Study on meta-heuristics for resource constrained project scheduling problem

Neeraj Kumar

Abstract—This study deals with the survey of meta-heuristics for resource constrained project scheduling problem (RCPSP). It is a NP-hard problem in strong sense. From the beginning of RCPSP research many researchers have attracted, rapid progress regarding models and methods has been documented in this survey. Most of study in this paper address to the non-preemptive case where resources are renewable and objective function is to minimizing project duration. meta-heuristic procedures are best to find near-optimal solutions for large problem instances quickly.

Index Terms—Project, Scheduling problem, RCPSP, Heuristic, Meta-Heuristic.

I. INTRODUCTION

The resource constrained project scheduling problem (RCPSP) is a general scheduling problem that includes precedence and resource constraint [12, 20]. RCPSP is an assignment to determine the feasible start time for each activity in the network of a project such that the precedence and resource availability constraints are satisfied, in order to optimize predefined performance measures (e.g. minimizing makespan, minimizing cost, weighted earliness/tardiness, max. net present value, etc.)

It is a NP hard problem in strong sense. This indicates that there are no known techniques or algorithms for finding optimal solution. Being one type of NP hard problem, The RCPSP presents too large a search space for exhaustive enumeration to attain an optimum.

The definition of resource constrained project scheduling problem (RCPSP) has been given by [12] as follow:

- A single project consists of a set \( J = \{0, 1, \ldots, n, n+1\} \) of activities which have to be processed. Fictitious activities 0 and \( n+1 \) correspond to the “project start” and to the “project end”, respectively.

- The activities are interrelated by two kinds of constraints. First, precedence constraints force activity \( j \) not to be started before all its immediate predecessor activities comprised in the set \( P_j \) have been finished. Second, performing the activities requires resources with limited capacities.

- K resource types, given by the set \( K = \{1, \ldots, K\} \). While being processed, activity \( j \) requires \( r_{j,k} \) units of resource type \( k \in K \) during every period of its non–preemptable duration \( p_j \). Resource type \( k \) has a limited capacity of \( R_k \) at any point in time.

- The parameters \( p_j, r_{j,k} \) and \( R_k \) are assumed to be deterministic; for the project start and end activities we have \( p_j = 0 \) and \( r_{j,k} = 0 \) for all \( k \in K \).

- The objective of the RCPSP is to find precedence and resource feasible completion times for all activities such that the makespan of the project is minimized.

II. CLASSIFICATION OF RCPSP

We broadly separate the RCPSP into the following 6 different classes according to [3,20],

(i) Basic Single-Mode RCPSP
(ii) Basic Multi-Mode RCPSP
(iii) RCPSP problems with Non-regular objective functions
(iv) Stochastic RCPSP
(v) Bin-packaging-related RCPSP
(vi) Multi-resource-constrained project scheduling problems (MRCPS)

I. Basic Single-Mode RCPSP

In the single-mode RCPSP, each project (activity) has a single execution mode: both the activity duration and its requirements for a set of resources are assumed to be fixed, and only one execution mode is available for any activity. The network of a project follows AON representation, in which the arcs in the graph specify precedence relationships. In stating precedence conditions, we can consider two types: cases in which activity \( j \) can start at any time following completion of activity \( i \), or cases in which activity \( j \) must start within some time window following the completion of activity \( i \) (we call these latter restrictions General Precedence Relationships, or GPR).
ii. Multi-Mode RCPSP

In a multi-mode RCPSP (MM-RCPSP), given the estimated work content for an activity, a set of allowable execution modes can be specified for the activity’s execution. Each mode is characterized by a processing time and amount of a particular resource type for completing the activity. For example, one worker might finish a job in 10 hours (mode 1), whereas 2 workers might finish the same activity in 5 hours (mode 2).

iii. RCPSP problems with non-regular objective functions

A regular objective function is one in which the objective function is never made worse by reducing the completion time of a job without increasing the completion time of any other job. A non-regular objective function violates this property.

