criteria.

Some GAs follows time depending terminations, in which the algorithm stops if it reached a certain value of time. For example, if the termination criteria are 15 minutes, then the algorithm stops when the time reaches 15 minutes.

In most GAs, the stopping criteria is usually the number of generations, which is assigned during the initial process of the GA. The algorithm will run until the number of defined generation and stops.

Also, the algorithm can be set to terminate if the solution does not improve for over a predefined generation. Some algorithms can also have a termination if the best and worst value from the generation becomes same. If both worst and best is same then there is no need of optimization.

In some GAs, there is a criterion in which it checks the worst and best solution from the generation and if both are same, then the program will be terminated. Since, there is no need for optimization if the best and worst solution in a generation is same, as the population is converged to one solution.

4. EXPERIMENT WITH GENETIC ALGORITHM

This chapter deals with creation active schedule by Giffler and Thompson algorithm and simple genetic algorithm based on the desired parameters and operators that are widely used. An experiment based on the replacement strategy used in genetic algorithm is conducted and a new replacement strategy is proposed and analysed with the simple genetic algorithm.

4.1 GIFFLER & THOMPSON ACTIVE SCHEDULE

The basic structure, limitation and parameters are explained in the section 2.3.1. The exact same approach is followed to generate an active schedule using various dispatching rules. The dispatching rules used in this thesis are as follows:

- SPT – Shortest Processing Time
- LPT – Longest Processing Time
- MTTR – Most Total Time Remaining
- LTTR – Least Total Time Remaining
- MRPT – Most Remaining Processing Time
- LRPT – Least Remaining Processing Time
- FIFO – First In First Out
- LIFO – Last In First Out
- FIFO+SPT – First In First Out + Shortest Processing Time
- LIFO+SPT – Last In First Out + Shortest Processing Time
- MWKR+SPT – Most Work Remaining + Shortest Processing Time
- LWKR+SPT – Least Work Remaining + Shortest Processing Time
- Random rule

The Problem sets used to test the algorithm are Fisher and Thompson (FT06, FT10, FT20) and Lawrence (LA01 to LA25, LA27, LA30, LA40). The results are tabulated in table 1 and the results are then compared with the simple genetic algorithm. From the table 1, the objective function f(x) – makespan varies for each dispatching rule and among those set of rules only one rule gives better results when compared to the other rules. MTTR (Most Total Time Remaining) rule can able to provide the best makespan value for most of the problem than other dispatching rules. Some SPT, LRPT, FIFO+SPT and MWKR+SPT are considered to be the worst as these cannot able to give best makespan value for not even a single problem among the set. The best makespan doesn't mean that it is the optimum, but the minimum that the algorithm can provide based on the dispatching rules.

Table 1. Results of active schedule based on dispatching rules

Problem	Optimum	SPT	LPT	MTTR	LTTR	MRPT	LRPT	FIFO	LIFO	FIFO	LIFO+	MWKR	LWKR	Random	
										+SPT	SPT	+SPT	+SPT	Avg	Best
FT06	**55**	94	86	**58**	92	67	92	84	74	80	82	94	86	75	62
FT10	**930**	1429	1355	**1191**	1498	1203	1449	1235	1450	1255	1274	1429	1355	1348	1201
FT20	**1165**	1675	1661	1594	1658	1604	1653	1520	1462	1569	1699	1675	1661	1602	**1437**
LA01	**666**	1185	927	**784**	1161	**784**	1161	866	865	1101	1004	1185	927	963	829
LA02	**655**	1055	1025	852	1063	879	1063	840	863	937	910	1055	1025	902	**835**
LA03	**597**	795	894	**713**	955	847	955	978	871	715	796	795	894	858	796
LA04	**590**	842	899	817	963	834	828	922	857	866	770	842	899	881	**767**
LA05	**593**	977	720	**621**	936	**621**	936	809	932	808	887	977	720	734	664
LA06	**926**	1439	1142	**930**	1612	985	1612	1169	1156	1515	1116	1439	1142	1092	981
LA07	**890**	1242	1299	**1031**	1290	1103	1265	1093	1213	1120	1220	1242	1299	1151	1061
LA08	**863**	1318	1146	1109	1474	1109	1474	1007	1219	1099	1176	1318	1146	1080	**993**
LA09	**951**	1287	1214	**1021**	1566	1105	1591	1065	1313	1262	1228	1287	1214	1191	1067
LA10	**958**	1604	1129	1052	1561	1030	1412	1280	1407	1438	1330	1604	1129	1117	**1020**
LA11	**1222**	1785	1523	**1274**	1838	1290	1695	1620	1507	1628	1452	1785	1523	1405	1307
LA12	**1039**	1545	1279	1167	1635	1217	1586	1287	1482	1504	1393	1545	1279	1260	**1131**
LA13	**1150**	1595	1420	**1201**	1614	1261	1619	1423	1358	1436	1673	1595	1420	1375	1272
LA14	**1292**	1721	1567	**1292**	2051	**1292**	2035	1443	1519	1787	1674	1721	1567	1428	**1292**
LA15	**1207**	1793	1612	**1415**	1841	1483	1814	1426	1427	1645	1483	1793	1612	1545	1418
LA16	**945**	1464	1336	1219	1416	**1207**	1540	1325	1440	1337	1344	1464	1336	1358	1214
LA17	**784**	1128	1122	**914**	1340	930	1340	1069	1363	966	1262	1128	1122	1168	1028
LA18	**848**	1250	1542	**1039**	1615	1087	1607	1243	1397	1055	1259	1250	1542	1228	1137
LA19	**842**	1286	1120	1123	1309	1244	1260	1144	1309	1278	1277	1286	1120	1166	**1059**
LA20	**902**	1436	1250	**1076**	1437	1230	1437	1150	1248	1289	1171	1436	1250	1256	1147
LA21	**1046**	1657	1560	**1314**	1828	1422	1861	1521	1539	1447	1775	1657	1560	1507	1387
LA22	**927**	1659	1485	**1135**	1834	1203	1780	1423	1597	1602	1515	1659	1485	1412	1269
LA23	**1032**	1786	1521	**1223**	1947	1322	1898	1450	1601	1457	1610	1786	1521	1408	1311
LA24	**935**	1692	1469	1231	1821	**1165**	1640	1505	1499	1396	1383	1692	1469	1390	1200
LA25	**977**	1768	1459	**1206**	1790	1346	1731	1440	1570	1387	1544	1768	1459	1473	1344
LA27	**1235**	2201	1737	**1567**	2282	1709	2119	1783	2026	1781	1970	2201	1737	1786	1632
LA30	**1355**	2194	1871	**1565**	2354	1650	2403	2192	2054	1786	1951	2194	1871	1896	1761
LA40	**1222**	1804	1914	**1549**	2306	1721	2077	1850	1891	1794	1927	1804	1914	1797	1692

The minimum value of the makespan (total completion time) for each problem instance is highlighted in the table with bold and green colour. The MTTR dispatching rule is the best among the 12 dispatching rule used in this paper.

4.2 SIMPLE GENETIC ALGORITHM

The framework of SGA used in this paper for the development is given in the figure 18.

```
•  Initialization of Population
•  Evaluation of Individuals in Population
•  While (Termination criteria not met) do
•  Individual Selection: Select parents (individuals) from Population
•      Crossover      : Perform crossover for the selected parents
•      Mutation       : Perform mutation for children produced by crossover
•      Evaluation     : Evaluate the children after Mutation
•      Replacement    : Replace the parents in Population with best children
•  Go to step 4 until Termination criteria met
•  end while
•  return the best individual found during the generations.
```

Figure 17. SGA pseudocode [40]

The algorithm is created to test two type of chromosome representation and their influence in the results of optimization. Operation order based representation and Job order based representation are used in SGAs. For Selection, Roulette wheel selection (RWS) is used for the selection of parents for reproduction process. For both representations, Job order crossover and Insert mutation operator is employed. This SGA is based on the generational replacement, in which best children are replaced in each generation. The replacement is carried out either by replacing some of the worst individuals in the parent population or the whole population is replaced. Here, the population of individual is replaced with best children with half the size of the Population.

4.2.1 SGA parameters value

Population: The number of individuals in the population (Pop_num) is defined with 2N, where N is the number of operations. So, the population will be 2 times the number of operations.

Maximum generation: 1500

Chromosome representation: Job order based representation and Operation based representation is used to define the individuals (chromosomes) in the population.

Fitness function: Giffler & Thompson algorithm is used to calculate the fitness function and repair the infeasible solutions.

Selection: Roulette Wheel Selection is used to select the individuals in to the mating pool for both type of representation The size of mating pool is half the size of population (Pop_num/2).

Crossover: Job Order Crossover (JOX) is used for both type of chromosome representations with crossover probability (Pxo) 0.5. So, half of the genes in the individual are subjected to the crossover operation.

