CHAPTER 2

JOB SHOP SCHEDULING

2.1 OVERVIEW

Scheduling is defined as the process of assigning a set of tasks to resources over a period or it may be defined as the allocation of resources over time to perform a collection of tasks. Scheduling is an optimization method proposed to make the possible use of the limited resources and to make a suitable allotment of the resources over a period. Such complex problems are combinatorial optimization problems, where the set of feasible solutions are discrete and the goal is to find the best possible solution.

The Job Shop Scheduling Problem (JSSP) is the combinatorial optimization problem, which has been the subject of many research efforts for several years. It is extremely difficult to solve, as it comprises of several concurrent goals and several resources, which must be allocated to lead to the goals, it is to maximize the utilization of machines and to minimize the time required to complete the entire process being scheduled (Mesghouni et al 2004). A large number of small to medium companies still operate as job shops (Lin et al 2009). Developing effective scheduling methods that can provide good schedules with less computational time is still a requirement. Most of the real world manufacturing companies aim at successfully meeting the customer needs while improving the performance efficiency.

JSSP is a Non Polynomial hard (NP-hard) problem, and no exact algorithms exist to solve these kinds of problems. It is very difficult to
calculate the optimal schedule even for two jobs and two machine problems; hence, researchers look for heuristics to solve the JSSP with no-wait constraint. JSSP is measured to be NP-hard (Garey et al 1976), and exact algorithms such as integer programming, mixed integer programming and branch and bound method become inefficient to deal with such problem since they can solve only small size problems and may take extremely long computing time when the problem size increases.

Mathematical programming is applied to JSSPs. Problems have been expressed using integer programming, mixed-integer programming, and dynamic programming. Until recently, the uses of these approaches have been limited because scheduling problems belong to the class of NP-complete problems. New solution techniques, more powerful heuristics, and the computational power of modern computers have enabled these approaches to be used on larger problems.

The exact methods such as the branch and bound method, dynamic programming and constraint logic programming need a lot of time to find an optimal solution using the heuristic methods. New search techniques such as genetic algorithms, simulated annealing, ant colony optimization, particle swarm optimization or Tabu search are able to solve the job-shop scheduling problems.

The main objective in solving the job shop scheduling problem is to find the sequence and schedule for each operation on each machine that optimises the objective function. The most common objective function used in scheduling the job shop problem is minimisation of makespan value or the time to complete all jobs. For modified job shop problems where release and due dates may be specified, tardiness may also be used as a performance measure.
There is a sequence of machines, each performs a small task in a production line. These machines are labelled from 1 to m. For a single job to be completed work must be done first with machine 1, then machine 2, etc. all the way to machine m. There are a total of n jobs to be done, each job requires a certain amount of time on each machine, and the amount of time required on one machine may vary from one job to another. A machine can only work on one job at any given time and once a machine starts working, it cannot be interrupted until it has completed its task. The objective is to find the ideal schedule so that the total time to complete all n jobs is minimized. This is the basic principle of solving JSSP problems.

2.2 PROBLEM REPRESENTATION

The Job Shop Scheduling Problem (JSSP) is defined as follows:
Given ‘n’ jobs, each composed of some operations that must be processed on ‘m’ machines. Each operation uses one of the ‘m’ machines for a fixed duration and it is not compulsory to make ‘n’ and ‘m’ are equal. Each machine can process at the most one operation at a time and once an operation initiates processing on a given machine it must complete processing on that machine without interruption. The operations of a given job have to be processed in a given order. The problem involves in finding a good schedule of the operations on the machines, taking into account the precedence constraints, that minimizes the makespan ($C_{max}$), that is the finish time of the last operation completed in the schedule.

JSSP consists of a job $J = \{1,...,n\}$ denotes the set of jobs to be scheduled and $M = \{1,...,m\}$ denotes the set of machines, $J_i$ denotes $i^{th}$ job ($1 \leq i \leq n$) and $M_j$ denotes $j^{th}$ machine ($1 \leq j \leq m$). On the machines $M_1, M_2, ..., M_m$, the jobs $J_1, J_2, ..., J_n$ are to be scheduled. Let $V$ be the set of all operations in all jobs. Each job $J_i$ has a set of operations $o_{i1}, o_{i2}, ..., o_{ik}$ where k is total number of operations in the job $J_i$. The precedence conditions
of operations are related with each job and ensure that operation o_{ij} will be handled only after the processing of operation o_{ij-1} in a specific job i.

