Chapter - 2

METHODS USED IN FLOW-SHOP SCHEDULING

Following are the techniques developed in order to solve the different types of production scheduling problem.

Mathematical techniques used in Flow-Shop Scheduling

2.1 Branch and Bound Technique:

Branch and bound technique is most widely used in scheduling. It is an enumeration technique and is applied to optimization. It is a useful method of solving many combinatorial problems and is a general purpose strategy for curtailed enumeration. As its name applies the approach consists of two fundamental procedures.

- **Branching:**
  It is the process of partitioning a large number into two or more sub-problems.

- **Bounding:**
  It is the process of calculating a lower bound on the optimal solution of given sub-problems.

- The branching procedure replaces an original problem by a set of new problem that are;

- Mutually exclusive and exhaustive subprograms of the original problems

- Partially solved versions of the original program and

- Smaller problems than the original problem.
Furthermore, sub-problems can themselves be partitioned in a similar fashion.

Branching and bounding form a branching tree with vertices or nodes representing partial of jobs, which have been assigned positions in optimal sequences, i.e. sought to optimized the given criterion. The partial sequences represent the subjects of the sub-problems formed by partitioning. The first level node (p) corresponds to not having assigned any job to any positions in the required sequence. From this node (p), n different branched emanate with a job fixed in the first position. Lower bound, on the performance measure for the given problems, are calculated separately for the partial sequence starting with 1, 2, 3......n respectively. Where lower bound for any partial schedule is the lowest value of the objective function. Irrespective of the order in which the remaining jobs are to be processed. The node or vertex with the lowest lower bound is branched further in (n-1) nodes with the second position being occupied by the remaining (n-1) jobs. Again lower bounds are calculated with the partial sequences with the first two positions being field. The vertex with the least lower bound among the un-branched vertices is branched further. This process continuous till vertex with complete sequence is reached. If the lower bound attached with this node is less than or equal to lower bounds attached with un-branched vertices. The complete sequence represented by this node is an optimal sequence. Otherwise, the node with a lesser lower bound is branched further and the optimal sequence is obtained.

The minimum number of nodes, in branch and bound techniques, which can be created is n(n+1)/2 . This happens when at each stage of further
branching only one node is chosen for the purpose and the value of the complete sequences so obtained is less than all the lower bounds. The maximized number of nodes, in branch and bound method for being created is 1+n+n(n+1)+ n(n+1)(n+2)+.............+n!, which is one all the entices are branched further, where n is the number of jobs.

2.2 Heuristic Approaches:

The branch and bound approach and the elimination approach have two inevitable disadvantages which are typical of implicit enumeration methods. First, the computational requirement will be severe for large problems. Second, even the relatively small problems. There is no guarantee that the solution can be obtained quickly, since the extant of the partial enumeration depends on the data in the problem. Heuristic algorithms avoid these two drawbacks; they can obtain solutions to the large problem with limited computational effort and their computational requirements are predictable for problems of given size. The drawback of heuristic approaches is, of course, that they do not guarantee optimality; and in some instances it may even be difficult to judge their effectiveness.

A Heuristic method is a procedure that is likely to discover a very good feasible solution, but not necessarily a optimal solution, for the specific problem being considered. No guarantee can be given about the quality of the solution obtained, but a well designed heuristic method usually can provide a solution that is at least nearly optimal (or conclude that no such solutions exist). The procedure also should be sufficiently efficient to deal with very large problems. The procedure often is a full-fledged
iterative algorithm, where each iteration involves conducting a search for a new solution that might be better than the best solution found previously. When the algorithm is terminated after a reasonable time, the solution it provides is the best one that was found during any iteration.

Heuristic methods often are based on relatively simple common-sense ideas for how it searches for a good solution. These ideas need to be carefully tailored to fit the specific problem of interest. Thus, heuristic methods tend to be ad hoc in nature. That is, each method is usually designed to fit a specific problem type rather than a variety of applications.