Mutation: Shift Mutation is used for both type of chromosome and probability is not used and only one gene is subjected to mutation and every individual are subjected to mutation operation.

Replacement: Generational replacement

Termination: Algorithm terminates if there is no improvement in the solution for a count up to 200.

4.2.2 Results of SGA with job order & operation based representation

The table 2 shows the results of the Simple Genetic Algorithm (SGA) based on job order representation and the table 3 shows the results obtained from the SGA based on the operation based representation. Both SGAs gives the optimum makespan value (objective function) for some problem instance (bold value at table 2 and 3). When comparing both the table 2 and table 3, SGA based on the job order representation is better in producing the best makespan value for the Lawrence problem instances and SGA with operation based representation is better in producing best makespan value for the Fisher and Thompson problem instances.

Table 2. Result of SGA_{JO}

Problem	**Optimum**	**Best f(x)**	**Avg. f(x)**	**Worst f(x)**	**Deviation**	**Avg. Dist. from optimum**	**Avg. Opt time in sec**	**Avg. gen taken to find**
FT06	**55**	**55**	55	66	0.00	0.00%	17.2	236
FT10	**930**	1042	1054.5	1230	35.42	13.39%	141.71	625
FT20	**1165**	1315	1337.6	1529	47.43	14.82%	154.45	762
LA01	**666**	**666**	666.2	811	0.00	0.03%	35.21	609
LA02	**655**	668	693.8	839	4.11	5.92%	35.25	863
LA03	**597**	637	646.5	802	12.65	8.29%	35.23	759
LA04	**590**	598	614.5	763	2.53	4.15%	36.86	951
LA05	**593**	**593**	593	639	0.00	0.00%	36.85	6
LA06	**926**	**926**	926	1012	0.00	0.00%	85.35	21
LA07	**890**	915	925.3	1066	7.91	3.97%	84.2	601
LA08	**863**	**863**	867.3	1004	0.00	0.50%	86.28	867
LA09	**951**	**951**	951	1052	0.00	0.00%	86.38	43
LA10	**958**	**958**	958	1040	0.00	0.00%	87.26	9
LA11	**1222**	**1222**	1222	1306	0.00	0.00%	161.31	28
LA12	**1039**	**1039**	1039	1140	0.00	0.00%	161.96	65
LA13	**1150**	1150	1150	1239	0.00	0.00%	156.76	210
LA14	**1292**	1292	1292	1305	0.00	0.00%	21.31	1
LA15	**1207**	1269	1276.8	1432	19.61	5.78%	156.66	702
LA16	**945**	985	998.5	1156	12.65	5.66%	127.46	1028
LA17	**784**	803	815.9	957	6.01	4.07%	127.35	930.8
LA18	**848**	888	907.7	1052	12.65	7.04%	127.38	473
LA19	**842**	869	888.3	1031	8.54	5.50%	129.58	1104
LA20	**902**	933	953.7	1160	9.80	5.73%	124.14	895
LA21	**1046**	1177	1201.2	1356	41.43	14.04%	318.39	551
LA22	**927**	1073	1087.6	1265	46.17	17.32%	307.54	679
LA23	**1032**	1079	1098.5	1282	14.86	6.44%	307.72	796
LA24	**935**	1059	1066.7	1236	39.21	14.09%	309.84	699
LA25	**977**	1105	1125.2	1277	40.48	15.17%	313.73	798
LA27	**1235**	1439	1451.7	1607	64.51	17.55%	603.55	695
LA30	**1355**	1456	1502.9	1674	31.94	10.92%	599.9	1043
LA40	**1222**	1374	1381.1	1566	48.07	13.02%	672.64	813

Table 3. Result of SGA_{OB}

Problem	Optimum	Best f(x)	Avg. f(x)	Worst f(x)	Deviation	Avg. Dist. from optimum	Avg. Opt time in sec	Avg. gen taken to find
FT06	**55**	**55**	55.6	68	0.00	1.09%	19.38	77
FT10	**930**	1038	1051.1	1230	34.15	13.02%	179.47	665
FT20	**1165**	1303	1319.5	1500	43.64	13.26%	180.01	1110
LA01	**666**	**666**	671	821	0.00	0.75%	41.8	444
LA02	**655**	684	703.3	865	9.17	7.37%	39.18	843.3
LA03	**597**	625	640.4	800	8.85	7.27%	39.09	962.5
LA04	**590**	603	622.3	764	4.11	5.47%	39.91	538
LA05	**593**	**593**	593	639	0.00	0.00%	41.01	2
LA06	**926**	**926**	926	1015	0.00	0.00%	99.41	7
LA07	**890**	894	916.4	1068	1.26	2.97%	95.705	751
LA08	**863**	**863**	867.1	1004	0.00	0.48%	99.15	485
LA09	**951**	**951**	951	1061	0.00	0.00%	98.65	37
LA10	**958**	**958**	958	1038	0.00	0.00%	98.575	3
LA11	**1222**	**1222**	1222	1298	0.00	0.00%	187.11	16
LA12	**1039**	**1039**	1039	1136	0.00	0.00%	185.28	27
LA13	**1150**	1150	1150	1241	0.00	0.00%	199.48	100
LA14	**1292**	1292	1292	1293	0.00	0.00%	29.89	1
LA15	**1207**	1244	1256.4	1433	11.70	4.09%	195.45	868
LA16	**945**	979	1008.3	1164	10.75	6.70%	185.94	766
LA17	**784**	812	831	961	8.85	5.99%	176.74	552
LA18	**848**	885	910.5	1061	11.70	7.37%	189.15	627
LA19	**842**	882	900.6	1041	12.65	6.96%	173.85	661
LA20	**902**	959	968	1147	18.02	7.32%	188.05	685
LA21	**1046**	1197	1214.7	1366	47.75	16.13%	475.311	752
LA22	**927**	1065	1084.3	1265	43.64	16.97%	463.62	988
LA23	**1032**	1106	1123.9	1278	23.40	8.91%	445.74	1084
LA24	**935**	1074	1089.9	1242	43.96	16.57%	454.56	1023
LA25	**977**	1118	1133.8	1290	44.59	16.05%	451.13	907
LA27	**1235**	1441	1452.9	1621	65.14	17.64%	856.14	804
LA30	**1355**	1470	1504.7	1677	36.37	11.05%	852.72	853
LA40	**1222**	1385	1407.8	1575	51.55	15.20%	1087.82	929

4.3 PROPOSED STRATEGY

The problem with the evolution algorithm is that there is a chance of solution to get stuck in the local optimum, in order to preserve diversity, various methods are formulated by the researchers. This process of preserving the diversity is highly depending on the replacement method used in the algorithm.

4.3.1 Motivation and basic idea of proposed algorithm

Most of the GAs are dealt with either generational or steady state replacements strategies. In steady state replacement the children formed by crossover is evaluated and replaced immediately in to the parent population. Likewise, after the application mutation, the technique of evaluating the children and replacing in the population is done. The iteration is carried out after every individual in the population are subjected to crossover and mutation. Whereas in generational replacements, the individuals are subjected to crossover and mutation and evaluated after every individual reproduced. After the evaluation, the children population replace the parent population completely. This concept is combining both the steady state and generational replacement in order to infuse a new replacement strategy. The concept is that, evaluating each child right after its reproduction in each genetic operation (crossover and mutation). The best individual is carried out to the generational replacement after the reproduction of every individual in the population. The generational replacement is carried out along with the elitist strategy to preserve the best individual from each generation.

There is also a concept of using the individuals that are not selected during the evaluation. The rejected individuals are used to introduce the diversity to the population based on criteria of uniqueness. A parameter will be used to measure the uniqueness of each individual in the population in other words, it will find the individuals with the same fitness value. The introduction of individuals from the rejected population to the population is performed if the uniqueness value reduces below a defined threshold value. This concept is applied to the SGA for both job order based and operation with job repetition representations.

The SGA is modified to implement the combined GSGA (Generational Simple Genetic Algorithm) and SSGA (Steady State Genetic Algorithm) reusable replacement strategy of replacement and the modification is carried out in function of crossover, function of mutation and in the replacement function. The evaluation of the individuals is done in each function in order to evaluate and sort the individuals for the next step and the replacement is done as a final step in each generation after evaluating the necessary parameters. The evaluation and selected right after crossover and mutation is inspired from

SSGA and replacing procedure is inspired rom GSGA. The individuals will not be replaced immediately in the population like SSGA, only the evaluation and selection is done based on it.