iv. Stochastic RCPSP

In a stochastic RCPSP, the processing time of any activity is a random variable, which follows some probability distribution. This problem class while often more realistic, leads to much greater complexity in analysis. Instead of minimizing makespan, we consider objectives such as minimizing the expected makespan. Since many interdependent activities are often represented in a project graph, and activity completion times are highly interdependent, the probability distribution of the total makespan is often extremely difficult or impossible to characterize, often leading to activity independence assumptions for tractable analysis. Such independence assumptions, however, can provide extremely misleading results in practice.

v. Bin-packing-related RCPSP problems

A bin-packing-related RCPSP can be seen as analogous to a bin-packing problem. The analogy between the bin packing and RCPSP problems can be explained as follows. The resource capacity represents the bin size, while a task’s resource consumption requirement represent an item size. In the RCPSP context, we can view each time period as a bin into which we can pack different tasks (of course in many of these problems, we can pack an item into consecutive bins, or equivalently schedule jobs across consecutive days). If we suppose, however, that each task takes less than one period of resource consumption and that every task must be fully completed within a single day, then minimizing makespan of this RCPSP is equivalent to minimizing the number of bins used in an equivalent bin-packing problem.

(vi) Multi-resource-constrained project scheduling problems (MRCPSP)

In multi-resource-constrained project scheduling problems, a job may require a set of operations, or a set of successive resources. For a given operation, several resources may be in parallel, which means the job can select any one of these resources for processing. A job might also need to complete processing on one resource before it begins processing on another resource, where successive resources are needed in series. These problems are often called machine-scheduling problems, since in manufacturing we often see machines and workstations in parallel and in series. In the manufacturing context, a resource might be a machine that can only process one job at any time. Machine and job scheduling characteristics can be broken down to several categories:
1. Identical machines in parallel;
2. Machines in parallel with different speeds: the speed of machine $i$ is $v_i$ for all jobs;
3. Unrelated machines in parallel: speed of machine $i$ for job $j$ is $v_{ij}$;
4. Flow shop: machines are in series; every job has to be processed on each of the machines in the same sequence.
5. Job shop: a job can visit any subset of the machines in any order.

Methods for solving RCPSP

The solution procedures for resource constrained project scheduling problems (RCPSP) have been classified into following as [12, 22].

The exact methods provide optimum schedules while the heuristic procedures provide feasible schedules which may or may not be optimal.

III. EXACT ALGORITHM

Exact methods are involved with finding the optimal solutions. However they are impractical when faced with problems of any significant size or with large sets of
constraints. Exact methods can be classified into two groups: (1) mathematical programming and (2) branch and bound.

3.1. Mathematical programming approach
The problems that mathematical programming deals with are presented as:

\[
\text{Minimize } f(x_1, x_2, \ldots, x_n) \\
\text{With respect to } x_1, x_2, \ldots, x_n \text{ subject to the constraints:}
\]
\[
g_1(x_1, x_2, \ldots, x_n) = b_1 \\
g_2(x_1, x_2, \ldots, x_n) = b_2 \\
\ldots \ldots \ldots \\
g_k(x_1, x_2, \ldots, x_n) = b_k \\
x_1, x_2, \ldots, x_n \geq 0
\]

We define the \(x_1, x_2, \ldots, x_n\) as limited to integral values. This will become Integer Programming. Mathematical Programming tends to take longer to find a solution than implicit enumeration algorithms (branch and bound) designed specifically for a particular class of problem e.g RCPCP.

3.2 Branch and bound Procedures
Branch and Bound is a typical implicit enumerative method. It enumerate, all possible schedules and then eliminate the non-optimal schedules from the list, leaving those that are optimal [22]. This approach consists of two fundamental processes: branching is the process of partitioning a large problem into two or more sub-problems, and bounding is the process of calculating a lower bound on the optimal solution of a given sub-problem. The various procedures differ primarily with respect to the branching strategy and the generation of bounds. Patterson and Huber (1974) suggested two exact bounding algorithms, a minimum bounding algorithm and a maximum bounding algorithm. The minimum bounding algorithm starts with an infeasible and super-optimal project completion time without the resource constraint. Then this minimum bounding algorithm increases the completion time by one unit until an optimal or feasible solution is obtained. Conversely, the maximum bounding algorithm starts with a feasible and sub-optimal project completion time. Then this maximum bounding algorithm iteratively decreases the completion time until an infeasible solution is found.

