Breadth-first and Best-first Exact Procedures for Regular Measures of the Multi-mode RCPSP

Madhukar Dayal
Sanjay Verma

W. P. No. 2014-10-04
November 2014

INDIAN INSTITUTE OF MANAGEMENT
AHMEDABAD – 380015
INDIA

Breadth-first and Best-first Exact Procedures for Regular Measures of the Multi-mode RCPSP

Madhukar Dayal
Indian Institute of Management Indore
e-mail: madhukar@iimidr.ac.in
Sanjay Verma
Indian Institute of Management Ahmedabad
e-mail: sverma@iimahd.ernet.in

Abstract

The multi-mode resource constrained project scheduling problem (MM RCPSP) is a NP-hard problem representing a generalization of the well-studied RCPSP. Depth-first tree search approach by Sprecher & Drexl (1998) is the best known exact solution tree search procedure for this problem. In this paper we present two exact solution single-processor approaches: a breadth-first approach and a best-first monotone heuristic. The comparison with depth-first and CPLEX show promising results on small problem sets. We report extension of the breadth-first approach to yield exact multi-objective solutions for the PSPLIB (Project Scheduling Problem Library, Kolisch & Sprecher, 1997) problem sets which is the first of its kind.

Keywords:

project scheduling, regular measures, exact multi objective solutions, breadth-first tree search, best-first monotone heuristic.

1 Introduction

The RCPSP is a well-studied problem with a large number of approaches developed for solving it, which can be classified as – heuristic, metaheuristic, and exact solution approaches. The *depth-first* branch and bound approach by Sprecher & Drexl (1998) is known to be the fastest tree search exact solution approach. No approach exists which yields exact solutions to concurrent multiple objectives (i.e. *exact multi objective solution*).

In this paper we present exact algorithms for optimally scheduling partially ordered multi-mode activities under resource constraints, known as the Multi-mode Resource Constrained Project Scheduling Problem (MM RCPSP). We consider, both, renewable and non-renewable resources and develop single processor breadth-first and best-first algorithms for exact solutions to a single objective. We also extend our breadth-first tree-search approach, which yields multiple exact optimal solutions for a single objective, to yield the *exact multi objective optimal solution*.

2 LITERATURE REVIEW

While small projects and shop floor scheduling problem instances may be solved using available exact approaches, large problem instances, being complex for the human mind to comprehend or computer to solve, are dependent on heuristics. The pursuit of one or more of several desirable objectives, simultaneously, enhances the complexity of the problem further.

2.1 Inexact (Heuristic and Metaheuristic) Approaches

Several heuristic and metaheuristic approaches have been presented in literature to solve large scheduling problems. However, these approaches do not guarantee the yield of an optimal solution. Usually, these approaches deploy one or more checking procedures for termination of the algorithm, such as, acceptable limit on minimum percentage improvement from previously found best solution, run time bounds, and/or the number of iterations limit. In these approaches, it is possible that in multiple runs of the same algorithm using same termination criteria, and on the same problem instance and computing machine, an inferior or superior result is obtained. This clearly establishes the need for improved exact algorithms for finding the exact solutions to such problems. However, the research approaches pursuing inexact or approximate solutions are many times more than that for exact solutions. Our research attempts to cover this gap.

Heuristic approaches for MM-RCPSP have been presented by Berman (1964), Leachman (1980), and Leachman, Dincerler, and Kim (1990). Talbot (1982) presented an integer programming model for MM-RCPSP with an optimal approach for solving small problems and a heuristic to solve large instances of the problem. A study of three

enumeration based approaches by Patterson (1984) reveals that "*each procedure was found to be generally superior on a specific class of problems*". Drexl and Gruenewald (1993) studied a problem with renewable, non-renewable, and doubly constrained resources involving a time-resource trade off, in which activities can be performed in discrete modes and propose a stochastic scheduling heuristic procedure which performs better than available routines.

Metaheuristic approaches have been most widely deployed to solve the MM-RCPSP in the literature. Genetic algorithm (GA) approaches have been applied by Mori and Tseng (1997), Reddy, Kumanan and Krishnaiah (2001), Ulusoy and Sahin (1998) and Ulusoy, Funda and Sahin (2001), Hartmann (2001), Alcaraz, Maroto and Ruiz (2002; 2003a; 2003b), and more recently by Peteghem and Van Houcke (2008). Józefowska, Mika, Rózycki, Waligóra, and Weglarz (1999; 2001), and Bouleimen and Lecocq (2003) have applied the Simulated Annealing (SA) approach to solve this problem. De Reyck, Demeulemeester, and Herroelen (1998), De Reyck and Herroelen (1999), and Mika, Waligóra, Weglarz, (2008) have used Tabu search; Chyu, Chen and Lin (2005) and Shan, Hong and Juan, (2007) have applied Ant Colony (AC) approaches; and Shan, Wu and Peng (2007) have used Particle Swarm (PS) optimization to solve the MM-RCPSP.

