Chapter 2

Job-Shop Scheduling Problem

2.1 Introduction

Scheduling is a decision-making process that deals with the allocation of resources to tasks over given time periods and its goal is to optimize one or more objectives. Essentially, scheduling may be considered as a search or optimization problem; where it is required to search for a feasible schedule or as an optimization problem; where it is required to search for the best feasible schedule. Scheduling is a daily process found in manufacturing, transportation, networks and distribution systems. The resources may be machines in a workshop, runways at an airport, processing units in a computing environment, and so on. The tasks may be operations in a production process, take-offs and landings at an airport, executions of computer programs, and so on. Scheduling problem is characterized by three components namely:

1. Number of machines, number of jobs and the processing time for each job using appropriate machine.
2. A set of constraints (precedence constraints of operations for a given job and non-overlapping constraints of operations for a given machine).

3. A target function called objective function consisting of single or multiple criteria that must be optimized.

The shop scheduling is one of the most challenging scheduling problems. The shop is a set of jobs $J = \{J_1, \ldots, J_n\}$ and a set of machines $M = \{M_1, \ldots, M_m\}$. The jobs consist of several operations which have to be processed on different machines; i.e. $J_i = \{o_{i,1}, \ldots, o_{i,g_i}, \ldots, o_{i,g_{i,m}} \mid M_j \leq g_j \leq M_m\}$. Two operations of the same job cannot be processed at the same time and a machine can process at most one operation at any time. Shop scheduling is the task of constructing an ordering of the operations on the machines. Shop scheduling may be classified into three main categories: flow-shop scheduling, job-shop scheduling, and open-shop scheduling. Detailed descriptions of these problems is found in (Pinedo, 1995):

- **Flow-Shop Scheduling Problem**: The flow-shop scheduling problem is a shop scheduling problem in which each job $i$ consists of $m$ operations, where $o_{ij}$ must be processed on machine $M_j$, and there are precedence constraints of the form $o_{ij} \rightarrow o_{ij+1}$ where $j = 1, \ldots, m-1$ for each $i = 1, \ldots, n$. This means that all the jobs must follow the same routing along the series of machines. Once a job is completed on one machine, it is placed into the queue of the next machine in the series. An example of a flow-shop Scheduling problem is an assembly line. A factory may want to produce 1000 identical cars. To do this, a scheduler starts the 1000 jobs at a machine and once the operation is processed, the car is sent to the
next station. This process continues until the car has been to every station.

- **Job-Shop Scheduling Problem:** The job-shop scheduling problem is a shop scheduling problem with chain precedences of the form $o_{i,g_1} \rightarrow o_{i,g_2} \rightarrow \ldots \rightarrow o_{i,g_m}$ for $i = 1, \ldots, n$ (i.e. the precedences between operations of the same job build a chain. But there are no precedences between operations of different jobs). This type of problem is widely found in the research publications and is often considered to be representative of many general scheduling problems in practice.

- **Open-Shop Scheduling Problem:** An open-shop scheduling problem is a special case of the shop scheduling in which each job $i$ consists of $m$ operations $o_{i,k}$ ($j = 1, \ldots, m$) where $o_{i,k}$ must be processed on machine $M_k$, and there are no precedence relations between the operations. There is no set order for routing the jobs through the machines. This allows for different jobs to have different routes. The problem is to find job orders (orders of operations belonging to the same job) and machine orders (orders of operations to be processed on the same machine).

In almost all the theoretical work on scheduling, very simple measures of performance have been employed. Some examples of the common performance measures are:

- **Makespan:** Makespan is a maximum completion time of all the jobs. The completion time of a job $i$ is given as follows:

  \[
  C_i = r_i + P_i + W_i
  \]  

  (2.1)
where \( r_i \) is the released time, \( P_i = \sum_{k=g_i}^{x_i} p_{i,k} \) the processing time of the job \( J_i \), and \( \mu_i = \sum_{k=g_i}^{x_i} w_{i,k} \) is the waiting time of the job \( J_i \).