Heuristic procedures for RCPSP can be classified into priority rule based heuristics and meta-heuristics.

4.1 Priority rule based heuristics
Priority rule based heuristics require some priority rule for activity prioritization and a schedule generation scheme. There are two types of schedule generation schemes are described below.

4.1.1 Schedule Generation Schemes
Schedule generation schemes (SGS) are the core of any heuristic solution procedures for the RCPSP. As the problem of resource constrained project scheduling requires the scheduling of the activities, they are most basic to any heuristics. Schedule generation scheme starts from scratch. It builds a feasible schedule by stepwise extension of a partial schedule. A partial schedule is a schedule where only a subset of the total activities has been scheduled. They can be distinguished w.r.t the incrementation into activity–incrementation.

(i) Serial Schedule Generation Scheme
Serial SGS performs activity–incrementation and the so–called parallel SGS performs time–incrementation.

(ii) Parallel Schedule Generation Scheme

IV. HEURISTIC ALGORITHMS
In recent years a good deal of work has been done in the development of heuristic methods for solving resource constrained project scheduling problems. A ‘heuristic’ is a guide or method of reducing the search space in a problem solving situation, an aid to the discovery of a solution.

Note:
The serial SGS always generate feasible schedules which are optimal for the resource–unconstrained scheduling.
Parallel SGS performs time-incrementation, i.e. extension of partial schedule is done on the time basis. Parallel SGS also begin the activity at time 0. The difference is that it computes a so-called decision point which is the time at which an activity to be scheduled is started. This decision point is determined by the earliest finish time of the activities currently in process. For each decision point, the set of eligible activities is computed as the set of those activities that can be feasibly started at the decision point. The eligible activities are selected successively and started until none are left. Then the next decision point and a related set of eligible activities are computed. This is repeated until all activities are feasibly schedule.

### 4.1.2 Priority Rules

A priority rule is a mapping which assigns each activity in the decision set \( D_g \) a value of the priority and an objective stating whether the activity with the minimum or the maximum value is selected. In case of ties, one or several tie breaking rules have to be employed. The easiest ways to resolve ties is to choose the activity with the smallest activity label.

Priority rules may be classified according to different criteria. With respect to information employed to calculate the priority value, distinguished priority rules based on network, time and resource as well as lower and upper bound rules. With respect to the amount of information employed to calculate a priority value they can be distinguished in local or global rules. Local rules employ only information from the activity under consideration such as the processing time while global rules make use of a wider range of information. With respect to the schedule generation scheme they can be distinguish in rules which can be employed in the serial, the parallel or both SGS.

Well known priority rules analyzed [12] as following in the Table:

<table>
<thead>
<tr>
<th>Serial No:</th>
<th>Priority Rule</th>
<th>Priority value ( v(j) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Greatest rank positional weight (GRPW)</td>
<td>( p_j + \sum_{i \in S_j} p_i )</td>
</tr>
<tr>
<td>2</td>
<td>Latest finish time (LFT)</td>
<td>( LF_j )</td>
</tr>
<tr>
<td>3</td>
<td>Latest start time (LST)</td>
<td>( LF_j - p_j )</td>
</tr>
<tr>
<td>4</td>
<td>Minimum slack (MSLK)</td>
<td>( LF_j - EF_j )</td>
</tr>
<tr>
<td>5</td>
<td>Most total successors (MTS)</td>
<td>(</td>
</tr>
</tbody>
</table>

### X-Pass Methods

Priority rules can be combined in order to obtain different priority rule based heuristics. Priority rule based heuristics combine priority rules and schedule generation schemes in order to construct a specific algorithm, namely: single pass and multi pass methods.