4.3.2 Parameters in Reusable Replacement Strategy (RRS)

Some parameters have been set and used to implement this strategy and the parameters are explained with its function.

i. Rejected population

This is a set of individuals that are not succeeded in the selection process after the reproduction process of crossover and mutation. This is a simple concept of utilizing the individuals that are close to the best individuals and giving a second chance to evolve better. It can eliminate the introduction of new individuals which probably reflect the same fitness values and may be better but this concept motivates to recycle the available resources than to create new individuals. The individuals in rejected population are sorted in ascending order before subjected to the replacement process to ensure the best from rejected are used in place of similar individuals in the existing population. The size of the rejected population is same as the size of the population of individuals after crossover and mutation. In this strategy the size of rejected population will be double the size of mating pool population and this size is reduced by sorting the unique individuals and with respect to the best fitness value. By this process the computational space and memory is save to improve the performance of the algorithm.

ii. Uniqueness parameter

This parameter is used to verify whether the individuals in the population becomes similar. A desired value is set (ideally 20% of the total population) and there should be at least 20% of unique individuals in the population. For example, if we have population of 100 and its 20% percentage is 20, if the individuals with same fitness value that exceed the count 20 then 19 individuals from the 20 is get replaced with the individuals from the rejected population. Leaving one individual helps to keep the individual which may have the best fitness. Having a population filled with unique individuals is not possible and it will not allow the GA to converge. It works as, if the number of similar individuals increases and exceeds the uniqueness threshold value, then the replacement of similar individuals with the individuals from the rejected population is performed.

iii. Unique count

Unique count calculates the number of unique individuals and evaluates each unique individual with the uniqueness parameter and determines whether to subject the non-unique individuals for the replacement process or proceed with the simple generational replacement with elitism. It counts the number of similar individuals, considering the above example with population 100, if the count exceeds 20 new replacement strategy occurs else the generational replacement will occur. This parameter will simply count individuals based on uniqueness parameter.

iv. Number of generational replacement

A parameter used to define the number of individuals is replaced in the population after each generation to proceed to the next generation. In this strategy the generational replacement is combined with elitist replacement. So the number of generational replacement is set to 80% of the total population. With this definition, 20% of individuals will not be replaced in every generation and favours the population to keep the best individuals found throughout the generations. The replacement is carried out if the unique count does not exceed the defined uniqueness parameter.

4.3.3 Proposed strategy structure in SGA

The Reusable replacement strategy proposed in this paper and its parameters are defined and the structure of the replacement is given in the figure 18.

```
•  Initialization of Population
•  Evaluation of Individuals in Population
•  While (Termination criteria not met) do
•      Individual Selection   : Select parents (individuals) from Population
•      Crossover     : Perform crossover for the selected parents
•      Evaluation    : Evaluate the children after crossover
•      Selection     : Two required individuals selected to proceed and two
                         rejected individuals are saved for future replacement
•      Mutation      : Perform mutation for the children produced by
                       crossover
•      Selection     : One required individuals selected to proceed and One
                         rejected individual is saved for future replacement
•      Replacement   :
                  If Unique count > Uniqueness Parameter
                       Replace the individuals with individuals from rejected
                       population after sorting
                  Else
                       Replace the individuals with best children formed by
                       reproduction based on number of replacement count
                  End If
•  Go to step 4 if Termination criteria not met
•  end while
•  return the best individual found during the generations
```

Figure 18. Structure SGA Reusable replacement strategy [42], [source: own]

4.4 PARAMETER SET VALUE & RESULTS

The value defined for each parameter use in the reusable replacement strategy within the SGA is given as follows

4.4.1 MSGA parameters value

Population: The number of individuals in the population (Pop_num) is defined with 2N, where N is the number of operations. So, the population will be 2 times the number of operations.

Maximum Generation: 1500

Chromosome representation: Job order based representation and Operation based representation is used to define the individuals (chromosomes) in the population.

Fitness function: Giffler & Thompson algorithm is used to calculate the fitness function and repair the infeasible solutions.

Selection: Roulette Wheel Selection is used to select the individuals in to the mating pool for both type of representation The size of mating pool is half the size of population (Pop_num/2).

Crossover: Job Order Crossover (JOX) is used for both type of chromosome representations with crossover probability (Pxo) 0.5. So, half of the genes in the individual are subjected to the crossover operation. Correlative family based selection is used to selected the individuals to proceed.

Mutation: Shift Mutation is used for both type of chromosome and probability is not used and only one gene is subjected to mutation and every individual are subjected to mutation operation. Best individual is selected to proceed the process.

Replacement: Reusable replacement strategy.

Termination: Algorithm terminates if there is no improvement in the solution for a count up to 200. If the solution doesn't improve for 200 generations, it will be terminated or else, it will be terminated at the end of maximum number of generations.

4.4.2 Replacement parameters

Uniqueness Parameter (Par_Unique) = 0.2.

Unique Count (Uniq_count) = Par_Unique*Pop_num.

Rejected Population (Rej_Pop) **=** Pop_num.

4.4.3 Result of Modified SGA using job order based representation (MSGA$_{JO}$)

The Table 4. shows the results obtained by the implementation of Reusable Replacement Strategy (RRS) in the simple genetic algorithm based on job order representation. The result shows that the replacement strategy improves the solution as compared to SGA. The best makespan values (objective functions) are highlighted with bold and green colour. The ability to produce the optimum results improved with this strategy.

Table 4. Result of MSGA$_{JO}$

Problem	Optimum	Best f(x)	Avg. f(x)	Worst f(x)	Deviation	Avg. Dist. from optimum	Avg. Opt time in sec	Avg. gen taken to find
FT06	**55**	**55**	55	63	0.00	0.00%	13.09	25
FT10	**930**	951	972.1	1175	6.64	4.53%	233.72	978
FT20	**1165**	1207	1425	1233.8	13.28	5.91%	213.37	295
LA01	**666**	**666**	666	793	0.00	0.00%	52.09	1376
LA02	**655**	**655**	665.1	786	0.00	1.54%	68.41	598
LA03	**597**	**597**	605.5	745	0.00	1.42%	64	711
LA04	**590**	**590**	594	709	0.00	0.68%	64.81	453
LA05	**593**	**593**	593	627	0.00	0.00%	19.88	4
LA06	**926**	**926**	926	987	0.00	0.00%	56.36	5
LA07	**890**	**890**	890	1019	0.00	0.00%	126.74	323
LA08	**863**	**863**	863	961	0.00	0.00%	143.04	19
LA09	**951**	**951**	951	1043	0.00	0.00%	101.37	12
LA10	**958**	**958**	958	1004	0.00	0.00%	30.36	3
LA11	**1222**	**1222**	1222	1278	0.00	0.00%	93.41	7
LA12	**1039**	**1039**	1039	1115	0.00	0.00%	88.03	8
LA13	**1150**	**1150**	1150	1220	0.00	0.00%	125.06	13
LA14	**1292**	**1292**	1292	1298	0.00	0.00%	37.35	1
LA15	**1207**	**1207**	1210.7	1382	0.00	0.31%	145.41	290
LA16	**945**	977	984.5	1123	10.12	4.18%	133.85	247
LA17	**784**	**784**	793.4	928	0.00	1.20%	104.24	162
LA18	**848**	**848**	861.8	1026	0.00	1.63%	151.76	189
LA19	**842**	852	863.1	1007	3.16	2.51%	186.5	360
LA20	**902**	907	915.9	1105	1.58	1.54%	156.01	231
LA21	**1046**	1082	1101.8	1326	11.38	5.33%	397.3	521
LA22	**927**	953	987.4	1244	8.22	6.52%	309.42	244
LA23	**1032**	**1032**	1039.3	1263	0.00	0.71%	261.5	192
LA24	**935**	971	989.4	1218	11.38	5.82%	286.94	369
LA25	**977**	1012	1034.2	1255	11.07	5.85%	372.07	287
LA27	**1235**	1282	1315.7	1585	14.86	6.53%	796.05	535
LA30	**1355**	1383	1400.2	1648	8.85	3.34%	530.08	443
LA40	**1222**	1255	1281.8	1543	10.44	4.89%	693.14	507

Even the most demanding problems like LA27, LA30 and LA40 can able to optimize to a solution better than the SGA. The average distance from optimum shows the repeatability of algorithm to find the best solution and except few high demanding problems, all other problems results 0% which is a good sign of improvement. Also, almost all expensive problem set reflects less than 5% of average distance from optimum.