2.2.1 Heuristic and Approximation:

Algorithms are found to perform fairly well. If it is assumed that heuristic methods will not be used when either a constructive polynomial time solution exist or when implicit enumeration is computationally feasible. In other words approximations are not accepted when optimal solution may be easily found.

2.2.2 Polynomial Time Approximation Scheme:

A scheme for generating heuristic algorithms of any desired accuracy and when the algorithm runs in polynomial time. As per the specified accuracy, is called the polynomial time approximation scheme. In other words, the approximation scheme that produces algorithm that has polynomial time complexity, i.e., polynomial both in problem size and the desired Accuracy is called polynomial time approximation scheme.
2.2.2 (a) The Classes P and NP:

To distinguish between two classes P and NP is to distinguish between two types of algorithms; those with polynomial time complexity and those without. The class P consists of all problems for which algorithms with polynomial time behaviour have been found. The class NP is essentially the set of problems for which algorithms with exponential behaviours have been found. Clearly P is contained in NP. If one has a polynomial time algorithm for a problem it can always be inflated inefficiently so that it takes exponential time. Also, occasionally a problem originally in NP, but not in P, is moved into P, as someone with flush of insight discovers a polynomial time algorithm.

2.2.2 (b) NP-Complete:

A problem TM lying in NP is NP-complete if every other problem in NP is polynomial reduces to TM, that is TM $\simeq$ TM for all TM lying in NP. Thus the NP complete problems form a subclass of NP. For if one finds a polynomial time algorithm for any NP complete problem then one can answer all the problems in NP in polynomial time. Thus, if one such algorithm is found, P=NP.

P=NP is equivalent to the conjecture that no NP-complete problem can be answered in polynomial time.

2.2.2 (c) NP-Hard:

When the optimization problem cannot be solved in polynomial time, it is called NP-Hard. For problems, that are NP-Hard, which includes most arising in scheduling; there are at present no easy solutions. The only
methods available are those of implicit (or explicit) enumeration. This may take a prohibitive amount of computation. Certainly large NP-Hard scheduling problems are for all practical purposes solvable. If a problem is large and NP-Hard, then one must consider using heuristic method. But it should be emphasized that NP-Hardness of a problem alone is not sufficient reason to resort to heuristic method. It must also be so large that enumerative method is untraceable.

2.3 Metaheuristic:

For many years, this mean that an OR team would need to start from scratch to develop a heuristic method to fit the problem at hand, whenever an algorithm for finding an optimal solution was not available. This all has changed in relatively recent years with the development of powerful metaheuristic. A metaheuristic is a general solution method that provides both a general structure and strategy guidelines for developing a specific heuristic method to fit a particular kind of problem. Metaheuristic have become one of the most important technique in the toolkit of OR practitioners.

2.3.1 Nature of Metaheuristics:

A metaheuristic is a general kind of solution method that orchestrates the interaction between local improvement procedures and higher level strategies to create a process that is capable of escaping from local optima and performing a robust search of feasible region. Thus, one key feature of a metaheuristic is its ability to escape from a local optimum. After reaching (or nearly reaching) a local optimum, different metaheuristics execute this escape in different ways. However, a
common characteristic is that the trial solutions that immediately follow a local optimum are allowed to be inferior to this local optimum. Consequently, when a metaheuristic is applied to a maximization problem eligible the objective function values for the sequence of trial solutions obtained typically would follow a pattern.

2.3.2 Tabu Search

Tabu search is a widely used metaheuristic that uses common sense ideas to enable the search process to escape from a local optimum.

(i) Basic Concepts

Any application of tabu research includes as a subroutine a local search procedure that seems appropriate for the problem being addressed. (A local search procedure operates like a local improvement procedure except that it may not require that each new trial solution must be better than the preceding trial solution.) The process begins by using this procedure as a local improvement procedure in the usual way. (i.e only accepting an improved solution at each iteration) to find a local optimum. A key strategy of tabu search is that it then continues the search by allowing non-improving moves to the best solutions in the neighbourhood of the local optimum. Once a point is reached where better solutions can be found in the neighbourhood of the current trial solution, the local improvement procedure is reapplied to find a new local optimum. Using the analogy of hill climbing, this process is sometimes referred to as the steepest ascent/mildest descent approach because each iteration selects the available move that goes furthest up the hill, or, when an upward move is not available, selects a move that drops
least down the hill. This use of memory to guide the search by using tabu lists to record some of the recent history of the search is a distinctive feature of tabu research. This feature has roots in the field of artificial intelligence.