(i) **Single pass method**

Single pass methods employ one SGS and one priority rule in order to obtain one feasible schedule.

(ii) **Multi pass method**

There are many possibilities to combine SGS and priority rules to a multi pass method. The most common ones are multi priority rule methods, forward–backward scheduling methods, and sampling methods.

**Multi priority rule methods**

Employ the SGS several times and each time a different priority rule each time is used. Generally the rules are used in the order of descending solution quality.

(i) **Forward–backward scheduling methods:**

Employ an SGS in order to iteratively schedule the project by alternating between forward and backward scheduling. Backward scheduling applies one of the SGS to the reversed precedence network where the former end activity \( n+1 \) has become the new start activity. The priority values are usually obtained from the start or completion times of the last generated schedule.

(ii) **Sampling methods:**

Make generally use of one SGS and one priority rule. Different schedules are obtained by biasing the selection of the priority rule through a random device. Instead of a priority value a selection probability is computed. Dependent on how the probabilities are computed, one can distinguish random sampling, biased random sampling, and regret based biased random sampling.

### V. META HEURISTICS FOR RCPSP

A **metaheuristic** is a heuristic method for solving a very general class **combinatorial optimization** problems by combining heuristics in a hopefully efficient way. The name...
combines the Greek prefix "meta" ("beyond", here in the sense of "higher level") and "heuristic" (from heuriskein, "to find").

Metaheuristics are generally applied to problems for which there is no satisfactory problem-specific algorithm or heuristic; or when it is not practical to implement such a method. Most commonly used metaheuristics are targeted to combinatorial optimization problems, but of course can handle any problem.

Several meta-heuristic strategies have been developed to solve hard optimization problems. There are following Meta-Heuristics describes those general approaches that have been used to solve the RCPSP.

Some well-known meta heuristics are:

- Local search
- Simulated annealing
- Genetic algorithms
- Ant colony algorithms
- Tabu search

### 5.1 Local search

Local search algorithm is also known as descent algorithm. It is a simple procedure for heuristically solving combinatorial problems[2,21]. The basic procedure of local search is following as:

**Step(i)** A simple form of local search algorithm start with an initial (arbitrary) solution perhaps chosen at random.

**Step(ii)** A neighbour of this solution is then generated by some suitable mechanism

**Step(iii)** The change in objective function ‘f’ (e.g. cost, time, etc) is calculated respectively neighbour solution. If the neighbour solution provide a better objective function value than the initial solution, accept the neighbour solution as the new the current solution otherwise the initial solution is retained.

**Step(iv)** If the stopping condition has not been met, The process is repeated from Step(ii) until no further improvement in the neighbourhood of the current solution and the descent algorithm terminates at a local minimum.

Although a descent algorithm is simple and quick to execute, the disadvantage of the method is that the local minimum found may be far from any global minimum.

One way of improving the solution is to run the descent algorithm several times starting from different initial solution, and take the best of the local minima found.

### 5.2 Simulated annealing algorithm

**Simulated Annealing** (SA), introduced by Kirkpatrick (1983), and originates from the physical annealing process in which a melted solid is cooled down to a low-energy state.

SA algorithm is a special type of local search algorithm with the first fit strategy (i.e. simple greedy procedure) where all solutions that improve the objective function value are accepted immediately, but other solution are either rejected or accepted with certain probability. Unlike the local search, the SA algorithm attempt to avoid becoming trapped in a local optimum by sometime accepting a neighbourhood move which increase the value of f (objective function). The acceptance or rejection of an uphill move is determined by a sequence of random numbers, but with a controlled probability. The probability of accepting a move which cause an increase Δ (next − f current) in f is called the acceptance function and is normally set to exp(−Δ/T), where T is a control parameter which correspond to temperature in the with physical annealing, see [23]. This acceptance function implies that small increase in f are more likely to be accepted than large increase, and that when T is high moves will be accepted than large increase, and that when T approaches zero most uphill moves will be rejected .