4.4.4 Result of Modified SGA using operation based representation ($MSGA_{OB}$)

Table 5. Result of $MSGA_{OB}$

Problem	Optimum	Best f(x)	Avg. f(x)	Worst f(x)	Deviation	Avg. Dist. from optimum	Avg. Opt time in sec	Avg. gen taken to find
FT06	**55**	**55**	55	61	0.00	0.00%	26.88	56
FT10	**930**	969	995.9	1163	12.33	7.09%	339.21	790
FT20	**1165**	1237	1257.4	1447	22.77	7.93%	362.37	963
LA01	**666**	**666**	666	777	0.00	0.00%	89.52	153
LA02	**655**	**655**	672	782	0.00	2.60%	86.66	783
LA03	**597**	606	618.7	708	2.85	3.63%	83.73	711
LA04	**590**	**590**	601.9	702	0.00	2.02%	88.24	639
LA05	**593**	**593**	593	605	0.00	0.00%	57.49	2
LA06	**926**	**926**	926	959	0.00	0.00%	198.75	4
LA07	**890**	**890**	890.8	990	0.00	0.09%	201.48	558
LA08	**863**	**863**	863	930	0.00	0.00%	209.74	71
LA09	**951**	**951**	951	1001	0.00	0.00%	211.12	7
LA10	**958**	**958**	958	972	0.00	0.00%	113.91	2
LA11	**1222**	**1222**	1222	1244	0.00	0.00%	350.32	4
LA12	**1039**	**1039**	1039	1071	0.00	0.00%	339.86	6
LA13	**1150**	**1150**	1150	1193	0.00	0.00%	354.32	11
LA14	**1292**	**1292**	1292	1292	0.00	0.00%	49.71	1
LA15	**1207**	**1207**	1215.2	1349	0.00	0.68%	679.73	778
LA16	**945**	973	987.4	1102	8.85	4.49%	341.81	791
LA17	**784**	787	802.1	908	0.95	2.31%	323.02	739
LA18	**848**	857	864.9	1014	2.85	1.99%	315.79	881
LA19	**842**	856	877	1002	4.43	4.16%	336.8	794
LA20	**902**	912	931	1063	3.16	3.22%	518.29	557
LA21	**1046**	1141	1164	1326	30.04	11.28%	875.2	912
LA22	**927**	1002	1032.9	1205	23.72	11.42%	832.27	1175
LA23	**1032**	1046	1091.3	1235	4.43	5.75%	851.64	916
LA24	**935**	1017	1035.2	1183	25.93	10.72%	834.98	813
LA25	**977**	1058	1088.8	1230	25.61	11.44%	837.05	834
LA27	**1235**	1388	1405	1559	48.38	13.77%	1647.03	985
LA30	**1355**	1459	1469.1	1611	32.89	8.42%	1649.3	890
LA40	**1222**	1334	1368.7	1509	35.42	12.00%	1989.51	1105

The table 5 displays the result obtained from the Modified SGA with reusable replacement strategy based on the operation based representation ($MSGA_{OB}$). The best makespan values (objective functions) are highlighted with bold and green colour. The results show that the operation based representation has some influence that affects the solution of the SGA. The average distance from optimum of some hard problems are above 10% which is not good as the SGA with job order representation. Modifying some parameters and implementing some criteria to improve the solution might help but from the result, it might or might not improve the solution. Also, the best solution is found quickly for some problems and took nearly thousand generations to find near optimum solution for expensive problems like LA20, LA30, LA40.

4.4.5 Comparison of Modified SGA of job order (MSGA$_{JO}$) and operation based representation (MSGA$_{OB}$)

Table 6. Comparison of MSGA$_{JO}$ and MSGA$_{OB}$

Problem	Optimum	MSGA$_{JO}$			MSGA$_{OB}$		
		Best f(x)	Avg. f(x)	Avg. distance from optimum	Best f(x)	Avg. f(x)	Avg. distance from optimum
FT06	55	55	55	0.00%	55	55	0.00%
FT10	930	951	972.1	4.53%	969	995.9	7.09%
FT20	1165	1207	1233.8	5.91%	1237	1257.4	7.93%
LA01	666	666	666	0.00%	666	666	0.00%
LA02	655	655	665.1	1.54%	655	672	2.60%
LA03	597	597	605.5	1.42%	606	618.7	3.63%
LA04	590	590	594	0.68%	590	601.9	2.02%
LA05	593	593	593	0.00%	593	593	0.00%
LA06	926	926	926	0.00%	926	926	0.00%
LA07	890	890	890	0.00%	890	890.8	0.09%
LA08	863	863	863	0.00%	863	863	0.00%
LA09	951	951	951	0.00%	951	951	0.00%
LA10	958	958	958	0.00%	958	958	0.00%
LA11	1222	1222	1222	0.00%	1222	1222	0.00%
LA12	1039	1039	1039	0.00%	1039	1039	0.00%
LA13	1150	1150	1150	0.00%	1150	1150	0.00%
LA14	1292	1292	1292	0.00%	1292	1292	0.00%
LA15	1207	1207	1210.7	0.31%	1207	1215.2	0.68%
LA16	945	977	984.5	4.18%	973	987.4	4.49%
LA17	784	784	793.4	1.20%	787	802.1	2.31%
LA18	848	848	861.8	1.63%	857	864.9	1.99%
LA19	842	852	863.1	2.51%	856	877	4.16%
LA20	902	907	915.9	1.54%	912	931	3.22%
LA21	1046	1082	1101.8	5.33%	1141	1164	11.28%
LA22	927	953	987.4	6.52%	1002	1032.9	11.42%
LA23	1032	1032	1039.3	0.71%	1046	1091.3	5.75%
LA24	935	971	989.4	5.82%	1017	1035.2	10.72%
LA25	977	1012	1034.2	5.85%	1058	1088.8	11.44%
LA27	1235	1282	1315.7	6.53%	1388	1405	13.77%
LA30	1355	1383	1400.2	3.34%	1459	1469.1	8.42%
LA40	1222	1255	1281.8	4.89%	1334	1368.7	12.00%

From the table 6 the comparison of the implementation of Reusable Replacement Strategy (RRS) in the SGA of both job order and operation based representations (MSGA$_{JO}$

and MSGA$_{OB}$). The best makespan values (objective functions) are highlighted with bold and green colour and also best average and average distance from makespan values are highlighted in bold. The results of best solution in MSGA$_{JO}$ is better than the best solution in MSGA$_{OB}$. The average distance from the optimum value of MSGA$_{JO}$ is good and better than the MSGA$_{OB}$. So, MSGA$_{JO}$ seems to be a better option when compared to MSGA$_{OB}$. The comparison of the average distance from the optimum solution of the objective function – makespan between the MSGA$_{JO}$ and MSGA$_{OB}$ is shown in the figure 19.

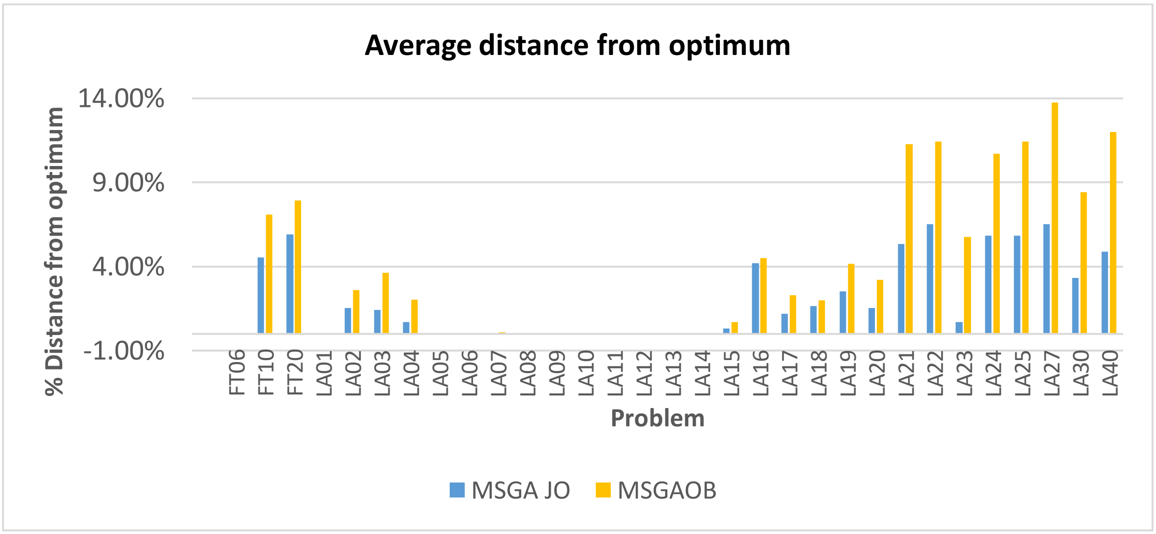

Figure 19. Comparison graph of MSGA$_{JO}$ and MSGA$_{OB}$ [source: own]