Outline Of a basic Tabu Search algorithm

(ii) Initialisation:

Start with a feasible initial trial solution.

(iii) Iteration:

Use an appropriate local search procedure to define the feasible moves into the local neighbourhood of the current trial solution. Eliminate from consideration any move on the current tabu list unless that move would result in a better solution that the best trial solution found so far. Determine which of the remaining moves provides the best solution. Adopt this solution as the next trial solution. Regardless of whether it is better or worse than the current trial solution. Update the tabu list to forbid cycling back to what had been the current trial solution. If the tabu list already had been full, delete the oldest member of the tabu list to provide more flexibility for future moves.

(iv) Stopping Rule:

Use some stopping criterion, such as a fixed number of iterations, a fixed amount of CPU time, or a fixed number of consecutive iterations without an improvement in the best objective function value. (The latter criterion is a particularly popular one.) Also stop at any iteration where
there are no feasible moves into the local neighbourhood of the current trial solution. Accept the best trial, solution found on any iteration as the final solution.

This outline leaves a number of questions unanswered.

1. Which local search procedure should be used?

2. How should that procedure define the neighborhood structure that specifies which solutions are immediate neighbors (reachable in a single iteration) of any current trial solution?

3. What is the form in which tabu moves should be represented on the tabu list?

4. Which tabu move should be added to the tabu list in each iteration?

5. How long should a tabu move be retained on the tabu list?

6. Which stopping rule should one use?

2.3.3 Simulated Annealing

Simulated annealing is another widely used metaheuristic that enables the search process to escape from a local optimum. To better compare it with tabu search

(i) Basic Concepts

The approach used in simulated annealing is to focus mainly on searching for the tallest hill. Since the tallest hill can be anywhere in the feasible region, the early emphasis is on taking steps in random directions (except for rejecting some, but not all, steps that would go
downward rather than upward) in order to explore as much of the feasible region as possible. Because most of the accepted steps are upward, the search will gradually gravitate toward those parts of the feasible region containing the tallest hills. Therefore, the search process gradually increases the emphasis on climbing upward by rejecting an increasing proportion of steps that go downward. Given enough time, the process often will reach and climb to the top of the tallest hill.

T = a parameter that measures the tendency to accept the current candidate to be the next trial solution if this candidate is not an improvement on the current trial solution.

The rule for selecting which immediate neighbour will be the next trial solution is the following.

To be more specific, each iteration of the simulated annealing search process moves from the current trial solution to an immediate neighbour in the local neighbourhood of this solution, just as for tabu search. However, the difference from tabu search lies in how an immediate neighbour is selected to be the next trial solution.

Let \( Z_c \) = objective function value for the current trial solution, \( Z_n \) = objective function value for the current candidate to be the next trial solution.

(ii) Move Selection Rule:

Among all the immediate neighbours of the current trial solution, select one randomly to become the current candidate to be the next trial solution.
solution. Assuming the objective I is maximization of the objective function, accept or reject this candidate to be the next trial solution as follow

If $Z_n \geq Z_c$, always accept this candidate.

If $Z_n < Z_c$, accept the candidate with the following probabilities;

$$\text{Prob(acceptance)} = e^x \text{ where } x = \frac{z_n - z_c}{T}$$

(if the objective is minimization instead, reverse $Z_n$ and $Z_c$ in the above formula). If this candidate is rejected, repeat this process with a new randomly selected immediate neighbour of the current trial solution. (if no immediate neighbour remain, terminate the algorithm)

Outline of a basic Simulated Annealing Algorithm

(iii) Initialization:

Start with a feasible initial trial solution.

(iv) Iteration:

Use the move selection rule to select the next trial solution. (if none of the immediate neighbours of the current trial solution are accepted, the algorithm is terminated).