So in SA, the algorithm is started with a relatively high value of T, to avoid being prematurely trapped in a local optimum. The algorithm proceeds by attempting a certain number of neighbourhood moves at each temperature, while the temperature parameter is gradually dropped.

#### 5.2.1 The Physical Analogy

Physical annealing [4,23] refers to the process of finding low energy states of a solid by initially melting the substance and then lowering the temperature slowly, spending a long time at temperature close to the freezing point.

Simulated annealing is based on an analogy to the cooling of heated metals. In any heated metal sample the probability of some cluster of atoms at a position, r, exhibiting a specific energy state, E(r), at some temperature T, is defined by the Boltzmann probability factor:

\[
P(E(r)) = e^{-\frac{E(r)}{k_B T}}
\]

where \(k_B\) is Boltzmann's constant. As a metal is slowly cooled, atoms will fluctuate between relatively higher and lower energy levels and allowed to equilibrate at each temperature T. The material will approach a ground state, a highly ordered form in which there is very little probability for the existence of a high energy state throughout the material.
If the energy function of this physical system is replaced by an objective function, 
\( f(X) \), that is dependent on a vector of design variables, \( X \), then the slow progression 
towards an ordered ground state is representative of a progression to a global optimum. 
To achieve this, a control parameter \( T \), analogous to a temperature, and a constant \( C \),
alogous to Boltzmann's constant, must be specified for the optimization problem. In 
standard iterative improvement methods, a series of trial points is generated until an
improvement in the objective function is noted in which case the trial point is accepted.
However, this process only allows for downhill movements to 
be made over the domain. In order to generate the annealing
behaviour, a secondary criterion is added to the
process. If a trial point generates a large value of the objective
function then the
probability of accepting this trial point is determined using the
Boltzmann probability

distribution:

\[
P[\text{accept } X_t] = e^{-\frac{f(X_t) - f(X_0)}{CT}}
\]

where \( X_0 \) is the initial starting point. This probability is
compared against a randomly
generated number over the range \([0..1] \). If \( P[\text{accept } X_t] \geq \) random \([0..1] \) then the trial
point is accepted. This dependence on random numbers
makes simulated annealing a
stochastic method.

5.2.2 SA Algorithm for RCPSP
This procedure based on [4, 6, 13,] as follow:
Step(i) Read the project data and calculate its critical path
(CP) value.
Step(ii) Set the SA parameter: \( T_{(o)max} \), \( C \), random
Step(iii) Calculate initial solution \((x_o)\) and objective function
\( f(x_o) \) using the priority rule based heuristics.
Step(iv) Generate a neighbour solution \((x_t)\) and calculate \( f(x_t) \).
Step(v) Calculate \( \Delta = f(x_t) - f(x_o) \)
Step(vi) if \( \Delta < 0 \), then store \( x_t \).
Step(vii) If \( P = \exp(-\Delta/CT) > x_{random} \) then accept \( x_t \), if not then
go to step(iv).
Step(viii) if not, calculate decrease the control parameter \((T)\)
\( = C T_{(o)max} \) it is as cooling schedule. And go to step(iv)
Step(ix) Repeat procedure.
Step(ix) END

5.2.3 Solution coding
It is a representation of feasible solution that obtained by
using the priority rule based heuristics. A feasible solution can
be represented by two n-element lists. First one is an activity
list in which each activity ‘j’ must occur after all its
predecessors and before all its successors. The second one is a
list of execution modes for all activities and is called mode
assignment in which the \( k_o \) element of this list defines the
execution mode of job \( k \) [9]. Solution coding can also be done
by other method e.g mode assignment.

5.2.4 Neighbourhood generation mechanism
A neighbour of a solution is generated using one of the three
operators:

(i) A \textit{neighbourhood shift} which operates only on the list
of activities.