4.4.6 Comparison of SGA and MSGA with job order based representation

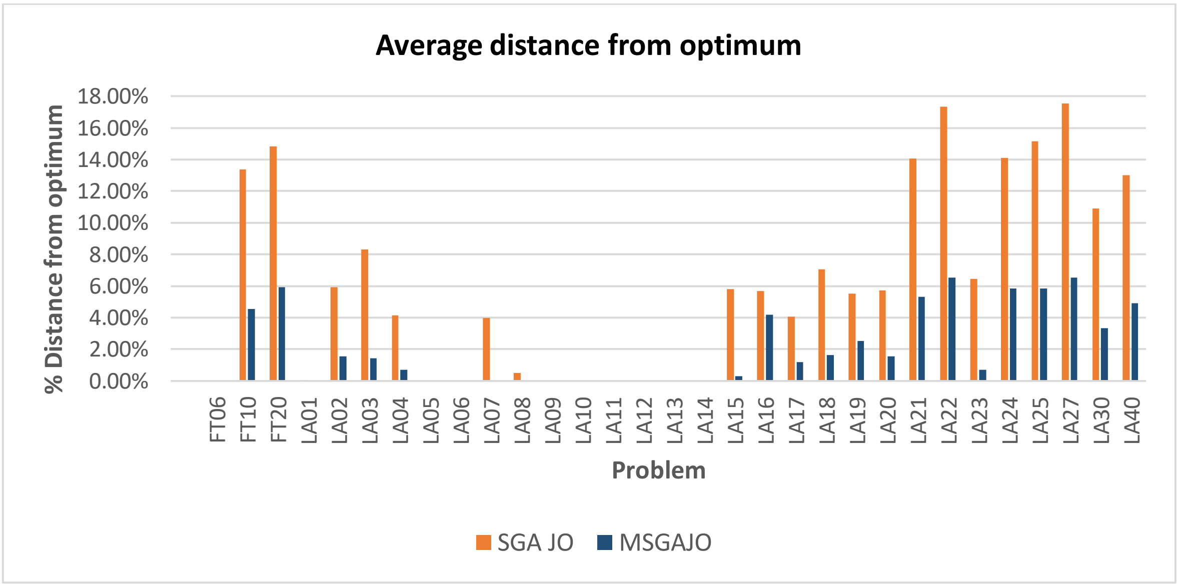

Figure 20. Comparison graph of SGA$_{JO}$ and MSGA$_{JO}$ [source: own]

Table 7. Comparison of SGA$_{JO}$ and MSGA$_{JO}$

Problem	Optimum	SGA$_{JO}$			MSGA$_{JO}$		
		Best f(x)	Avg. f(x)	Avg. distance from optimum	Best f(x)	Avg. f(x)	Avg. distance from optimum
FT06	**55**	**55**	**55**	**0.00%**	**55**	**55**	**0.00%**
FT10	**930**	1042	1054.5	13.39%	**951**	**972.1**	**4.53%**
FT20	**1165**	1315	1337.6	14.82%	**1207**	**1233.8**	**5.91%**
LA01	**666**	**666**	666.2	0.03%	**666**	**666**	**0.00%**
LA02	**655**	668	693.8	5.92%	**655**	**665.1**	**1.54%**
LA03	**597**	637	646.5	8.29%	**597**	**605.5**	**1.42%**
LA04	**590**	598	614.5	4.15%	**590**	**594**	**0.68%**
LA05	**593**	**593**	**593**	**0.00%**	**593**	**593**	**0.00%**
LA06	**926**	**926**	**926**	**0.00%**	**926**	**926**	**0.00%**
LA07	**890**	**915**	925.3	3.97%	**890**	**890**	**0.00%**
LA08	**863**	**863**	867.3	0.50%	**863**	**863**	**0.00%**
LA09	**951**	**951**	**951**	**0.00%**	**951**	**951**	**0.00%**
LA10	**958**	**958**	**958**	**0.00%**	**958**	**958**	**0.00%**
LA11	**1222**	**1222**	**1222**	**0.00%**	**1222**	**1222**	**0.00%**
LA12	**1039**	**1039**	**1039**	**0.00%**	**1039**	**1039**	**0.00%**
LA13	**1150**	**1150**	**1150**	**0.00%**	**1150**	**1150**	**0.00%**
LA14	**1292**	**1292**	**1292**	**0.00%**	**1292**	**1292**	**0.00%**
LA15	**1207**	1269	1276.8	5.78%	**1207**	**1210.7**	**0.31%**
LA16	**945**	985	998.5	5.66%	**977**	**984.5**	**4.18%**
LA17	**784**	803	815.9	4.07%	**784**	**793.4**	**1.20%**
LA18	**848**	888	907.7	7.04%	**848**	**861.8**	**1.63%**
LA19	**842**	869	888.3	5.50%	**852**	**863.1**	**2.51%**
LA20	**902**	933	953.7	5.73%	**907**	**915.9**	**1.54%**
LA21	**1046**	1177	1201.2	14.04%	**1082**	**1101.8**	**5.33%**
LA22	**927**	1073	1087.6	17.32%	**953**	**987.4**	**6.52%**
LA23	**1032**	1079	1098.5	6.44%	**1032**	**1039.3**	**0.71%**
LA24	**935**	1059	1066.7	14.09%	**971**	**989.4**	**5.82%**
LA25	**977**	1105	1125.2	15.17%	**1012**	**1034.2**	**5.85%**
LA27	**1235**	1439	1451.7	17.55%	**1282**	**1315.7**	**6.53%**
LA30	**1355**	1456	1502.9	10.92%	**1383**	**1400.2**	**3.34%**
LA40	**1222**	1374	1381.1	13.02%	**1255**	**1281.8**	**4.89%**

The comparison data from table 7, clearly depicts that the Modified SGA (MSGA$_{JO}$) with Reusable Replacement Strategy (RRS) outruns the result of SGA$_{JO}$. The best, average and average distance from the makespan values (objective function) of MSGA$_{JO}$ is far better than the SGA$_{JO}$. The generational replacement used in the SGA$_{JO}$, can produce best makespan values (objective functions) are highlighted with bold and green colour and also best average and average distance from makespan values are highlighted in bold in table 7. The MSGA$_{JO}$ with the Reusable Replacement Strategy has results improved in every aspects of the problem. The comparison of the average distance from the optimum solution of the objective function – makespan between the SGA_{JO} and $MSGA_{JO}$ is shown in the figure 20.

4.4.7 Comparison of SGA and MSGA with operation based representation

Table 8. Comparison of SGA_{OB} and $MSGA_{OB}$

Problem	Optimum	SGA_{OB}			$MSGA_{OB}$		
		Best f(x)	Avg. f(x)	Avg. distance from optimum	Best f(x)	Avg. f(x)	Avg. distance from optimum
FT06	55	55	55.6	1.09%	55	55	0.00%
FT10	930	1038	1051.1	13.02%	969	995.9	7.09%
FT20	1165	1303	1319.5	13.26%	1237	1257.4	7.93%
LA01	666	666	671	0.75%	666	666	0.00%
LA02	655	684	703.3	7.37%	655	672	2.60%
LA03	597	625	640.4	7.27%	606	618.7	3.63%
LA04	590	603	622.3	5.47%	590	601.9	2.02%
LA05	593	593	593	0.00%	593	593	0.00%
LA06	926	926	926	0.00%	926	926	0.00%
LA07	890	894	916.4	2.97%	890	890.8	0.09%
LA08	863	863	867.1	0.48%	863	863	0.00%
LA09	951	951	951	0.00%	951	951	0.00%
LA10	958	958	958	0.00%	958	958	0.00%
LA11	1222	1222	1222	0.00%	1222	1222	0.00%
LA12	1039	1039	1039	0.00%	1039	1039	0.00%
LA13	1150	1150	1150	0.00%	1150	1150	0.00%
LA14	1292	1292	1292	0.00%	1292	1292	0.00%
LA15	1207	1244	1256.4	4.09%	1207	1215.2	0.68%
LA16	945	979	1008.3	6.70%	973	987.4	4.49%
LA17	784	812	831	5.99%	787	802.1	2.31%
LA18	848	885	910.5	7.37%	857	864.9	1.99%
LA19	842	882	900.6	6.96%	856	877	4.16%
LA20	902	959	968	7.32%	912	931	3.22%
LA21	1046	1197	1214.7	16.13%	1141	1164	11.28%
LA22	927	1065	1084.3	16.97%	1002	1032.9	11.42%
LA23	1032	1106	1123.9	8.91%	1046	1091.3	5.75%
LA24	935	1074	1089.9	16.57%	1017	1035.2	10.72%
LA25	977	1118	1133.8	16.05%	1058	1088.8	11.44%
LA27	1235	1441	1452.9	17.64%	1388	1405	13.77%
LA30	1355	1470	1504.7	11.05%	1459	1469.1	8.42%
LA40	1222	1385	1407.8	15.20%	1334	1368.7	12.00%

The table 8 shows, how the RRS influence the performance of the SGA and all the best, worst and average results of $MSGA_{OB}$ is better than the SGA_{OB}. This shows that the Reusable Replacement Strategy has the ability to optimize the SGA efficiently. The best makespan values (objective functions) are highlighted with bold and green colour and also best average and average distance from makespan values are highlighted in bold. The comparison of the average distance from the optimum solution of the objective function – makespan between the SGA_{OB} and $MSGA_{OB}$ is shown in the figure 21.