In general, typical model of n jobs, m machines job shop is represented by \( \frac{n}{m}/G/P/C_{\text{max}} \). The value \( G \) is a matrix representing the processing order of machines for different jobs. The machine order for \( i^{th} \) job is given by \( G_{ij}(1 \leq j \leq m) \), where \( j \) denotes \( j^{th} \) operation in \( i^{th} \) job. An example of the matrix \( G \) as follows:

\[
G = \begin{pmatrix}
M_2 & M_3 & M_1 \\
M_1 & M_2 & M_3 \\
M_3 & M_1 & M_2 \\
\end{pmatrix}
\]

In the mentioned above matrix, each row of the matrix represents a job. For first job, first operation is completed on machine \( M_2 \), second operation is completed on machine \( M_3 \) and third operation is completed on machine \( M_1 \). Similarly other jobs are performed on different machines.

Matrix \( P \), representing the processing time of different operations, is denoted as follows:

\[
P = \begin{pmatrix}
P_{11} & P_{12} & P_{13} \\
P_{21} & P_{22} & P_{23} \\
P_{31} & P_{32} & P_{33} \\
\end{pmatrix}
\]

where \( p_{ij} \) represents time of \( j^{th} \) operation of \( i^{th} \) job.

The matrix \( G \) and the processing time matrix \( P \) are given as problem facts. The processing order (machine sequence) for machine \( M_i \) is given by \( X_{ik} (1 \leq k \leq n) \), where \( k \) denotes \( k^{th} \) operation to be processed on machine \( M_i \). A solution to JSSP can be represented by matrix \( X \) signifying
processing orders of all machines. For example, one solution of the mentioned problem is considered as follows:

\[
X = \begin{pmatrix}
M1 & o_1 & o_2 & o_3 \\
   J_2 & J_3 & J_1 \\
M2 & J_1 & J_2 & J_3 \\
M3 & J_3 & J_1 & J_2
\end{pmatrix}
\]

Based on the above schedule, first operation of second job is scheduled on machine M_1, followed by second operation of third job and third operation of first job. Likewise other machines have schedules represented in second and third rows. Commonly, subscript values representing machine numbers in G and job numbers in X are given to formulate matrix and matrix representing a solution respectively.

Processing of \( j \)\( ^{th} \) operation of \( i \)\( ^{th} \) job on a machine is denoted as \( o_{ij} \). Each operation \( o \) has at most two direct predecessor operations, a job predecessor \( PJ_o \) and a machine predecessor \( PM_o \). First operation of a machine sequence has no \( PM_o \) and first operation of a job has no \( PJ_o \). In the same way each operation has at most two direct successor operations, a job successor \( SJ_o \) and a machine successor \( SM_o \). Last operation of a machine sequence has no \( SM_o \) and last operation of a job has no \( SJ_o \). An operation \( o \) is called schedulable, if both \( PJ_o \) and \( PM_o \) are already scheduled.
The main objective is to minimize $C_{\text{max}}$ value with certain limitations listed as follows:

- No two operations of one job may be processed simultaneously.
- A machine performs only one job at a time.
- Once an operation is initiated for processing, it will not be interrupted until its completion.
- An operation of a job cannot be started until its previous operations of the same job are completed.
- More than one operations of a job cannot be processed on a single machine.
- Jobs must wait for the next machine to be available.
- Machines may be idle within the schedule period.
- One job is independent with other jobs.

In real world teething troubles, the set of limitations are more complex. Only a few conventions of the basic JSSP may be hold in practice. Therefore, JSSP is widely held in academic research as a trial way for different solution techniques to solve combinatorial optimization problems.

### 2.3 VARIOUS OPTIMIZATION METHODS

The Figure 2.1 describing about the different search methods and optimization techniques (Bandyopadhyay et al 2013)
In recent years, much attention has been made to general heuristics such as Tabu Search, Ant Colony Optimization, Particle Swarm Optimization, Genetic algorithm, and Simulated Annealing to solve the combinatorial optimization problems like JSSP.

2.3.1 Tabu Search

Tabu Search (TS) is a meta-heuristic method which is used to solve combinatorial optimization problems. TS offer solutions very close to optimality and are used to solve the difficult problems. TS are based on Local Search (LS) improvement techniques. LS can be concise as an iterative search procedure that, starting from an initial feasible solution, gradually progresses it by applying a series of local alterations. At each and every iteration, the search moves to improve a feasible solution that varies a little from the
current one. In LS, the excellence of the solution attained and calculating times are usually highly dependent upon the set of alterations considered at each iteration of the heuristic.