(ii) A \textit{mode change} which operates only on the mode
assignment.

(iii) A \textit{combined move} which operates simultaneously on
both structures and is a composition of the neighbourhood
shift and the mode change.

5.2.5 Cooling schedule
It is defined as a fixed number of visited solutions. It
concerns precedence and resource feasible solution only.
In cooling schedule the control parameter will be
decremented. Often the most difficult step in the
annealing process is the development of an appropriate
cooling schedule. To ensure the success of the
optimization, the temperature (control parameter) must
be controlled so that it is large enough to move off a local
minimum, but small enough not to move off a global
minimum. Due to the wide variety and complicated
nature of most combinational optimization problems, the
suitable cooling schedule will be unique for each
problem. Ideally, the temperature should be lowered slow
ever to ensure that a good minimum is achieved, but
also quick enough so that computational time is
minimized.

5.3 Genetic Algorithms
Genetic Algorithm (GA) is inspired by the process of
biological evolution [22]. In contrast to the local search
strategies above, a GA simultaneously considers a set or
population of solutions instead of only one. They work with
a population of individuals, each representing a possible
solution to a given problem. Each individual is assigned a
fitness value (i.e it measure quality of solution) according to
how good a solution to the problem it is. The highly fit
individuals are given opportunities to reproduce, by mating
two existing ones (crossover) and/or by altering an existing
one (mutation) in the population. This produces new
individuals as offspring, which share some features taken
from each parent. The least fit members of the population are
less likely to get selected for reproduction, and so die out.
In this way, over many generations, good characteristics are spread throughout the population, being mixed and exchanged with other good characteristics as they go. If the GA has been designed well, the population will converge to an optimal solution to the problem.

5.3.1 Procedure of Genetic Algorithm

<table>
<thead>
<tr>
<th>Step(i) (Start)</th>
<th>Generate random population of n chromosome (suitable solution for the problem)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step(ii) (Fitness)</td>
<td>Evaluate the fitness f(x) of each chromosome ‘x’ in the population.</td>
</tr>
<tr>
<td>Step(iii) (New population)</td>
<td>Create a new population by repeating following step (iv-vii) until the new population.</td>
</tr>
<tr>
<td>Step(iv) (Selection)</td>
<td>Select two parent chromosomes from population according to their fitness (the better fitness, the bigger chance to be selected).</td>
</tr>
<tr>
<td>Step(v) (Crossover)</td>
<td>With a crossover probability cross over the parents to form new offspring (children) if no cross over was performed offspring is an exact copy of parents.</td>
</tr>
<tr>
<td>Step(vi) (Mutation)</td>
<td>With a mutation probability mutate new offspring at each locus (position in chromosome).</td>
</tr>
<tr>
<td>Step(vii) (Acceptance)</td>
<td>Place new offspring in the new population.</td>
</tr>
<tr>
<td>Step(viii) (Replacement)</td>
<td>Use new generated population for a further run of the algorithm.</td>
</tr>
<tr>
<td>Step(ix) (Test)</td>
<td>If the end condition is satisfied, then GA stop, otherwise it returns to the best solution in the current population and goes to step(ii).</td>
</tr>
<tr>
<td>Step(x) (End)</td>
<td>Print out the result.</td>
</tr>
</tbody>
</table>

5.3.2 Representation

When solving any problem by genetic algorithm a potential solution to the problem must be represented as a set of decision parameters. These parameters (known as genes) are joined together to form a string of values (often referred to as a chromosome). In RCPSP the following representations have been used

- **Activity List Representation**
  - Random Key Representation
  - Priority Rule Representation
  - Shift Vector Representation
  - Schedule Scheme Representation

**Chromosome:**
A chromosome describes a string of a certain length where all the genetic information of an individual is stored. Although nature often uses more than one chromosome, most GA only uses one chromosome of encoding the genotypic information. Each chromosome consists of many alleles. An allele is the smallest information unit in a chromosome. A gene is a region on a chromosome that must be interpreted together and which is responsible for a specific phenotypic property. Figure (below) illustrates the difference between the chromosome, gene, and allele.