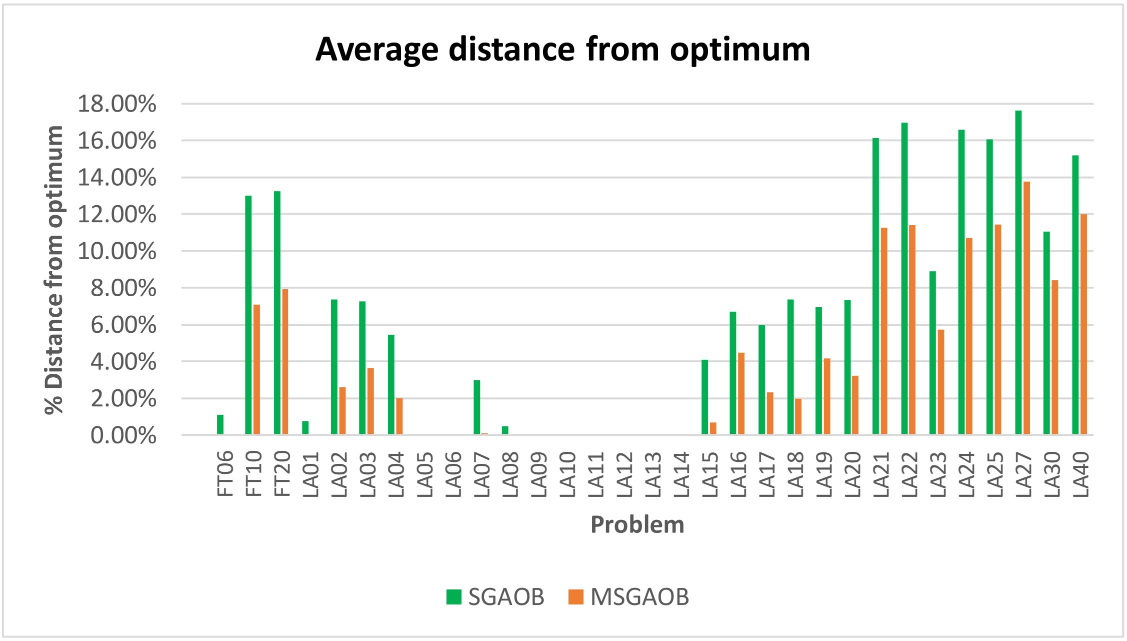

Figure 21. Comparison graph of SGA_{OB} and $MSGA_{OB}$ [source: own]

4.4.8 Comparison of all SGAs with active schedule based on best dispatching rule

The table 9 shows the comparison of the results obtained by both SGAs and MSGAs with the results of active schedule based on dispatching rules with best objective function (makespan). $MSGA_{JO}$ turns out to be the best algorithm to determine the optimum or the best makespan.

The schedules generated with the help of dispatching rules are worst when compared to the SGAs but there are no options of evolution in the active schedules. The dispatching rules are better in case of optimization time. They will require only a fraction of seconds to generate a schedule and SGAs will take more time with respect to the problem size. This comparison of SGAs, MSGAs and active schedule gives a result that the $MSGA_{JO}$ is again the best algorithm to provide the best makespan values for almost every problem dealt in this paper. The best makespan values (objective functions) are highlighted with bold and green colour.

Table 9. Comparison SGAs with active schedule based on MTTR

Problem	Optimum	SGA_{JO}	SGA_{OB}	$MSGA_{JO}$	$MSGA_{OB}$	Active Schedule with MTTR
FT06	55	**55**	**55**	**55**	**55**	58
FT10	930	1042	1038	**951**	969	1191
FT20	1165	1315	1303	**1207**	1237	1594
LA01	666	**666**	**666**	**666**	**666**	784
LA02	655	668	684	**655**	**655**	852
LA03	597	637	625	**597**	606	713
LA04	590	598	603	**590**	**590**	817
LA05	593	**593**	**593**	**593**	**593**	621
LA06	926	**926**	**926**	**926**	**926**	930
LA07	890	915	894	**890**	**890**	1031
LA08	863	**863**	**863**	**863**	**863**	1109
LA09	951	**951**	**951**	**951**	**951**	1021
LA10	958	**958**	**958**	**958**	**958**	1052
LA11	1222	**1222**	**1222**	**1222**	**1222**	1274
LA12	1039	**1039**	**1039**	**1039**	**1039**	1167
LA13	1150	**1150**	**1150**	**1150**	**1150**	1201
LA14	1292	**1292**	**1292**	**1292**	**1292**	1292
LA15	1207	1269	1244	**1207**	**1207**	1415
LA16	945	985	979	977	**973**	1219
LA17	784	803	812	**784**	787	914
LA18	848	888	885	**848**	857	1039
LA19	842	869	882	**852**	856	1123
LA20	902	933	959	**907**	912	1076
LA21	1046	1177	1197	**1082**	1141	1314
LA22	927	1073	1065	**953**	1002	1135
LA23	1032	1079	1106	**1032**	1046	1223
LA24	935	1059	1074	**971**	1017	1231
LA25	977	1105	1118	**1012**	1058	1206
LA27	1235	1439	1441	**1282**	1388	1567
LA30	1355	1456	1470	**1383**	1459	1565
LA40	1222	1374	1385	**1255**	1334	1549

4.4.9 Comparison of optimization time per generation of SGAs and MSGAs

The table 10 shows the time taken by each SGA to optimize to the best solution (objective function – makespan). The influence of the representation can be seen in the time taken by the SGA and MSGA as the SGA and MSGA with Job order representation has the best optimization time than the SGA and MSGA with operation based representation. However, based on overall overview, SGA_{JO} has the most number of best optimization time and $MSGA_{JO}$ has good number of best optimization time next to SGA_{JO}. The figure 22 shows the results graphically and from that the SGA_{JO} has most number of best optimization time and the SGA with worst number of optimization time is $MSGA_{OB}$. The best optimization time for each problem instances are highlighted in bold.

Table 10. Comparison of optimization time per generation

Problem	SGA_{JO} Opt time in sec	SGA_{OB} Opt time in sec	$MSGA_{JO}$ Opt time in sec	$MSGA_{OB}$ Opt time in sec
FT06	**0.039**	0.070	0.058	0.105
FT10	**0.172**	0.207	0.306	0.343
FT20	0.161	**0.137**	0.714	0.312
LA01	**0.044**	0.065	0.233	0.254
LA02	**0.033**	0.038	0.081	0.088
LA03	0.037	**0.034**	0.070	0.092
LA04	**0.032**	0.054	0.099	0.105
LA05	0.179	0.203	**0.098**	0.285
LA06	0.386	0.481	**0.275**	0.978
LA07	0.105	**0.101**	0.242	0.266
LA08	**0.081**	0.145	0.655	0.775
LA09	**0.355**	0.417	0.479	1.020
LA10	0.419	0.487	**0.150**	0.566
LA11	0.709	0.867	**0.452**	1.722
LA12	0.611	0.817	**0.424**	1.653
LA13	**0.382**	0.665	0.589	1.679
LA14	**0.106**	0.149	0.186	0.247
LA15	**0.174**	0.183	0.297	0.695
LA16	**0.104**	0.192	0.299	0.345
LA17	**0.113**	0.235	0.288	0.344
LA18	**0.189**	0.229	0.399	0.292
LA19	**0.099**	0.202	0.333	0.339
LA20	**0.113**	0.213	0.362	0.684
LA21	**0.424**	0.499	0.551	0.787
LA22	**0.350**	0.390	0.697	0.605
LA23	**0.309**	0.347	0.668	0.763
LA24	**0.345**	0.372	0.504	0.824
LA25	**0.314**	0.408	0.765	0.810
LA27	**0.674**	0.853	1.506	1.390
LA30	**0.483**	0.810	0.825	1.513
LA40	**0.664**	0.963	0.980	1.525

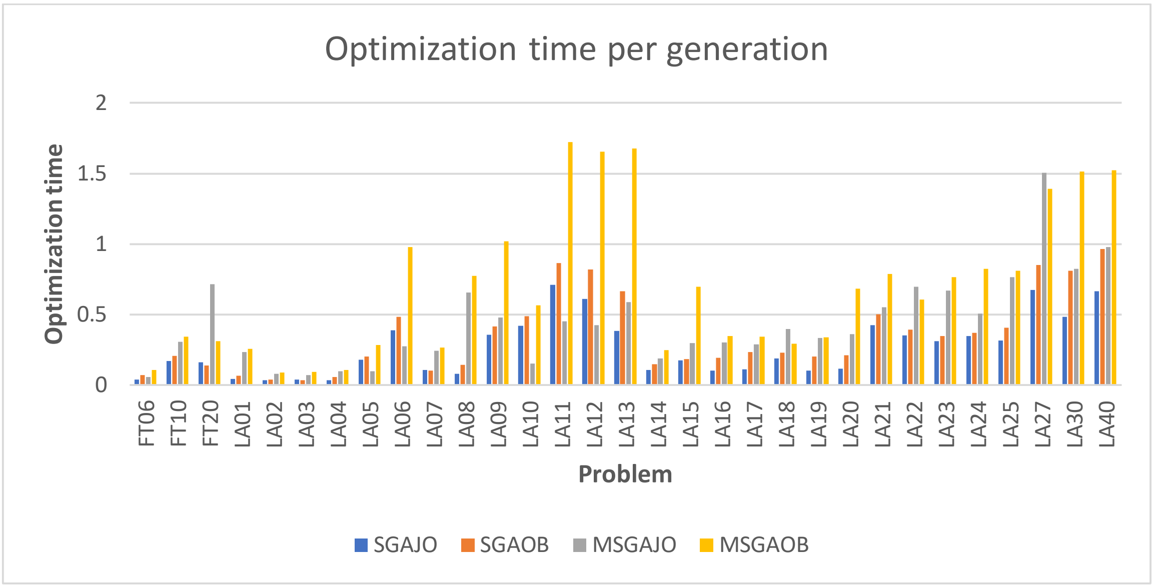

Figure 22. Comparison of optimization time per generation [source: own]