Metaheuristic approaches are able to tackle problems of large size and appear to be very promising for further research too. Typically they terminate when the improvement in the objective reaches a pre-determined threshold or a limit on specified time and/or number of iterations has been reached. However, they do not guarantee an optimal solution even at the expense of phenomenal computational processing power, memory and time. For large problems, applying the same algorithm to the same problem instance for the same duration may also not yield the same objective function value in each run. Much research effort is needed to improve available techniques to yield optimal solutions to even the smallest real life problem instances.

Schirmer (1996) established that the MM-RCPSP is NP-complete and requires at least exponential time in its binary formulation. Kolisch and Sprecher (1997) point out that exact methods can only solve small problem instances and heuristic approaches may fail to generate feasible solutions, if one exists, for highly resource constrained problems. They also prove that the MM-RCPSP *feasibility* problem itself is NP-complete, while Alcaraz, Maroto, and Ruiz (2003a) established that the optimization problem is NP-hard. Kolisch, Sprecher, and Drexl (1995) have observed that as "*the number of variables and constraints grows rapidly*", MILP approaches are of limited effectiveness. Thus, the solution to problems of sizes as are encountered in real life has remained elusive, and in particular, the pursuit of exact algorithms appears to have been ignored.

2.2 Exact Approaches

Developing a mode alternative, similar to Demeulemeester and Heroelen's (1992) delay alternative, and applying a B&B procedure with search tree reduction scheme, Sprecher, Hartmann, and Drexl (1997) and Sprecher and Drexl (1998) presented

algorithms for obtaining an exact solution to the MM-RCPSP. Hartmann and Drexl (1998) compare three B&B approaches for the MM-RCPSP and conclude that the precedence tree guided enumeration scheme performs the best. A B&B depth-first procedure for obtaining optimal solution and its truncated version are presented by Sprecher and Drexl (1998) for obtaining exact solutions and tested on a large number of problem instances. This approach remains the best exact approach till date. They also discuss the impact of variation of several project characteristics on solution time and quality.

Erenguc, Ahn and Conway (2001) presented an integer programming model and an exact solution B&B procedure adopting branching rules, minimal resource conflict sets, and node fathoming rules for improving efficiency. Heilmann (2001, 2003) has presented another exact B&B approach for small instances and a priority rule based heuristic approach for larger instances of the MM-RCPSP. Sabzehparvar and Seyed-Hosseini (2008) studied the problem in a mode dependent time lag environment and presented an exact algorithm. They relate the problem to a bin-packing problem and present its mixed-integer programming formulation. They also presented a geometric formulation of the problem and a B&B approach to obtain solutions to the problem instances tested.

MM RCPSP, Regular Measures: Sabzehparvar and Seyed-Houseini (2008) pointed out that the best known depth-first B&B exact algorithm by Sprecher and Drexl (1998) is capable of solving only small instances in a reasonable time.

A breadth-first exact solution approach, and corresponding best-first approach was presented by Nazareth, Verma, Bhattacharya, and Bagchi (1999) for the single-mode case. Extending their work we study these approaches for the multi-mode case. For a more detailed review of multi objective approaches we refer the reader to Ballestin and Blanco (2011).

3 PROBLEM STATEMENT

A project has been defined in many research papers and books. We reproduce the description provided in Nazareth et. al. (1999) in rest of this section.

A *project* is a set of activities which are partially ordered by precedence relationships. An *activity* can be performed in finite number of modes, where each mode is unique and has a corresponding non-negative duration. An activity is ready to be processed only when all its predecessor activities are completed *and* the number of units of the various resource types required by it, in the mode that it is to be performed, are free and can be allocated to it. Once started, an activity is not interrupted (*non-preemptive*) and runs to its completion. The dummy (start and end) activities consume no resources and take no time. For each of its *modes*, an activity uses different types of resources, such as manpower and machinery, in different amounts, which are specified in advance. A mode specifies an activity's resource requirements of each resource type and its duration in

that mode. A *resource* is an essential facilitator for an activity to be performed. It may be durable (*renewable*) or consumable (*non-renewable*). The resources are allocated exclusively to a single activity for its entire duration in the selected mode. A resource may also be *doubly constrained*, i.e. it has an overall limit of availability for the whole project, as well as, time period wise limit of consumption for each time period. The availability of each resource type is known in advance. After completion of an activity, renewable resources may be assigned to another activity, whereas, the amounts of non-renewable and doubly constrained resources decrease by the respective amounts of each of these resources consumed in completion of the activity in its assigned mode, and only the residual amounts can be used further.