5.3.3 Decoding fitness function

Decoding means development of a schedule from the genome. The decoding is done in the following manner as given below.
• First the one element is selected from the initial population to be decoded. First this sequence/element is assigned in the earliest starting manner. The completion time of the last activity gives the completion time of the project, i.e. project make span or first objective.
• Secondly the same sequence/element from the population is assigned in the latest finishing manner, i.e. backward scheduling is done.
• The difference between, completion time in backward and forward scheduling for any activity give the slack associated with that activity. The summation of slacks for all activities gives the second objective.

5.3.4 Crossover
• Crossover is inspired by the role of sexual reproduction in the evolution of living things. GA attempts to combine elements of existing solutions in order to create a new solution, with some of the features of each "parent." The elements (e.g. decision variable values) of existing solutions are combined in a "crossover" operation, inspired by the crossover of DNA strands that occurs in reproduction of biological organisms.
• Crossover combines the features of two parent chromosomes to form two offspring which inherit their characteristics. Crossover takes two individuals, and cuts their chromosome strings at some randomly chosen position, to produce two head segments, and two tail segments. The tail segments are then swapped over to produce two new full length chromosomes. The two offspring each inherit some genes from each parent. Crossover can also produce offspring of low fitness, but these will not be likely to get selected for reproduction in the next generation (Beasley et. al. 1993).

There are four types of crossovers:
(1) Single point crossover
(2) Two point crossover
(3) Uniform crossover
(4) Arithmetic crossover

5.3.5 Mutation
Mutation is inspired by the role of mutation of an organism’s DNA in natural evolution. Once the crossover operator has been applied and the offspring population has replaced the parent population, the mutation operator is applied to the offspring population. Mutation alters one or more genes (positions) of selected chromosomes (solution) to reintroduce lost genetic material and introduce some extra variability in the population. There are many possible ways to perform a "mutation". The mutation Operator suddenly changes some parts of the offspring.

5.4 Ant Colony Optimization
In the early 1990s, ant colony optimization (ACO) was introduced by M. Dorigo and colleagues as a novel nature-inspired meta-heuristic for the solution of hard combinatorial optimization (CO) problems.

Ant Colony Optimization (ACO) studies artificial systems that take inspiration from the behavior of real ant colonies [17, 22].

It is well known that the main means used by ants to form and maintain a path is a pheromone trail. Ants deposit a certain amount of pheromone while walking, and each ant probabilistically prefers to follow a direction rich in pheromone rather than a poorer one. This elementary behavior of real ants can be used to explain how they can find the shortest path that reconnects a broken line after the sudden appearance of an unexpected obstacle in the initial path [22].

In fact, once the obstacle has appeared, those ants which are just in front of the obstacle cannot continue to follow the pheromone trail and therefore they have to choose between turning right or left.

Fig-1 : The ants adding the pheromones to the trail.

In this situation we can expect half the ants to choose to turn right and the other half to turn left. The very same situation can be found on the other side of the obstacle.

It is interesting to note that those ants that choose, by chance, the shorter path around the obstacle will more rapidly reconstitute the interrupted pheromone trail compared to those that choose the longer path. Hence,
5.4.1 ACO Algorithm for RCPSP

According to [18] ACO approach is to use the ant algorithm for finding an activity list that gives a good schedule when used by Serial schedule generation scheme (SGSS) or Parallel schedule generation scheme (PSGS).

ACO Algorithm has following steps as:

Step1: Generate the schedule based on the state transition rule

For the selection of an activity the ant uses heuristic information as well as pheromone information. The heuristic information, denoted by \( n_{ij} \), and the pheromone information \( z_{ij} \), are indicators of how good it seems to put activity at place of the activity list for the SSGS or PSGS. The heuristic value is generated by some problem-dependent heuristic and the pheromone information stems from former ants that have found good solutions. The activity is chosen according to the probability distribution over the set of eligible Dg activities determined either by direct evaluation or by summation evaluation. Actually the only difference between these two pheromone evaluation methods is that the second pheromone evaluation methods will give weighted value to each pheromone value in order to get rid of the local minimum.