- **Flow time**: Flow time is total time that the jobs spend in the shop.

\[
\sum_{j=1}^{n} F_j = \sum_{j=1}^{n} (P_j + W_j) = \sum_{i=1}^{n} (C_i - r_i)
\]  

(2.2)

- **Total lateness**: Total lateness is given as follows:

\[
\sum_{j=1}^{n} L_j = \sum_{j=1}^{n} (C_j - d_j)
\]

(2.3)

where \( d_i \) is the due time of the job \( J_i \).

However, most of the researches have been focused on optimizing one particular performance measure.

### 2.2 Job-Shop Scheduling Problem

Static and deterministic job-shop scheduling problem (JSSP) is the most widely studied problem. JSSP has been known to the operations research community since 1950's (Jain and Meeran, 1999). JSSP has drawn intense attention from researchers in a variety of disciplines, ranging from operations research to computer science and mathematics; many of which are involved with real-world scheduling problems. JSSP has numerous practical applications which make it an excellent test problem for the quality of new scheduling algorithms.

JSSP of size \( n \times m \) consists of \( n \) jobs \( J = \{J_1, \ldots, J_n\} \) to be achieved on a set of \( m \) machines \( M = \{M_1, \ldots, M_m\} \). Each job \( J_i \) has a sequence of \( m \)
operations \( O_i := (o_{i,g1}, o_{i,g2}, \ldots, o_{i,gm}) \). The operations of one job have to be processed in sequence according to the precedence constraints; i.e., \( o_{i,g1} \rightarrow o_{i,g2} \rightarrow \ldots \rightarrow o_{i,gm} \). Each operation \( o_{i,k} \) is to be processed on a specific machine \( k \) at a starting time \( t_{i,k} \) with a processing time \( p_{i,k} \). Each machine can process only one operation at a time due to disjunctive constraints. In other words, JSSP has the following properties as mentioned in (Mattfeld, 1996):

- Jobs may be started at any time; i.e., no release times exist.
- Jobs may be finished at any time; i.e., no due dates exist.
- No pre-emption of operations is allowed.
- No two operations of one job may be processed simultaneously.
- No job is processed twice on the same machine.
- Each job must be processed to completion.
- Jobs must wait for the next machine to become available in the processing order.
- There is only one type of each machine.
- Machine setup times are negligible.
- Machines are available at any time.
- Machines may be idle within the schedule period.
- The machines processing order of each job is known in advance and immutable.
- No machine may process more than one operation at a time.
A solution to a JSSP is a schedule that specifies the order of all the operations of all jobs on the machines and does not violate any of the constraints. Various scheduling objectives have been introduced for the JSSP. However, most research considers the objective of makespan minimization; i.e., minimizing the maximal completion time of all jobs. In other words, makespan is the total elapsed time since the beginning of the first operation until the completion of the last operation in the schedule. Makespan is denoted $C_{\text{max}}$. If $c_{i,k}$ is the completion time of the operation $o_{i,k}$ of the job $J_i$ on the machine $M_k$, then:

$$c_{i,k} = t_{i,k} + p_{i,k} \quad (2.4)$$

$$C_{\text{max}} = \max \{c_{i,k} : \forall J_i \in J, M_k \in M\} \quad (2.5)$$

where $t_{i,k}$ and $p_{i,k}$ are the starting and the processing times of the operation $o_{i,k}$.

Let $C^*_{\text{max}}$ denotes the minimal makespan in the set of the feasible schedules $S$ of the search space. The JSSP problem is represented mathematically as follow:

$$C^*_{\text{max}} = \min_S \{C_{\text{max}}\} \quad (2.6)$$

Subject to:

$$c_{i,k} - p_{i,k} \geq c_{i,k-1} \quad (2.7)$$

$$c_{h,j} - c_{i,j} \geq p_{h,j} \quad \text{or} \quad c_{i,j} - c_{h,j} \geq p_{i,j} \quad (2.8)$$

where $1 \leq i \leq n$, $2 \leq k \leq m$, $1 \leq h \leq n$, $h \neq i$, and $1 \leq j \leq m$.