Scheduling is the process of selecting the mode and committing resources to the realization of each activity, while meeting the precedence and resource restrictions, to optimize a given objective. The aim is to assign modes and start times to all activities so that the desired objective (for example, makespan, flowtime, maximum tardiness, number of tardy jobs, etc.) is optimized.

The objective to be optimized may be *regular* or *non-regular*. Regular measures are those measures for which no performance improvements will occur with delay in start of the activities, for example, minimizing completion time or minimizing tardiness. Non-regular measures are those measures for which the performance may improve with delay in start of the activities, for example, in objectives like minimizing the earliness-tardiness in just in time (JIT) and maximizing net present value (NPV).

In the case of non-regular measures (NPV), every mode of each activity has an associated cash flow (either inflow or outflow) at the start of the activity, for each unit time of its duration in the selected mode, and at the end of the activity. The objective in this case is to schedule all activities such that the NPV of their cash flows, at the given rate of interest per period, is maximized.

3.1 Definitions of Terms

Extending from Nazareth, et al.(1999), we now define the terms we use in the studied problem model.

> **Project:** A project consists of N activities a_1, a_2,..., a_i,..., a_N. Activity a_i has j modes a_{i1}, a_{i2},..., a_{im},..., a_{ij}. Activity i in mode j, a_{ij}, has a duration of p_{ij} units, which includes the set-up time, processing time and set-down time. We use the Activity-on-Node (AON) convention when discussing an algorithm, though Activity-on-Arc (AOA) convention can be used without loss of generality. The dummy start activity a_1 and dummy end activity a_N have only one mode each.
>
> **Precedence Constraints:** Activity a_i (i = 1,..., N) can start only when all its predecessor activities have finished. P_i is the set of predecessors of a_i. Predecessors of an activity are usually determined by technological considerations of the project. An activity a_p is said to be a predecessor of a_i,

when a_i can not start until a_p has finished. This is represented as $a_p<a_i$, where the symbol '<' defines the 'precedes' relationship. Similarly a_s is said to be a successor of activity a_i if a_s cannot begin until a_i has finished, i.e. $a_i<a_s$. S_i represents the set of successors of activity a_i.

Renewable Resource Types: M types of renewable resources are available for the project's completion. R_j (j = 1,..., M) denotes the total availability in number of units of resource type j. Activity i in mode j, a_{ij}, requires r_{ijm} units of the m^{th} renewable resource.

Non-renewable Resource Types: K types of non-renewable resources are assumed to be available. L_k (k = 1 to K) denotes the total availability in number of units of non-renewable resource type k. Activity i in mode j, a_{ij}, requires l_{ijk} units of the k^{th} non-renewable resource. Note that on completion of an activity, the non-renewable resources allocated to it are consumed and only residual amounts of these resources are available for further allocations.

Resource Constraints: The renewable resource constraints emphasize that the total units of renewable resource type j used by all the activities in progress at any instant of time should not exceed the total availability of that renewable resource, R_j. The non-renewable resource constraints imply that the total number of units of non-renewable resource type k consumed by the activities completed and allocated to the activities in progress should not exceed the total availability of that non-renewable resource, L_k.

Integrality Condition: The values of parameters such as an activity's duration (processing time) in a mode (p_{ij}), resource availabilities (R_j and L_k) and resource requirements (r_{ijm} and l_{ijk}) are non-negative integers.

Non-preemption Constraint: Once an activity starts it proceeds until its completion, i.e., no activity can be pre-empted. If s_i denotes the start time and f_i the finish time of activity i in mode j, a_{ij}, then $s_i + p_{ij} = f_i$.

A project has two dummy activities, a unique dummy start activity a_1 and a unique dummy finish activity a_N. These activities have a single mode. The duration of these activities is zero and they do not consume any renewable or non-renewable resources. In symbols, $p_1 = p_N = 0$, $r_{1j} = r_{Nj} = 0$ for all renewable resources j = 1,..., M, and $l_{1k} = l_{Nk} = 0$, for all non-renewable resources k = 1,..., K.