This may be due to the impact of the RRS implementation, as it requires some additional computation time. But on the other hand the MSGA$_{JO}$ with RRS also have some good optimization time. So, the impact on optimization time will also be based on the type of representation used in the SGAs. With or without the usage of RRS in SGA, there is a significant difference between the optimization time of SGA based on job order and operation based representation. This clearly shows that the chromosome representation is one of the important parameter in genetic algorithms.

4.4.10 Comparison of improvement result from best dispatching rule

The best dispatching rule which gives the best makespan value is MTTR (Most Total Time Remaining). The makespan value provided by this rule is compared with the SGAs and MSGAs to determine the improvement of the makespan. The improvement of makespan can be calculated by finding the difference between the best makespan (objective function) of dispatching rule and best makespan of each SGA. This can be explained with the equation (2),

$$Imp = f(x)_{BDR} - f(x)_{BEA} \tag{2}$$

where, Imp = Improvement in min.

$f(x)$ = makespan (objective function) in min.

BDR = Best Dispatching Rule

BEA = Best Evolutionary Algorithm (SGAs and MSGAs).

Table 11. Improvement comparison of best dispatching rule, SGAs and MSGAs

Problem	Active Schedule with MTTR	SGA_{JO}		SGA_{OB}		$MSGA_{JO}$		$MSGA_{OB}$	
		Improve ment	Opt time in min.	Improve ment	Opt time in min.	Improve ment	Opt time in min.	Improve ment	Opt time in min.
FT06	58	3	0.29	3	0.32	3	0.22	3	0.45
FT10	1191	149	2.36	153	2.99	240	3.90	222	5.65
FT20	1594	279	2.57	291	3.00	387	3.56	357	6.04
LA01	784	118	0.59	118	0.70	118	0.87	118	1.49
LA02	852	184	0.59	168	0.65	197	1.14	197	1.44
LA03	713	76	0.59	88	0.65	116	1.07	107	1.40
LA04	817	219	0.61	214	0.67	227	1.08	227	1.47
LA05	621	28	0.61	28	0.68	28	0.33	28	0.96
LA06	930	4	1.42	4	1.66	4	0.94	4	3.31
LA07	1031	116	1.40	137	1.60	141	2.11	141	3.36
LA08	1109	246	1.44	246	1.65	246	2.38	246	3.50
LA09	1021	70	1.44	70	1.64	70	1.69	70	3.52
LA10	1052	94	1.45	94	1.64	94	0.51	94	1.90
LA11	1274	52	2.69	52	3.12	52	1.56	52	5.84
LA12	1167	128	2.70	128	3.09	128	1.47	128	5.66
LA13	1201	51	2.61	51	3.32	51	2.08	51	5.91
LA14	1292	0	0.36	0	0.50	0	0.62	0	0.83
LA15	1415	146	2.61	171	3.26	208	2.42	208	11.33
LA16	1219	234	2.12	240	3.10	242	2.23	246	5.70
LA17	914	111	2.12	102	2.95	130	1.74	127	5.38
LA18	1039	151	2.12	154	3.15	191	2.53	182	5.26
LA19	1123	254	2.16	241	2.90	271	3.11	267	5.61
LA20	1076	143	2.07	117	3.13	169	2.60	164	8.64
LA21	1314	137	5.31	117	7.92	232	6.62	173	14.59
LA22	1135	62	5.13	70	7.73	182	5.16	133	13.87
LA23	1223	144	5.13	117	7.43	191	4.36	177	14.19
LA24	1231	172	5.16	157	7.58	260	4.78	214	13.92
LA25	1206	101	5.23	88	7.52	194	6.20	148	13.95
LA27	1567	128	10.06	126	14.27	285	13.27	179	27.45
LA30	1565	109	10.00	95	14.21	182	8.83	106	27.49
LA40	1549	175	11.21	164	18.13	294	11.55	215	33.16

The resulting improvement value is tabulated and the respecting optimization time is shown in the table 11. The result shows that $MSGA_{JO}$ performed well in terms of the improvement. It may have a bit high optimization time than the minimum optimization of SGA_{JO} in some problem instance, but the improvement makes it to be the best algorithm for the better results.

4.4.11 Result of saved time by improvement

Table 12. Results of saved time by improvement

Problem	SGA_{JO}	SGA_{OB}	$MSGA_{JO}$	$MSGA_{OB}$
	Saved time in min.	Saved time in min.	Saved time in min.	Saved time in min.
FT06	2.71	2.68	2.78	2.55
FT10	146.64	150.01	236.10	216.35
FT20	276.43	288.00	383.44	350.96
LA01	117.41	117.30	117.13	116.51
LA02	183.41	167.35	195.86	195.56
LA03	75.41	87.35	114.93	105.60
LA04	218.39	213.33	225.92	225.53
LA05	27.39	27.32	27.67	27.04
LA06	2.58	2.34	3.06	0.69
LA07	114.60	135.40	138.89	137.64
LA08	244.56	244.35	243.62	242.50
LA09	68.56	68.36	68.31	66.48
LA10	92.55	92.36	93.49	92.10
LA11	49.31	48.88	50.44	46.16
LA12	125.30	124.91	126.53	122.34
LA13	48.39	47.68	48.92	45.09
LA14	-0.36	-0.50	-0.62	-0.83
LA15	143.39	167.74	205.58	196.67
LA16	231.88	236.90	239.77	240.30
LA17	108.88	99.05	128.26	121.62
LA18	148.88	150.85	188.47	176.74
LA19	251.84	238.10	267.89	261.39
LA20	140.93	113.87	166.40	155.36
LA21	131.69	109.08	225.38	158.41
LA22	56.87	62.27	176.84	119.13
LA23	138.87	109.57	186.64	162.81
LA24	166.84	149.42	255.22	200.08
LA25	95.77	80.48	187.80	134.05
LA27	117.94	111.73	271.73	151.55
LA30	99.00	80.79	173.17	78.51
LA40	163.79	145.87	282.45	181.84

The SGAs have improved the makespan value based on their capabilities and all SGAs performed well. With the improvement, the performance of each SGA can be evaluated by calculating the time in which each algorithm saved with the improvement of the makespan value. The saved time can be calculated by finding the difference of

Improvement and total optimization in time. This can be given mathematically in the equation (3),

$$T_s = Imp - T_{opt} \tag{3}$$

where,	Imp	= Improvement in min.
	T_s	= Saved Time in min.
	T_{opt}	= Optimization time in min.

The obtained results of the saved time are tabulated in the table 12. The algorithm MSGA$_{\mathrm{JO}}$ saved more time than the other algorithm on the fly optimization. But on the other all the other algorithms performed well and have reasonable saved time on the fly optimization. The problem LA14 gives the negative result as EAs involves in iterative mechanism, the optimization time will be higher and which results the negative value if makespan obtained by all the search algorithm are same.

5. CONCLUSION

The main aim of this thesis is to review the existing evolution algorithms and selecting one evolution algorithm to select one shop scheduling problem. So, genetic algorithm is chosen and the problem to solve is selected to be the job shop scheduling. In order to develop a new evolutionary algorithm based on genetic algorithm, a new replacement strategy is proposed and implemented in the simple genetic algorithm and also, two type of chromosome representation is analysed in SGA and also in the developed SGA the replacement strategy (RRS).

The outcome of the developed SGA with RRS turns out to be a good strategy for replacement in which the rejected individuals are reused in order to infuse diversity in to the population. The basic idea of the replacement strategy is worked well in the process of optimization of the genetic algorithm.

This RRS method can be applied for various problems and the outcome can be analysed and new possibilities for its usage can be discovered. Also, some of the parameters used in this paper can be changed according to the need of the problem and can combine some existing strategy with the RRS to create a new more efficient than the current one.