Equation 2.7 represents the precedence constraints that the operation $o_{i,k-1}$ precedes the operation $o_{i,k}$. Equation 2.8 represents the disjunctive
constraints that the operation $o_{ij}$ of the job $J_i$ has to be processed on the
machine $M_j$ before or after the operation $o_{hk}$ of the job $J_h$.

2.3 Graphical Representation of JSSP

A disjunctive graph $G = (V, D, U)$ is used to represent JSSP graphically.
$G = (V, D, U)$ is defined as follow:

- $V$ is the set of nodes representing the operations of all jobs. In
  addition, there are two pseudo-nodes: a source ‘$b$’, and a sink ‘$e$’.

$$V = O \cup \{b, e\}$$

(2.9)

In a JSSP of $n$ jobs and $m$ machines, the number of nodes in the
graph is $n \times m + 2$. A weight is associated with each node. The weights
of the pseudo-nodes are zero, while the weights of the other nodes
are the processing times of the corresponding operations.

- $D$ is the set of directed conjunctive arcs which represent the
  precedence constraints between consecutive operations of the same
  job. Additionally, there are conjunctive arcs between the source and
  all the first operations of all jobs, and between all the last operations
  of all jobs and the sink.

$$D = \{(b, o_{1,k}) \cup (o_{i,k}, o_{i+1,k}) \cup (o_{n,k}, e) \mid \forall i, k : o_{i,k} \in O\}$$

(2.10)

Based on the precedence constraints excluding the start node ‘$b$’ and
the completion node ‘$e$’, each node has exactly one pre-operation
node and exactly one immediate one.
Chapter 2. JSSP

- U is the set of undirected disjunctive arcs which represent the disjunctive constraints between the operations that have to be processed on the same machine.

\[
U = \{ (o_{i,k}, o_{j,k}) \mid \forall i, j, k : o_{i,k}, o_{j,k} \in O \}
\] (2.11)

Let us consider a JSSP instance with 3 jobs and 3 machines which is described in Table 2.1 as follows:

**Table 2.1 JSSP (3×3) instance.**

<table>
<thead>
<tr>
<th>Jobs</th>
<th>Sequence of operations</th>
<th>Machine number(Processing time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>J1</td>
<td>2(5) 3(2) 1(3)</td>
<td>1(2)</td>
</tr>
<tr>
<td>J2</td>
<td>1(2) 2(4) 3(1)</td>
<td>3(1)</td>
</tr>
<tr>
<td>J3</td>
<td>1(1) 3(4) 2(3)</td>
<td>2(3)</td>
</tr>
</tbody>
</table>

In Table 2.1, each row represents one job. Each operation is represented by the number of the machine to be processed on and the processing time. The first column is representing the first operations to be processed regarding to the precedence constraints, the second column for the second operations to be processed regarding to the precedence constraints, and so on. The disjunctive graph of the JSSP instance in Table 2.1 is shown in Figure 2.1 where each node (operation) is indexed with the job and machine numbers and the processing time of that node is mentioned above it. The undirected disjunctive arcs are drawn as dashed arcs. The basic scheduling decision is to define an ordering between the operations connected by disjunctive arcs. This can be done by turning the undirected disjunctive arcs into directed ones. A solution is a complete selection if each disjunctive arc has been directed, and the resulting graph \( G' = (V, D, U') \) is acyclic. Where, \( U' \) is the set of directed disjunctive arcs. The schedule can be constructed from the acyclic graph \( G' \) and the makespan of the schedule is the weight (length) of
the critical path. The critical path is the longest path from the source node ‘b’ to the sink node ‘e’. The length of a path is the sum of the processing times of all the nodes on the path.

![Disjunctive Graph](image)

**Figure 2.1** Disjunctive graph for the JSSP (3 × 3) instance

Figure 2.2 represents a possible acyclic Disjunctive graph which is a feasible schedule to the JSSP instance. The critical path is shown as set of thick arcs from the source node to the sink node $C_{\text{Path}} = \{(b, 0_{1,2}), (0_{1,2}, 0_{2,2}), (0_{2,2}, 0_{3,2}), (0_{3,2}, e)\}$ with the weight of 12 which is equal to the makespan of this feasible schedule.