Check the temperature schedule. When the desired numbers of iteration have been performed at the current value of $T$, decrease $T$ to the next value in the temperature schedule and resume performing iterations at this next value.
(v) **Stopping Rule:**

when the desired number of iterations have been performed at the smallest value of $T$ in the temperature schedule (or when none of the immediate neighbours of the current current trial solutions are accepted), as the final solution.

Before applying this algorithm to any particular problem, a number of details need to be worked out to fit the structure of the problem.

1. How should the initial trial solution be selected?

2. What is the neighborhood structure, that specifies which solutions are immediate neighbours (reachable in a single iteration) of any current trial solution?

3. What device should be used in the move selection rule to randomly select one of the immediate neighbours of the current trial solution to become the current candidate to be the next trial solution?

4. What is the appropriate temperature schedule?

### 2.3.4 Genetic Algorithms

Genetic algorithms provide a third type of metaheuristic that is quite different from the first two. This type tends to be particularly effective at exploring various parts of the feasible region and gradually evolving toward the best feasible solutions. Just as simulated annealing is based on an analogy to a natural phenomenon (the physical annealing process), genetic algorithms are greatly influenced by another form of a natural phenomenon. In this case, the analogy is to the biological theory of
evolution formulated by Charles Darwin in the mid-19th century. Each type of plants and animals has great individual variation. Darwin observed that those individuals with variations that impart a survival advantage through improved adaptation to the environment are most likely to survive to the next generation. This phenomenon has since been referred to as survival of the fittest. The modern field of genetics provides a further explanation of this process of evolution and the natural selection involved in the survival of the fittest. In any species that reproduces by sexual reproduction, each offspring inherits some of the chromosomes from each of the two parents, where the genes within the chromosomes determine the individual features of the child. A child who happens to inherit the better features of the parents is slightly more likely to survive into adulthood and then become a parent who passes on some of these features to the next generation. The population tends to improve slowly over time by this process. A second factor that contributes to this process is a random, low-level mutation rate in the DNA of the chromosomes. Thus, a mutation occasionally occurs that changes the features of a chromosome that a child inherits from a parent. Although most mutations have no effect or are disadvantageous, some mutations provide desirable improvement. Children with desirable mutations are slightly more likely to survive and contribute to the future gene pool of the species. These ideas transfer over to dealing with optimization problems in a rather natural way. Feasible solutions for a particular problem correspond to members of a particular species, where the fitness of each member is measured by the value of the objective function. Outline of a basic genetic Algorithm
(i) **Initialization:**

Start with an initial population of feasible trial solutions, by generating them randomly. Evaluate the fitness (the value of the objective function) for each member of this current population.

(ii) **Iteration:**

Use a random process that is biased toward the more fit members of the current population to select some of the members to become parents. Pair up the parents randomly and then have each pair of parents give birth to two children (new feasible trial solutions) whose features (genes) are a random mixture of the features of the parents, except for the occasional mutations. (Whenever the random number of mixture of features and any mutations result in an infeasible solutions, this is a miscarriage, so the process of attempting to give birth then is repeated until a child is born that corresponds to a feasible solution). Retain the children and enough of the best members of the current population to form the new population of the same size for the next iteration. (discard the other members of the current population). Evaluate the fitness for each new member (the children) in the new population.

(iii) **Stopping Rule:**

Use some stopping rule, such as a fixed number of iterations, a fixed amount of CPU time, or a fixed number of consecutive iterations without any improvement in the best trial solution found so far. Use the best trial solution found on any iteration as the final solution.
Before this algorithm can be implemented the following questions need to be answered.

1. What should the population size be?

2. How should the members of the current population be selected to become parents?

3. How should the features of the children be derived from the features of the parents?

4. How should mutations be injected into the features of the children?

5. Which stopping rule should be used?

The answers to these questions depend on the structure of the specific problem being addressed. The metaheuristic area includes two versions of algorithm. One is for very small integer nonlinear programming problems and the other is for small travelling salesman problems.