The objective of the Tabu Search algorithm is to force a drive in heuristic from returning in recent times visited areas of the search space, mentioned as cycling. The scheme of the approach is to maintain a short term memory of the exact changes of current moves within the search space and avoiding future moves from not doing those changes. Additional intermediate-term memory structures may be introduced to bias moves toward promising areas of the search space, as well as longer-term memory arrangements that support a general diversity in the search across the search space.

The Tabu search starts from local minima. To avoid reviewing the steps used and the method of records recent moves are available in Tabu lists. The original intent of the list was not to prevent an earlier move from being repeated, but fairly to ensure that it was not reversed. The Tabu lists are historical in nature and form the Tabu search memory. The role of the memory can change as the algorithm proceeds.

The idea of Tabu Search (Glover 1989, 1990) is to explore the search space of all feasible scheduling solutions by a sequence of moves. A move from one schedule to another schedule is made by appraising every candidate and choosing the best available, like gradient-based techniques. Certain moves are categorized as Tabu for the reason that they either trap the search at a local optimum, or they lead to cycling. These moves are named as the Tabu List, which is constructed from the moves used during the search. These Tabu moves force exploration of the search space until the old solution area (e.g., local optimum) is left behind. Tabu search methods have been evolving to more cutting-edge frameworks that include longer term memory
mechanisms. These types of cutting-edge frameworks are mentioned as Adaptive Memory Programming (AMP, Glover 1996).

Tabu search methods have been applied to scheduling problems and as solvers of mixed integer programming problems. Nowicki and Smutnicki (1996) employed Tabu search methods for job shop and flow shop scheduling problems. Vaessens (1996) revealed that Tabu search methods (in specific job shop scheduling cases) are superior over other approaches such as simulated annealing, genetic algorithms, and neural networks. Eswaramoorthy & Tamilarasi (2007) developed an algorithm for the problem finding a minimum makespan using Tabu search strategies and the algorithm based on specific neighbourhood and dynamic Tabu length strategies.

2.3.2 Ant Colony Optimization

In new approach in the research area on combinatorial optimization problem using heuristic algorithms take stimulation from the observation of natural processes, modelling them in a technique that can be exploited in optimization problems. Ant Colony Optimization (ACO) is a meta-heuristic algorithm which brings an idea from Artificial Intelligence and Biology, inspired in the collective behaviour of ants. These common insects form colonies of ants, which are self-organizing systems and decentralized which are considered as a Swarm Intelligence. The basic idea of the Ant System is to keep a population or colony of ‘m’ artificial ants that iteratively builds a solution by continually applying probabilistic decision policy ‘n’ times until a solution is found. Ants found a good solution to mark their path through the decision space by putting some amount of pheromone on the edges of the path. Ants of the next iteration are attracted to the pheromone resulting in a higher probability to follow the already traversed good paths. In addition to the pheromone values, the ants will usually be guided by some problem specific heuristic for evaluating possible decisions regarding which direction
to take along the way. Ant System ants have a memory that stores visited components of their current path.

The forerunners of this type of approach can be outmoded back to the 50s, with the Metropolis algorithm (Metropolis 1953). It is only with the growth of interest on genetic algorithms (Holland 1975; Goldberg 1989) and on neural networks (Hopfield & Tank 1985) that few of major characteristics of this type of algorithms emerged, like distributed computation and population-based search. Ant system (AS), is introduced in this type, has already proved effective on the Travelling Salesman Problem (Colorni et al 1991, 1992) and on the Quadratic Assignment Problem (Colorni et al 1994). The ACO meta-heuristic was first introduced by Dorigo et al (1991) and Dorigo (1992), and since then it has been the subject of both theoretical studies and applications. Ant Colony Optimization (ACO) is an approach which aims at manipulating the successful behaviour of real ants in cooperating to find shortest paths to food for solving combinatorial problems (Dorigo & Stützle 2002, Dorigo & Blum 2005).