![Acyclic Disjunctive Graph](image)

**Figure 2.2** Acyclic disjunctive graph for the JSSP (3 × 3) instance
Another graphical representation for JSSP solution is a Gantt Chart. The Gantt Chart has two axes; $X$ axis for the time span and $Y$ axis for the machines. Operations are presented as blocks. Each operation has two identifiers, $i$ and $k$; where $i$ is the job number to which the operation belongs and $k$ is the machine number to which the operation is to be performed. The length of the block represents the processing time of its operation. Figure 2.3 shows the Gantt Chart for the solution of the JSSP instance. It is noticeable from the figures 2.2 and 2.3 that the makespan in the Gantt chart is easier to be depicted than in the Acyclic Disjunctive Graph. The last operation of the solution of the JSSP instance is either $o_{3,2}$ or $o_{1,1}$ with a completion time equal to 12.

![Gantt chart for the JSSP (3x3) instance](image)

**Figure 2.3** Gantt chart for the JSSP (3x3) instance

### 2.4 Complexity of JSSP

The complexity of JSSP increases with the number of constraints imposed and the size of search space employed. The primary draw for most researchers is the fact that the JSSP is widely accepted as empirically one of the most difficult NP-Hard problems (Lenstra and Kan, 1979), where NP stands for Non-deterministic Polynomial time. In addition, the JSSP has been considered as a hard combinatorial optimization problem, which is also one of the worst members in that class (Garey et al, 1976). Even a
simple version of the standard JSSP is NP-Hard if the performance measure is the makespan and the number of machines and jobs are greater than 2. The solution to the JSSP can be represented as the operations permutation of the jobs on each machine. The size of search space is the total number of the possible permutations of the jobs on the machines; i.e. \((n!)^m\). It is computationally infeasible to try every possible solution in the search space of the problem. This is because the required computation time increases exponentially with the problem size. However, due to the precedence and conjunctive constraints of the JSSP, not all the possible permutations produce feasible solutions to the JSSP. So, there is only need to find the optimum solution in the set of the feasible schedules. In practice, many real-world JSSPs have a larger number of jobs and machines as well as additional constraints or flexibilities, which further increase its complexity. This means it is believed that there is no way to get an optimal solution in polynomial time. Also, the JSSP is considered in (Garey and Johnson, 1979) as a particularly hard combinatorial optimisation problem. Essentially, the combinatorial optimization seeks a proper permutation and/or combination of some items for a given problem under some constraints. A combinatorial minimization problem can be defined as triple \((S, f(.), \Omega)\), where \(S\) is set of solutions, \(f(.)\) is the objective function, and \(\Omega\) is a set of constraints. The aim of the problem is to find the optimal solution \(s^* \in S\) such that \(f(s^*) \leq f(s)\ \forall s \in S\). One solution is built as a permutation or combination of a set of basic components \(s = \{x_1, x_2, \ldots x_n\}\). A common feature of combinatorial optimization problems is that if the permutation or combination can be determined, a solution then can be derived with a problem-specific procedure.
2.5 Optimization Approaches for JSSP

The JSSP has been extensively studied and a wide variety of approaches have been proposed in many diverse areas, such as operations research, production management and computer engineering. Traditionally, the exact methods such as the branch-and-bound algorithm have been successfully applied by Bucker et al. (1994) to solve small JSSPs. However, for today’s large JSSPs with complex search spaces, it is computationally intractable to obtain an exact optimal schedule within a reasonable time. Because obtaining the exact optimal solution for large JSSPs is non-trivial, it is desirable to obtain as many as near-optimal or possibly optimal solutions in polynomial time, which can be later judged by human experts. Many heuristic and meta-heuristic techniques have been proposed in the literature to search for near-optimal scheduling solutions in reasonable amount of processing time. Heuristic approaches include the local search (Vaessens et al., 1996), the shifting bottleneck approach (Adams et al., 1988), and the like. Meta-heuristic approaches include the Simulated Annealing (SA) (Steinhöfel et al., 2002), the Tabu Search (TS) (Nowicki and Smutnicki, 1996), the Genetic Algorithms (GA) (Yamada and Nakano, 1992), the Ant Colony Optimization (ACO) algorithm (Blum and Samples, 2004), the Particle Swarm Optimization (PSO) algorithm (Ge et al., 2005), and the like. Recently, hybrid heuristics have been a vital topic in the field of computer science, and it is often the case that local search is incorporated into evolutionary approaches in order to improve the results obtained with these methods (Wang and Zheng, 2001). Comprehensive surveys of the general JSSP approaches are found in (Blazewicz et al., 1996) and (Jain and Meeran, 1999). Figure 2.4 shows a possible classification of some optimization techniques that have been used for JSSP.
2.5.1 Exact Methods