One recommendation is that the individuals in the rejected population is sorted in ascending order and then subjected to the replacement process. In this, the individuals can be randomized and then included in the replacement process to make the population with random individuals at certain conditions where there is no improvement in the solution.

The results of saved time with the improvement displays the performance of the SGAs and in most cases there is a chance of getting negative results and hence on the fly optimization is not recommended.

The work can be used in future works related to the optimization of job shop scheduling and other genetic algorithm related problems. This concept can also be used in various optimization problem that involves replacement methods.

REFERENCES

[1] F. Werner, “A survey of genetic algorithms for shop scheduling problems,” *Math. Res. Summ.*, vol. 2, p. 15, 2017.

[2] S. Kolharkar, “‘Scheduling in Job Shop Process Industry,’” *IOSR J. Mech. Civ. Eng.*, vol. 5, no. 1, pp. 1–17, 2013, doi: 10.9790/1684-0510117.

[3] T. Yamada and R. Nakano, “Job-shop scheduling,” no. September 2013, 2000.

[4] L. C. Reyes *et al.*, “Heuristic Algorithms,” *Logist. Manag. Optim. through Hybrid Artif. Intell. Syst.*, pp. 238–267, 2012, doi: 10.4018/978-1-4666-0297-7.ch009.

[5] M. A. Nascimento, S. The, and N. May, “Giffler and Thompson ’ s Algorithm for Job Shop Scheduling is Still Good for Flexible Manufacturing Systems Published by : Palgrave Macmillan Journals on behalf of the Operational Research Society Stable URL : https://www.jstor.org/stable/2583918 REFERENC,” vol. 44, no. 5, pp. 521–524, 2020.

[6] M. Moonen, “A Giffler-Thompson Focused Genetic Algorithm for the Static Job-Shop Scheduling Problem,” pp. 1–14.

[7] J. J. A. Moors *et al.*, “A Survey of Scheduling Rules Author (s): S . S . Panwalkar and Wafik Iskander Published by : INFORMS Stable URL : http://www.jstor.org/stable/169546 REFERENCES Linked references are available on JSTOR for this article : You may need to log in to JSTOR t,” *Int. J. Prod. Res.*, vol. 30, no. 1, pp. 45–61, 2018, doi: 10.1080/00207548208947800.

[8] Werner. F, “An adaptive stochastic search procedure for special scheduling problems,” *Econ. Obz.*, pp. 50–67, 1988.

[9] L. Davis, “Job Shop scheduling with genetic algorithms,” *Proc. First Int. Conf. Genet. Algorithms Their Appl.*, pp. 136–140, 1985.

[10] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st ed. USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[11] D. Beasley, D. R. Bull, and R. R. Martin, “A Sequential Niche Technique for Multimodal Function Optimization,” *Evol. Comput.*, vol. 1, no. 2, pp. 101–125, Jun. 1993, doi: 10.1162/evco.1993.1.2.101.

[12] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in *Proceedings of ICNN’95 - International Conference on Neural Networks*, 1995, vol. 4, pp. 1942–1948 vol.4, doi: 10.1109/ICNN.1995.488968.

[13] V. R. S. E. College and C. Rand, “Particle Swarm Optimization Approach for

Scheduling of Flexible Job Shops," vol. 1, no. 5, pp. 1–6, 2012.

[14] M. Dorigo and M. Birattari, "Ant Colony Optimization," in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds. Boston, MA: Springer US, 2010, pp. 36–39.

[15] E. Özcan, "Memetic Algorithms for Nurse Rostering," in *Computer and Information Sciences - ISCIS 2005*, 2005, pp. 482–492.

[16] J. S. Scheduling, "A memetic algorithm for minimizing the makespan in the Job Shop Scheduling problem Un algoritmo memético para minimizar el makespan en el problema del Job Shop Scheduling Um algoritmo memético para minimizar o makespan no problema do," vol. 26, no. 44, pp. 111–121, 2016.

[17] P. J. M. van Laarhoven, E. H. L. Aarts, and J. K. Lenstra, "Job Shop Scheduling by Simulated Annealing," *Oper. Res.*, vol. 40, no. 1, pp. 113–125, 1992, doi: 10.1287/opre.40.1.113.

[18] E. Nowicki and C. Smutnicki, "An Advanced Tabu Search Algorithm for the Job Shop Problem," *J. Sched.*, vol. 8, no. 2, pp. 145–159, 2005, doi: 10.1007/s10951-005-6364-5.

[19] T. Watanabe, H. Tokumaru, and Y. Hashimoto, "Job-shop scheduling using neural networks," *Control Eng. Pract.*, vol. 1, pp. 957–961, 1993, doi: 10.1016/0967-0661(93)90005-C.

[20] F. Werner, "A survey of genetic algorithms for shop scheduling problems," *Math. Res. Summ.*, vol. 2, no. April 2013, p. 15, 2017.

[21] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1992.

[22] R. Abd Rahman, R. Ramli, Z. Jamari, and K. R. Ku-Mahamud, "Evolutionary algorithm with roulette-tournament selection for solving aquaculture diet formulation," *Math. Probl. Eng.*, vol. 2016, 2016, doi: 10.1155/2016/3672758.

[23] V. Mallawaarachchi, "Introduction to Genetic Algorithm," *Towards Data Science*, 2017. [Online]. Available: https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3.

[24] F. Geyik, M. Gocken, and A. Science, "Genetic algorithm representation types for a two-stage supply chain distribution problem," no. January, 2014.

[25] C. Zhang, Y. Rao, and P. Li, "An effective hybrid genetic algorithm for the job shop scheduling problem," *Int. J. Adv. Manuf. Technol.*, vol. 39, no. 9–10, pp. 965–974,

2008, doi: 10.1007/s00170-007-1354-8.

[26] J. C. Bean, "Genetic Algorithms and Random Keys for Sequencing and Optimization," *ORSA J. Comput.*, vol. 6, no. 2, pp. 154–160, 1994, doi: 10.1287/ijoc.6.2.154.

[27] L. Wang and D.-Z. Zheng, "A Modified Genetic Algorithm for Job Shop Scheduling," *Int. J. Adv. Manuf. Technol.*, vol. 20, no. 1, pp. 72–76, 2002, doi: 10.1007/s001700200126.

[28] S. Mukherjee and A. K. Chatterjee, "On the representation of the one machine sequencing problem in the shifting bottleneck heuristic," *Eur. J. Oper. Res.*, vol. 182, no. 1, pp. 475–479, 2007, doi: 10.1016/j.ejor.2006.07.024.

[29] Y. Song and J. G. Hughes, "Genetic algorithm with a machine order-based representation scheme for a class of job shop scheduling problem," *Proc. Am. Control Conf.*, vol. 2, no. June, pp. 895–899, 1999, doi: 10.1109/acc.1999.783169.

[30] U. Dorndorf and E. Pesch, "Evolution based learning in a job shop scheduling environment," *Comput. Oper. Res.*, vol. 22, no. 1, pp. 25–40, 1995, doi: 10.1016/0305-0548(93)E0016-M.

[31] S. J. T. Liang and J. M. Lewis, "Sparse matrix representation for production scheduling using genetic algorithms," *Proc. ACM Symp. Appl. Comput.*, pp. 313–317, 1995, doi: 10.1145/315891.316004.

[32] B. Integer, L. Programming, G. Algorithm, T. Ga, and T. Ga, "CHAPTER 6 TEST MINIMIZATION FOR COMBINATIONAL BENCHMARK CIRCUITS USING GENETIC ALGORITHM 6 . 2 Overview of Genetic Algorithm 6 . 4 Genetic Algorithms versus Traditional Algorithms," pp. 81–93.

[33] E. Available, "Evolutionary Algorithms for Scheduling Operations," 2016.

[34] "Chapter 6 : SELECTION," pp. 94–124, 1991.

[35] B. MILLER, "Genetic algorithms, tournament selection, and the effects of noise," *Complex Syst.*, vol. 9, pp. 193–212, 1996.

[36] O. Kramer, *Studies in Computational Intelligence 679 Genetic Algorithm Essentials*. .

[37] I. Science, I. Science, I. Science, I. Science, I. Science, and I. Science, "asayuki Yamamura," pp. 3–8, 1996.

[38] Yun-Chia Liang and A. E. Smith, "An ant colony optimization algorithm for the redundancy allocation problem (RAP)," *IEEE Trans. Reliab.*, vol. 53, no. 3, pp. 417–423, 2004, doi: 10.1109/TR.2004.832816.

[39] “Chapter 7 : REPLACEMENT,” pp. 125–135.

[40] N. Design and U. Genetic, “Genetic Algorithm,” pp. 63–85.

[41] J. Mcdermott, “When and Why Metaheuristics Researchers can Ignore ‘ No Free Lunch ’ Theorems,” *SN Comput. Sci.*, vol. 1, no. 1, pp. 1–18, 2020, doi: 10.1007/s42979-020-0063-3.