The Ant Systems loosely models the behaviour that ant colonies have in finding food (Denebourg et al 1983). Exactly, Ant System attempts to capture the basic mechanisms that allow ants, which are almost visionless insects, to find the shortest path between their colony and a feeding source. The algorithm allocates the search effort over many cooperating, simple agents called ants. When these agents are taken alone not capable of attaining good results; on the other hand, their performance becomes very effective when they act collectively.
2.3.3 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is based on observations of the social behaviour of animals, like birds in flocks or fish in schools, as well as on swarm theory. The population containing of individuals or particles is initialized randomly. Each particle is allocated with a randomized velocity based on its own movement experience and that of the rest of the population. The relationship between the swarm and particles in PSO is analogous to the relationship between the population and chromosomes in a GA. In PSO, the problem solution space is formulated as a search space. Each particle position in the search space is a correlated solution to the problem. Particles cooperate to determine the best position (solution) in the search space (solution space).

One of the evolutionary techniques for optimization is Particle Swarm Optimization (PSO) suggested by Kennedy et al (1995). PSO is efficaciously used in different fields due to its ease of implementation and computational efficiency. Coello et al (2002) provided an approach in which one is incorporated into PSO to allow the heuristic to handle problems with numerous objective functions. This PSO algorithm uses a secondary source of particles to guide particle flight. That method could be confirmed using some test functions and metrics derived from the standard literature on evolutionary multi-objective optimization. The result shows that this PSO approach is highly competitive one. Liang et al (2006) conceived a novel PSO-based algorithm for JSSPs. That algorithm efficiently exploits the ability of distributed and parallel computing systems, with simulation results showing the probability of high-quality solutions for classic benchmark problems. Lei & Wu (2006) presented a PSO for the multi-objective JSSP to minimize makespan and total job tardiness concurrently.
Particle Swarm Optimization (PSO) is a method for solving numerical optimization without the clear knowledge of the gradient of the problem to be optimized. Due to Kennedy et al (1995) PSO was originally proposed for simulating social behaviour, but the algorithm was simplified and it was realized that the particles were actually performing optimization. Particle Swarm Optimization is modelled by particles in multi-dimensional space that have a position and a velocity. These particles fly through hyperspace and remember the best position that they have seen. Members of a swarm communicate good positions to each other and adjust their own position and velocity based on these good positions. Communication is done regarding the best known swarm to all and local bests known in neighbourhoods of particles. The PSO algorithm is simple in concept, easy to implement and computational efficient.

2.3.4 Simulated Annealing

Simulated Annealing (SA) is a stochastic heuristic algorithm which is used to resolve combinatorial optimization problems. Simulated annealing optimization technique is analogous to the annealing of metals. SA is motivated by an analogy to annealing in solids. SA is a numerical optimization technique based on the principles of thermodynamics. The idea was first developed by Metropolis et al (1953). SA simulates the behaviour of a system of particles which is gradually cooling down until a strong crystalline structure is reached. If a solid is heated and past melting point and then cool it, the structural properties of the solid depend on the rate of cooling.

When the liquid is cooled slowly enough, large crystals will be formed and, if the liquid is cooled quickly the crystals will contain imperfections. This algorithm simulates the cooling process by gradually lowering the temperature of the system until it converges to a steady, frozen
state. In terms of computational simulation, a global minimum would correspond to such a “cold” (steady) state.

When a “temperature” parameter in the heuristic is high, a great deal of random movement in the solution is tolerated, and as the “temperature” parameter is lowered, less random movement is allowed, until the solution settles into a final “frozen” state. This allows the algorithm to sample the solution space widely when the “temperature” is high, and then gradually move towards simple steepest ascent/descent as the “temperature” cools. The effect is to allow the solution to move out of local optima during the high temperature phase of the operation. This technique is used in SA to search for feasible solutions and converge to an optimal solution. In this model, a parameter (t), equivalent to temperature in annealing, is reduced slowly.

Different from other algorithms, SA uses a possibility mechanism to control the method of jumping out of the local minimum. In the course of search, SA not only receives better solutions, but also accepts worse solutions randomly with a certain probability. At high temperatures, the probability of accommodating better solutions is relatively big. With the reduction of the temperature, the probability of accepting worse solutions also descends, and when the temperature closes in upon zero, SA no longer admits any worse solution. These make SA have more coincidental to avoid getting trapped in a local minimum and avoid the restriction of other local search algorithms and the incline algorithms. As a result of its merits above, SA has become an enormously popular method for solving large-sized and practical problems. However, SA may get trapped in a local minimum or take a long time to find a reasonable solution. For these reasons, SA is often used as a part of a hybrid method.
The current state of the thermodynamic system is analogous to the current scheduling solution, the energy equation for the thermodynamic system is analogous to the objective function, and the ground state is analogous to the global optimum. The global energy $J$, there is a global temperature $T$, which is lowered as the iterations progress. Using this analogy, the technique randomly produces new schedules by sampling the probability distribution of the system (Kirkpatrick et al 1983).

$$p_j = \exp \left( -T \left( \Delta J_{\text{best}} - \Delta J_j \right) / K \right)$$

where $p_j$ represents the probability of making move $j$ from among the neighbourhood choices. $\Delta J_{\text{best}}$ represents the improvement of the objective function for the best choice, and $\Delta J_j$ represents the improvement for choice $j$. $K$ is a normalization factor. Since increases of energy can be accepted, the algorithm is able to escape local minima.