The main enumerative strategy is Branch and Bound (B&B) where a dynamically constructed tree representing the solution space of all feasible schedules is implicitly searched. The principle of B&B is the enumeration of all feasible solutions of the problem, where each node of the tree represents a feasible schedule for a subset of the operations. Each branching represents a selection of one of non-scheduled operations. Selection of a particular branch is based on determining the minimum value of a lower bound of the objective function. The strength of the B&B technique depends heavily on the quality of the lower bound. Balas (1969) presented one of the first branch and bound method for the job shop scheduling problem, based on critical operations on the disjunctive graph. Exact methods, such as the B&B method, are always computationally expensive to use when searching for an optimum scheduling solution with a large search space.
2.5.2 Heuristic Methods

A Heuristic is defined by Reeves (1995), as "a technique that seeks good solutions at a reasonable computational cost without being able to guarantee either feasibility or optimality, or even in many cases to state how close to optimality a particular feasible solution is". The term 'heuristic' is derived from the Greek word which means 'to search'. Two types of Heuristic methods have been formed. A constructive heuristic which starting from a null solution and adding elements to build a good complete one, and a local search heuristic which starting from a complete solution and iteratively modifying some of its elements in order to achieve a better one. Many heuristic methods are proposed in the literature to solve job shop scheduling problems such as Priority Dispatching Rules and Shifting Bottleneck Procedure.

Priority Dispatching Rules: Priority Dispatching Rules (PDR) are the earliest approximation algorithms, which have been applied by Fisher and Thompson (1963) to solve JSSPs. In general, whenever a machine is freed, a job with the highest priority in the queue is selected to be processed on a machine. PDR are very easy to implement and have a low computation burden. The highly problem dependent nature of PDR, as in the case of makespan minimization no single rule shows superiority; their nature in making decisions, as they only consider the current state of the machine and its immediate surroundings and that solution quality degrades as problem dimensionality increases. Priority rules are probably the most frequently applied heuristics for solving JSSP in practice because of their ease of implementation and their low time complexity. Blackstone et al. (1982) has given a comprehensive survey on PDR. Some of PDR are described in Table 2.2.
Chapter 2. JSSP

Table 2.2 Priority rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>The operation for the considered machine is randomly chosen.</td>
</tr>
<tr>
<td>SOT</td>
<td>An operation with shortest processing time on the considered machine.</td>
</tr>
<tr>
<td>LOT</td>
<td>An operation with longest processing time on the considered machine.</td>
</tr>
<tr>
<td>FCFS</td>
<td>The first operation in the queue of jobs waiting for the same machine.</td>
</tr>
<tr>
<td>LRPT</td>
<td>An operation with longest remaining job processing time.</td>
</tr>
<tr>
<td>SRPT</td>
<td>An operation with shortest remaining job processing time.</td>
</tr>
<tr>
<td>LORPT</td>
<td>An operation with highest sum of tail and operation processing time.</td>
</tr>
<tr>
<td>SPT</td>
<td>A job with smallest total processing time.</td>
</tr>
<tr>
<td>LPT</td>
<td>A job with longest total processing time.</td>
</tr>
<tr>
<td>LOS</td>
<td>An operation with longest subsequent operation processing time.</td>
</tr>
<tr>
<td>SNRO</td>
<td>An operation with smallest number of subsequent job operations.</td>
</tr>
<tr>
<td>LNRO</td>
<td>An operation with largest number of subsequent job operations.</td>
</tr>
</tbody>
</table>