Every non-dummy activity has at least one predecessor and at least one successor, i.e., $a_1 \in P_i$ (i = 2,..., N) and $a_N \in P_i$ (i = 1,..., N-1). In other words, the dummy start activity, a_1, is a predecessor to all other activities. Further, the dummy end activity, a_N, is a successor to all activities. Note that these may not be *direct* predecessors or successors to all activities.

For any activity, if its direct predecessors have completed, all indirect predecessors must have completed, thus, only direct predecessors of an activity need to be listed when needed.

The project starts at time t = 0, i.e. $s_1 = 0$. An activity is said to be scheduled when it is assigned a mode and a start time, and renewable and non-renewable resources needed in its selected mode are available and allocated to it. A schedule for the project is an assignment of a mode and a start time to each activity of the project. The time when last activity of the project, a_N, finishes represents the *makespan* of the project, $T = f_N$. A *feasible schedule* is a schedule that satisfies the given precedence and resource constraints. An *optimum schedule* is a feasible schedule that optimizes the given objective function. The scheduling problem is to determine an optimum schedule, given the resource availabilities R_j (j =1,..., M), L_k (k = 1,..., K), processing times of all activities in each of their modes p_{ij}, predecessor sets of all activities P_i, and renewable and non-renewable resource requirements r_{ijk}, l_{ijl}, for all resources, for each activity, in each of its modes.

3.2 Mathematical Formulation

The MM-RCPSP can be mathematically formulated as follows:

Min f_N (finish time of the dummy end activity) (1)

such that

$f_i - f_j \geq p_{im}, 1 \leq j < i \leq N, j \in P_i$ (processing time constraints) (2)

$\sum_{j=1}^{M} r_{imj} \leq R_j, 1 \leq j \leq M, 0 \leq t \leq f_N$ (renewable resource usage constraints) (3)

$\sum_{k=1}^{L} l_{imk} \leq L_k, 1 \leq k \leq L, 0 \leq t \leq f_N$ (non-renewable resource usage constraints). (4)

$i, j, k, m, L, M, R > 0$ (non-negativity constraints) (5)

$i, j, k, m, L, M, R \in I$ (Integrality constraints) (6)

Peteghem and Vanhoucke (2010) have presented a conceptual mathematical formulation of the problem, while Sabzehparvar and Seyed-Hosseini (2008) have pointed out that "*The most efficient method for solving this problem known so far is the algorithm of Sprecher and Drexl...*" referring to the depth-first branch and bound algorithm for MM-RCPSP with which we compare our algorithms. They also present an efficient MILP formulation, as well as a geometric formulation of the problem and present an algorithm for the version with minimal and maximal time lags. We refer the interested reader to these for the complete integer linear formulation of the problem.

1 PRE-PROCESSING RULES

The pre-processing rules adopted before a problem instance can be processed are: (a) the removal of infeasible modes (checked for renewable and non-renewable resources), and (b) the removal of inefficient or inferior modes. Sprecher, Kolisch, and Drexl (1995) have explained these pre-processing rules in great detail. We provide a description of the pre-processing below.

Infeasible Modes: A mode m of an activity i is renewable resource infeasible with respect to the renewable resource r, if the units of r required by i in mode m are greater than the available units of resource r. Such a mode is excluded from consideration in generating the schedules.

A mode m of an activity i is non-renewable resource infeasible on account of a non-renewable resource n, if the sum of the minimum amounts of n required by all *other* activities together, in those of their modes which consume least units of n, and the units of n needed by i in mode m, is greater than the availability of the resource n. Such a mode of an activity, if deployed, will leave inadequate residual of the non-renewable resource n for the project to be completed, and hence, is an infeasible mode of that activity.

Inferior Modes: A mode m of an activity i is inferior to another mode of i, say mode l, if the duration of i in mode m is same or greater than the duration in mode l, and for none of the renewable, as well as, non-renewable resources, the units needed by i in mode m are less than corresponding resource units needed in mode l. That is, while the duration is same or greater, all the resource requirements are also same or greater. Such modes, identified as *inferior modes*, may be removed from consideration. It is possible that when an abundant non-renewable resource is removed from consideration (explained in the next section), one mode of an activity becomes inferior to another mode, as it completes in (same or) greater duration, while it also consumes (same or) more amounts of all remaining resources, but less amount of only the non-renewable resource now eliminated from consideration due to its abundance. Such an inferior mode can be removed from consideration. At any stage of pre-processing, if identical modes are revealed, only one needs to be retained to reduce computational requirements.

