Machine Scheduling Problem (MSP)

2. The Multi-Criteria Scheduling Problem

The multi-criteria scheduling problem can be stated as follows. There are n jobs to be processed on a single machine, each job i has processing time p_i and due date d_i at which ideally should be completed. Penalties are incurred whenever a job i is completed earlier or later than its due date d_i. Multi-criteria optimization with conflicting objective functions provides a set of Pareto optimal solutions, rather than one optimal solution. This set includes the solution that no other solution is better than with respect to all objective functions. In the literature, there are two approaches for multi-criteria scheduling problems: the **hierarchical approach** and the **simultaneous approach**. In the hierarchical approach, one of the two criteria is considered as the primary criterion and the other one is considered as the secondary criterion. The problem is to minimize the primary criterion while breaking ties in favor of the schedule has minimum secondary criterion value.

For the simultaneous approach, there are two types; the first one is to find the sum of these objectives. The second one typically generates all efficient schedules (set of **Pareto optimal solutions**) and selects the one that yields the best composite objective function value of the criteria. Several studies by Van Wessenhove and Gelder [88], Hoogeveen [48], Alasaf [7], Findi [35] and Hoogeveen [46] are examples of simultaneous minimization scheduling problems. The objective function of MSP can be classified as in figure (1).

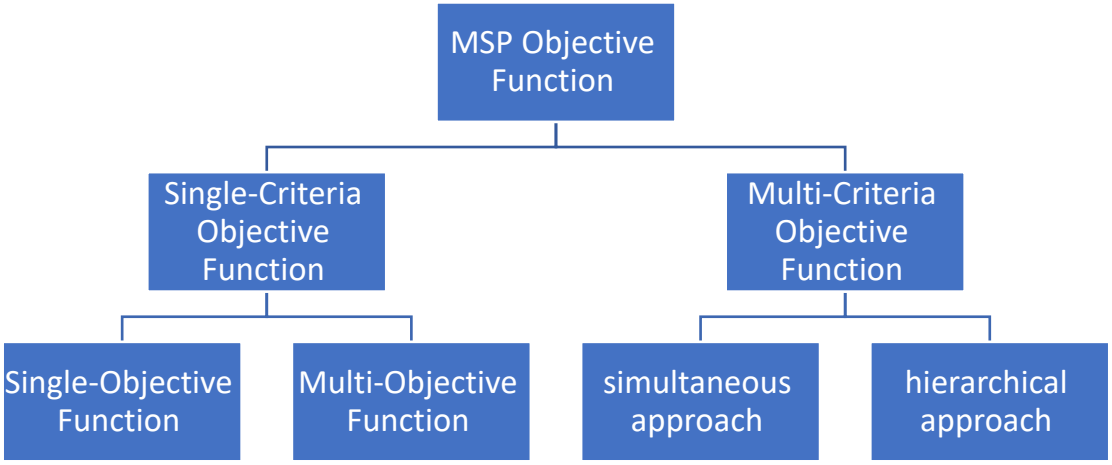

Figure (1): Objective function of MSP classification.

3. Algorithm and Complexity

An **algorithm** is a method for solving a class of problems on a computer. The **complexity** of an algorithm is the cost, measured in running time, or storage, or whatever units are relevant, of using the algorithm to solve one of those problems [42].

The time complexity of a calculation is measured by expressing the running time of the calculation as a function of some measure of the amount of data that is needed to describe the problem to the computer.

The general rule is that if the running time is at most a polynomial function of the amount of input data, then the calculation is an easy one, otherwise it's hard. For instance, matrix inversion is easy. The familiar Gaussian elimination method can invert an n×n matrix in time at most cn^3. For instance, think about this statement: I just bought a matrix inversion program, and it can invert an n×n matrix in just $1.2n^3$ minutes. We see here a typical description of the complexity of a certain algorithm. The running time of the program is being given as a function of the size of the input matrix. A faster program for the same job might run in $0.8n^3$ minutes for an n×n matrix [42].

A significant research topic in scheduling is the use of complexity theory to classify scheduling problems into two classes [27]:

- P (polynomial), which the class P contains all decision problems that are polynomially solvable.
- NP (non-deterministic polynomial), contains many difficult problems for which no polynomial algorithm has been found.

However there is always a possibility that someone will prove that P=NP but the chance would seem to be a very small one.

Many problems in CO can be solved by using an appropriate algorithm. Informally, an algorithm is given a (valid) input, i.e., a description of an instance of a problem and computes a solution after a finite number of "elementary steps". The number of bits used to describe an input I is called the (binary) *length* or *size* of the input and denoted size(I). Let $t : \mathbb{N} \to \mathbb{R}$ be a function. We say that an algorithm runs in time $O(t)$ if there is a constant α such that the algorithm uses at most $\alpha t(\text{size}(I))$ many elementary steps to compute a solution given any input I. An algorithm is called *polynomial time* if $t: n \mapsto n^c$ for some constant c. This contrasts *exponential time* algorithms where $t: n \mapsto c^n$ for some constant $c>1$.

Because the running times of exponential time algorithms grow rather rapidly as the input size grows, we are mostly interested in polynomial time algorithms. Of course, we desire to find an optimum

solution for any given COP in polynomial time. Unfortunately this is not always possible as many COPs are NP-hard. (It is widely believed that no polynomial time algorithm exists that solves some NP-hard COP optimally on every instance). Thus our goal is to find "good" solutions in polynomial time.

For example, for MSP, a major theme in recent research has been used the complexity theory to classify MSP's as polynomially solvable or NP-hard. Many fundamental results in this area are derived by Lenstra et al. (1977) [63]. Classifying MSP's according to their computational complexity was first discussed by Cook (in 1971) [27] and Karp (1972) [57].