Shifting Bottleneck Procedure: The Shifting Bottleneck Procedure (SBP) has been implemented by Adams et al. (1988). The SBP was the first successful heuristic to solve the FT10 benchmark instance of the JSSP optimally. The actual strategy involves relaxing the problem into $m$ one-machine problems and solving each sub-problem one at a time. Each machine solution is compared with all the others and the machines are ranked on the basis of their solution. The machine having the largest lower bound is identified as the bottleneck machine. SBP sequences the bottleneck machine first, with the remaining non-sequenced machines ignored and the machines already scheduled held fixed. Every time the bottleneck machine is scheduled each previously sequenced machine susceptible to improvement is locally reoptimised by solving the one machine problem again. The main contribution of this approach is the way the one machine relaxation is used to decide the order in which machines
should be scheduled. The fundamental problem with SBP is the difficulty in performing reoptimisation and the generation of infeasible solutions. In addition no method is available to decide the size of the sub-problem or which machine(s) to fix and in order to solve problems where job routings are more structured SBP will need to be modified.

**Local Search:**

Consider a minimization combinatorial optimization problem is given as \( \min \{ f(x) | x \in S \} \), where \( f(.) \) is the objective function and \( x \) is a feasible solution from a set of feasible solutions \( S \) of a problem. One of the most intuitive solution approaches to this optimization problem is to start with a known feasible solution and slightly perturb it while decreasing the value of the objective function. The following formal definitions are given:

- **A neighbourhood structure** is a function \( N:S \rightarrow 2^S \) that assigns to every solution \( s \) from \( S \) a set of neighbours \( N(s) \) in \( S \). \( N(s) \) is called the neighbourhood structure of \( s \). Often, neighbourhood structures are implicitly defined by specifying the changes that must be applied to a solution \( s \) in order to generate all its neighbours.

- **A local minimum solution** with respect to a neighbourhood structure \( N \) is a solution \( s^* \in S \) such that \( \forall s \in N(s) : f(s^*) \leq f(s) \).

- **A global minimum solution** is a solution \( s^* \in S \) such that for all \( s \in S \) it holds that \( f(s^*) \leq f(s) \).

The Local search (LS) starts form a single initial solution \( s \) and searches the neighbourhood \( N(s) \) for a better solution. The LS move from neighbour to neighbour as long as possible while optimizing the objective.
value. The LS process is repeated until the local minimal solution is found. The main shortcoming of the local search is inability to escape from local minima, once trapped. Vaessens et al. (1996) had proposed a local search procedure to JSSP.

### 2.5.3 Meta-heuristic Methods

(Reeves, 1995) has defined meta-heuristics as high-level heuristics that guide local search heuristics to escape from local optima. The term ‘meta’ is Greek word which means ‘beyond’ or ‘high level’. Stutzle (1998) has cited that the meta-heuristics are typically high-level strategies which guide an underlying, more problem specific heuristic, to increase their performance. The main goal is to avoid the disadvantages of iterative improvement and, in particular, multiple descents by allowing the local search to escape from local optima. This is achieved by either allowing worsening moves or generating new starting solutions for the local search in a more ‘intelligent’ way than just providing random initial solutions. Many of the methods can be interpreted as introducing a bias such that high quality solutions are produced quickly. This bias can be of various forms and can be cast as descent bias (based on the objective function), memory bias (based on previously made decisions) or experience bias (based on prior performance). Many of the meta-heuristic approaches rely on probabilistic decisions made during the search. But, the main difference to pure random search is that randomness in the meta-heuristic algorithms is not used blindly but in an intelligent and biased form. The strength of meta-heuristics comes from the fact that they allow perturbations to non improving solutions and hence are able to transcend local optima. The meta-heuristics are classified into the single-point (trajectory methods) and the population-based methods. The single-point meta-heuristic techniques
start with one solution such as the simulated annealing and the tabu search techniques, while the population-based meta-heuristic techniques are the evolutionary computational techniques such as the genetic algorithm, the particle swarm optimization, and the ant colony optimization. The population-based techniques use population of solutions to escape from local optima.