Step2: Pheromone Updating

The best solution found so far and the best solutions found in the current generation are used to update the pheromone information. Some portion of pheromone is evaporated according to:

\[
Z_{ij} = (1 - \rho)Z_{ij,\text{old}}
\]

Where: ‘\( \rho \)’ is the evaporation rate. The reason for this is that old pheromone should not have too strong an influence on the future. Due that reason, for every activity ‘\( j \)’ some amount of pheromone is added to element \( z_{ij} \), where ‘\( i \)’ is the place of activity ‘\( j \)’ in the activity list of the best solution found so far. This is an elitist strategy that leads ants to search near the best solution. The amount of pheromone added is \( \rho/2T \), Where ‘\( T \)’ is the makespan of the best-found schedule, i.e.,

\[
Z_{ij,\text{new}} = Z_{ij,\text{old}} + \frac{\rho}{2T}
\]

Due to that updating assist ants to find a good solution mark their paths through the decision space by putting some amount of pheromone on the edges of the path. The following ants of the next generations are attracted by the pheromone so that they search in the solution space near previous good solutions. The algorithm runs until some stopping criterion is met, e.g., a certain number of generations has been done or the average quality of the solutions found by the ants of a generation has not changed for several generations.

5.5 Tabu Search

Tabu Search (TS), developed by Glover and Greenberg (1989), is essentially a steepest descent/mildest ascent method, often called Best Fit Strategy (BFS), which scans the neighborhood and then accepts the best neighbour solution, until none of the neighbours improves the current objective function value.

The aim of these methods is to progressively transform a current solution to a new and perhaps better solution. So, TS can accept to deteriorate the current solution to avoid getting trap in local optimal by accepting both non-improving and infeasible solutions. This concept, however, bears the possibility of cycling, that is, one may always move back to the same local optimum one has just left. In order to avoid this problem, a tabu list is set up as a form of memory for the search process. Usually, the tabu list is used to forbid those neighborhood moves that might cancel the effect of recently performed moves and might thus lead back to a recently visited solution.

5.5.1 Tabu Search method

It can be briefly outlined as follows:

Step1: Obtain a suitable heuristic.

Step2: Obtain neighbourhood solution.

Two simple moves for obtaining a neighbourhood solution (schedule) of initial solution are:

(i) Activity swap
(ii) Activity Insert

Step3: Select best one neighbourhood solution. If cycling occur then selected move is not allowed to be reversed for a certain iteration with help of tabu list. However, in case a
better solution than the current best is found then such solution will be selected regardless of its Tabu list status.

**Step4:** a tabu status is overrun if the corresponding neighborhood move would lead to a new overall best solution, otherwise the process is repeated.

**Conclusion**

Exact methods are involved with finding the optimal solutions. However they are impractical when faced with problems of any significant size or with large sets of constraints. So, for these problem heuristics algorithms are adopted which give near optimal solution.

Although priority rules based heuristic methods do not give the best results because of several reasons. First, they are indispensable when solving large problem instances in a short amount of time. Second, priority rule based heuristic methods are needed to determine the initial solution for meta-heuristic procedures.

Hence, meta-heuristic procedures are best to find near-optimal solutions for large problem instances quickly. All metaheuristics have common features to find near-optimal solutions, as following:

They generating a new solution $s'$ from current solution $s$ and checking whether one should stop there or accept $s'$ to substitute $s$ to perform the next iterative step. The new solution $s'$ is selected among the solutions in a neighbourhood $N(s)$ that is defined for each solution $s$. The aim of metaheuristics for given a real valued function ($f$), is to find some solutions $s^*$ in solution space $S$ such that $f(s^*)$ is acceptable with respect to some criterion.

**References**