Simulated Annealing has been applied effectively to Job Shop Scheduling Problems. Vakharia & Chang (1990) established a scheduling system based on simulated annealing for manufacturing cells. Jeffcoat & Bulfin (1993) applied simulated annealing to a resource-constrained scheduling problem. Their computational results specified that the simulated annealing procedure provided the best results when compared to other neighbourhood search procedures.

The cooling schedule of a simulated annealing algorithm consists of four components. They are Starting Temperature, Final Temperature, Temperature Decrement and Iterations at each temperature.

**Starting Temperature:** The starting temperature must be hot to allow a move to nearly any neighbourhood state. If this is not done then the final solution will be the same or close to end solution. But, if the temperature
starts at too high value then the search can move to any neighbour and thus transform the search into a random search. Efficiently, the search will be random until the temperature is cool enough to start acting as a simulated annealing algorithm.

Finding the starting temperature is a difficult one and there is no recognised method to find a suitable starting temperature for the entire problem. If the distance between one neighbour and another are already known, then it is easy to compute the starting temperature.

A different way is, starting with a very high temperature and cooling it rapidly more than half of the values of worst solutions are being accepted. This forms a starting temperature and it can be cooled very slowly. Or rapidly heat the system until a certain proportion of worse solutions are accepted and then slow cooling can start.

**Final temperature:** It is a common one to the temperature decrease until it reaches zero. This can make the algorithm run for longer time especially when a geometric cooling schedule is being used. In real time, it is not necessary to reduce the temperature to reach zero because when it approaches zero the chances of accepting a worse move are almost the same as the temperature being equal to zero. So, the stopping can be a suitable low temperature or when the system is “freeze” at the current temperature that is no better or worse moves are being accepted.

**Temperature Decrement:** To decrement the temperature it is critical to the attainment of the algorithm. Ideally, enough number of iterations should be allowed at each temperature so that the system stabilises at that temperature and the number of iterations at each temperature to achieve this might be exponential to the problem size. To decrement the temperature, one mode is a simple linear method and another mode is a geometric decrement.
**Iterations at each temperature:** The final decision is to find out how much iterations are to be made at each temperature. A constant number of iterations at each temperature are an obvious scheme. For that to do only one iteration at each temperature, but to decrease the temperature very slowly or to dynamically change the number of iterations as the algorithm progresses. In a lower temperature a large number of iterations are done and the local optimum can be fully reconnoitred. At higher temperatures, the number of iterations can be less.

### 2.3.5 Genetic Algorithms

Genetic Algorithms (GA) is an optimization methodology based on a direct analogy to Darwinian natural selection in biological reproduction. In this principle, genetic algorithms are encoding a parallel search through concept space, with each process attempting coarse-grain hill climbing (Goldberg 1988). A number of approaches have been utilized in the application of Genetic Algorithms (GA) to job shop scheduling problems (Davis 1985, Goldberg and Lingle 1985, Starkweather et al 1992). Induced variations and recombination of these concepts are tested against an evaluation function to see which one will survive to the next generation.

The use of genetic algorithms requires the following components:

- A mode of encoding solutions to the problem
- An evaluation operation that returns a rating for each solution
- A mode of initializing the population of solutions
• Operators that may be applied to parents when they reproduce to alter their genetic composition such as crossover, mutation and other domain specific operators.

• Parameter setting for the algorithm, the operators and so forth.

Genetic algorithms with simple recombination operators have been employed in job shop scheduling. Their importance on relative ordering schema, absolute ordering schema, cycles, and edges in the offspring arise differences in such blind recombination operators. Sequencing problems have been addressed by mapping their constraints to a Boolean satisfaction problem using partial payoff patterns. This pattern has produced good results for very simple problems. Different types of Heuristic genetic algorithms have been applied to job shop scheduling. In these genetic schemes, problem specific heuristics are incorporated in the recombination operators.