**Simulated Annealing:** Simulated annealing (SA) is a random oriented search technique that was introduced as an analogy from statistical physics of the computer simulation of the annealing process of a hot metal until its minimum energy state is reached. Matsuo et al. (1988) presented SA as a search algorithm for combinatorial optimization problems. SA starts with initial temperature $T$ and an initial solution. At each iteration of the SA process, the current solution $s$ is replaced by the newly found solution $s'$ according to the following rules:

- If $s'$ is better than $s$, the probability of replacing the solution $s$ with $s'$ is 1.

- Else, the probability of replacing $s$ with $s'$ is $e^{-\Delta / T}$, where $\Delta = f(s') - f(s)$ and $T$ is the current temperature of the annealing process.

The non-improving moves are performed probabilistically for the sake of escaping from the local optima and finding a better solution later during the search. SA algorithm stops when the temperature reaches a predefined minimum value $T_{\text{min}}$. The probability of acceptance non-improving moves is relatively high during the first iterations, and the procedure explores the search space $S$ freely. But, as the iteration count
increases, only improving transitions tend to be accepted, and the solution path is likely to terminate in a local optimum.

The SA has been applied to JSSP first by van Laarhoven et al. (1992). However, they noticed that SA as a generic technique is unable to achieve good solutions to JSSP quickly. As a result, researches are currently directed at combining SA with other techniques in order to improve results and reduce the required computing time. Kolonko (1998) combined a genetic algorithm with SA to solve JSSP efficiently.

**Tabu Search:** Tabu Search (TS) is a meta-heuristic and iterative approximation approach that utilizes a tabu list to search in the neighbourhood for a new solution. A ‘memory’ is implemented by the recording of previously-seen solutions. These can be thought of as a ‘tabu list’ of moves which have been made in the recent past of the search, and which are ‘tabu’ or forbidden for a certain number of iterations. This prevents cycling, allows escape from local optima, and also helps to promote a diversified coverage of the search space. Glover (1989, 1990) proposed successful TS approaches to solve JSSP. More elaborate schemes can be applied to intensify the search in areas which have historically been good or diversify the search to unexplored regions of the solution space. The technique of Nowicki and Smutnicki (1996) is currently one of the most powerful TS approaches allowing good solutions to be achieved very quickly.

**Population-based methods:** Population-based methods include Genetic algorithm (GA), Particle swarm optimization (PSO), and Ant colony optimization (ACO) are the most resent methods applied successfully to solve complicated optimization problems such as the JSSP. The JSSP has
been solved efficiently by hybridization of one Population-based method and other heuristic or meta-heuristic methods. The study in this thesis is based on hybridization approach to solve the JSSP problem. The GA, PSO, and ACO are discussed in details in the chapters 4, 5, and 6 respectively.

2.6 JSSP BENCHMARK PROBLEMS

In the previous section we have briefly described various techniques and algorithms which are used to solve JSSP. However to find the comparative merits of these techniques and algorithms they need to be tested on the same problems. Hence, the birth of 'benchmark problems' has provided a common standard on which all JSSP algorithms can be tested and compared. As the benchmark problems are of different dimensions and grades of difficulty it is simple to determine the capabilities and limitations of a given method by testing it on the benchmark problems. Table 2.3 shows these benchmark problems which were formulated by various authors. The FT, LA, ABZ, ORB, SWV and YN problems are available from the anonymous ftp site (ftp://mscmga.ms.ic.ac.uk/pub/jobshop1.txt), while the TA problems are available from (ftp://mscmga.ms.ic.ac.uk/pub/jobshop2.txt). Both hyper links are at the (OR Library) of the Management School, Imperial College, London, UK. The DMU problems are available from Professor Reha Uzsoy at his Electronics Manufacturing Research.
Table 2.3 JSSP benchmark problems

<table>
<thead>
<tr>
<th>Problem</th>
<th>Size (Jobs×Machines)</th>
<th>Authors (Year)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT06</td>
<td>(6×6)</td>
<td></td>
</tr>
<tr>
<td>FT10</td>
<td>(10×10)</td>
<td>Fisher and Thompson (1963)</td>
</tr>
<tr>
<td>FT20</td>
<td>(20×5)</td>
<td></td>
</tr>
<tr>
<td>LA01 to LA05</td>
<td>(10×5)</td>
<td>Lawrence (1984)</td>
</tr>
<tr>
<td>LA06 to LA10</td>
<td>(15×5)</td>
<td></td>
</tr>
<tr>
<td>LA11 to LA15</td>
<td>(20×5)</td>
<td></td>
</tr>
<tr>
<td>LA16 to LA20</td>
<td>(10×10)</td>
<td></td>
</tr>
<tr>
<td>LA21 to LA25</td>
<td>(15×10)</td>
<td></td>
</tr>
<tr>
<td>LA26 to LA30</td>
<td>(20×10)</td>
<td></td>
</tr>
<tr>
<td>LA31 to LA35</td>
<td>(30×10)</td>
<td></td>
</tr>
<tr>
<td>LA36 to LA40</td>
<td>(15×15)</td>
<td></td>
</tr>
<tr>
<td>ABZ5 – ABZ6</td>
<td>(10×10)</td>
<td>Adams et al. (1988)</td>
</tr>
<tr>
<td>ABZ7 to ABZ9</td>
<td>(15×20)</td>
<td></td>
</tr>
<tr>
<td>ORB1 to ORB10</td>
<td>(10×10)</td>
<td>Applegate and Cook (1991)</td>
</tr>
<tr>
<td>SWV01 to SWV05</td>
<td>(20×10)</td>
<td>Storer et al. (1992)</td>
</tr>
<tr>
<td>SWV06 to SWV10</td>
<td>(20×15)</td>
<td></td>
</tr>
<tr>
<td>SWV11 to SWV20</td>
<td>(50×10)</td>
<td></td>
</tr>
<tr>
<td>YN1 to YN4</td>
<td>(20×20)</td>
<td>Yamada and Nakano (1992)</td>
</tr>
<tr>
<td>TD01 to TD10</td>
<td>(15×15)</td>
<td>Taillard (1993)</td>
</tr>
<tr>
<td>TD11 to TD20</td>
<td>(20×15)</td>
<td></td>
</tr>
<tr>
<td>TD21 to TD30</td>
<td>(20×20)</td>
<td></td>
</tr>
<tr>
<td>TD31 to TD40</td>
<td>(30×15)</td>
<td></td>
</tr>
<tr>
<td>TD41 to TD50</td>
<td>(30×20)</td>
<td></td>
</tr>
<tr>
<td>TD51 to TD60</td>
<td>(50×15)</td>
<td></td>
</tr>
<tr>
<td>TD61 to TD70</td>
<td>(50×20)</td>
<td></td>
</tr>
<tr>
<td>TD71 to TD80</td>
<td>(100×20)</td>
<td></td>
</tr>
<tr>
<td>DMU01 to DMU10</td>
<td>(20×15)</td>
<td>Demirkol et al. (1998)</td>
</tr>
<tr>
<td>DMU11 to DMU20</td>
<td>(20×20)</td>
<td></td>
</tr>
<tr>
<td>DMU21 to DMU30</td>
<td>(30×15)</td>
<td></td>
</tr>
<tr>
<td>DMU31 to DMU40</td>
<td>(30×20)</td>
<td></td>
</tr>
<tr>
<td>DMU41 to DMU50</td>
<td>(40×15)</td>
<td></td>
</tr>
<tr>
<td>DMU51 to DMU60</td>
<td>(40×15)</td>
<td></td>
</tr>
<tr>
<td>DMU61 to DMU70</td>
<td>(50×15)</td>
<td></td>
</tr>
<tr>
<td>DMU71 to DMU80</td>
<td>(50×20)</td>
<td></td>
</tr>
</tbody>
</table>