Starkweather et al (1992) used genetic algorithms to solve a dual-criteria job shop scheduling problem in a real production facility. Those criteria were the minimization of average inventory in the plant and the minimization of the average waiting time for an order to be selected. To represent the production or shipping optimization problem, a symbolic coding was used for each member (chromosome) of the population. In this pattern, customer orders are represented by discrete integers. Hence, each member of the population is a permutation of customer orders.

Scheduling types of applications have emphasized the deployment of genetic algorithms as a simple or single algorithmic technique. This has some limitation level and has complexity of the problems solved and their success. Recent research publications have established the sensitivity of genetic algorithms to the initial population. When the initial population is
randomly generated, genetic algorithms are shown to be less efficient than the annealing-type algorithms, but better than the heuristic methods alone.

Genetic Algorithms are based on a resemblance with the genetic structure and activities of chromosomes within a population of individuals. A whole new population of possible solutions is thus produced by selecting the best individuals from the current “generation”, and mating them to produce a new set of individuals. This new generation contains a higher proportion of the characteristics possessed by the good members of the previous generation. In this way, over many generations, good characteristics are spread throughout the population. By favouring the mating of the more fit individuals, the most promising areas of the search space are explored.

Initialization: In the beginning many individual solutions are randomly generated to form an initial population. The population size depends on the nature of the problem, but typically contains several hundreds or thousands of possible solutions. By tradition, the population is generated randomly, covering the entire range of possible solutions. Occasionally, the solutions may be seeded in areas where optimal solutions are likely to be found.

Fitness function: A fitness function must be developed for each problem to be solved. Given specific chromosome, the fitness function returns a single numerical “fitness”, or “figure of merit”, which is supposed to be relative to the “utility” or “ability” of the individual which that chromosome represents. For many problems, particularly function optimization, and the fitness function should simply measure the value of the function. Fitness function is a designed function that measures the goodness of a solution. It should be designed in the way that better solutions will have a higher fitness function value than worse solutions. The fitness function plays a major role in the selection process.
**Selection:** Selection is a genetic operator that chooses a chromosome from the current generation’s population for inclusion in the next generation’s population. Good individuals will probably be selected several times in a generation and poor ones may not be included. Before creating it into the next generation’s population, selected chromosomes may undergo crossover and mutation in which case the offspring chromosomes are actually the ones that make it into the next generation’s population.

Different types of selections have been proposed to select individuals from a population. The commonly used method is Roulette-wheel sample selection. The main technique of this is, better individuals get higher chance to participate to get the fitness value. For that, assign to each individual a part of the roulette wheel then, Spin the wheel n times to select n individuals.

In crossover two individual genetic operators are chosen from the population using the selection operator and that combines a crossover site along the bit strings is randomly chosen. Then the values of the two strings are exchanged up to certain point and new offsprings are created. The two new offspring created from this mating are put into the next generation of the population. By recombining portions of good individuals, this process is likely to create even better individuals.

Mutation is a genetic operator has some low possibility, a portion of the new individuals have some of their bits flipped. The purpose of this is to maintain diversity within the population and inhibit premature convergence. Mutation alone induces a random walk through the search space. Mutation and selection (without crossover) create parallel, noise-tolerant, hill-climbing algorithms. Mutation is generating new offspring from single parent and maintaining the diversity of the individuals.
All chromosomes are evaluated to see how fit they are as solutions. Individuals with the highest fitness are always copied in to the next generation. Evaluation ranks the individuals by some fitness measure that corresponds with the quality of the individual solutions. The selection algorithm determines the reinsertion schemes are global reinsertion for all population based selection algorithm and local reinsertion for local selection. Different schemes of global reinsertions are: Produce as many offspring as parents and replace all parents by the offspring, produce less offspring than parents and replace parents uniformly at random, produce less offspring than parents and it will replace the worst parents and produce more offspring than needed for reinsertion and reinsert only the best offspring.

The generation creation process is repeated till a certain termination condition has been reached. The common terminating conditions are a solution is found that satisfies minimum criteria, fixed number of generations reached, allocated computation time or money reached, the highest ranking solution’s fitness is reaching or has reached plateau such that successive iterations no longer produce better results, manual inspection and combinations of the previous things.

2.4 SUMMARY

This chapter has provided an overview of job shop scheduling problems. The standard model of a job shop scheduling problem has been defined. Various meta-heuristic methods have been analysed.