Redundant Non-renewable Resources: A non-renewable resource n, is said to be redundant if the sum of the largest quantities of n needed by all activities, in their modes consuming largest amounts of n, is less than the total availability of n. Such a resource is available in abundance and hence may be removed from consideration to save computational effort.

It is noteworthy, as pointed out by Sprecher, Kolisch, and Drexl (1995), that during pre-processing, a mode of an activity may become inferior or identical to another mode. Such modes would need removal from consideration, and on their removal, it is further possible that yet another non-renewable resource is rendered abundant. Hence, a carefully implemented repeated check of the whole problem instance is necessary. The

structure of pre-processing rules for renewable resources and non-renewable resources is explained in pseudo codes below.

Algorithm Pre-processing

Renewable resource feasibility check:
Step 1 (Loop) for each activity do
Step 2 (Loop) for each of its mode do
Step 3 (Loop) for each renewable resource do
Step 4 If mode is not renewable resource feasible, delete mode
Step 5 If feasible mode for activity is not found exit with error message

Non-renewable resource feasibility check:
Step 1 (Loop) for each activity do
Step 2 (Loop) for each non-renewable resource do
Step 3 determine min need of non-renewable resource for all other activities
Step 4 (Loop) for each of activity's mode do
Step 5 If mode non-renewable resource feasible with all other activities' minimum requirements then retain this mode
Step 6 if activity has no feasible mode, exit with error

Figure 1: Example 1 Problem – Pre-processing Rules

Abundance check for non-renewable resources is similarly implemented. Now, consider the example problem instance, Example 1, in Figure 1. R1 and R2 are the two renewable resources and NR1 and NR2 are the two non-renewable resources in the problem instance. In the figure, the two numeric subscripts to letter 'a', respectively, denote the *activity number* and the *activity mode*. The maximum availabilities of the two renewable resources (R1 and R2) and two non-renewable resources (NR1 and NR2) in the problem instance are indicated at the top right in the diagram. The activity numbers for the problem instance are denoted inside the circles as in AON convention. For each mode of the activity, its duration, renewable resource requirements, and non-renewable resource requirements are detailed below the circle representing the activity according to the given legend.

Figure 2: Partially Reduced Example 1 Project for Pre-processing Rules

The activity two in its first mode, a_{21}, needs five units of RR2, whereas, only four units are available. Hence, this activity mode combination is infeasible, and therefore, removed from consideration. For convenience, and without loss of generality, in its place the old second mode of this activity is now treated as its first mode. Further, activity four in its second mode, a_{42}, needs seven units of RR2, while only four units are available. Hence, it is an infeasible mode and is removed from consideration. After the above two reductions in Example 1, the maximum requirement for NR2, considering the individual maximum requirements for each activity from their remaining feasible

modes, is reduced to (0 + 6 + 5 + 9 + 7 + 7 + 0 = 34) thirty-four units. As sufficient quantity of NR2 is available for performing any activity in any mode (abundant resource), we remove NR2 from consideration totally. The partially pre-processed, reduced problem instance is presented in Figure 2.

It can be clearly seen from the two modes for activity six, one is inferior as it consumes same (or more) duration, and (same or) more amounts of all renewable resources and non-renewable resources. Hence, the second mode of activity six is removed from consideration, too. After its removal, the maximum requirement for NR1 is now reduced to thirty-four units, which is less than its availability (thirty-six units). NR1 is now available abundantly, and therefore, it is also removed from consideration. Example 1 is a case to illustrate the implementation and effect of the pre-processing rules, however, in randomly generated problem instances, a situation in which all non-renewable resources are completely removed seldom occurs.

2 SINGLE PROCESSOR BREADTH-FIRST ALGORITHM

The algorithm is presented now. First the basic algorithm is presented following by pruning rules.

2.1 Algorithm Without Pruning Rules

The breadth-first algorithm is a tree-search procedure with pruning rules. The nodes in the search tree correspond to the partial schedules. A *partial schedule* is a schedule of a subset of the N activities that does not violate any of the given precedence and resource constraints. A *complete schedule* is a partial schedule of all the N activities, and a state corresponding to a complete schedule is a *solution state*. The starting node or root node of the search tree corresponds to a state where no activity has been completed and the (dummy) start activity, a_{11}, is in progress. Complete information of a state comprises:

c_x Current time: The time of creation of state X, represents the earliest time at which the processing of an activity in progress was completed in the parent state of X and a scheduling decision was made.
F_x Finished set: The set of activities that have already finished at or before time c_x without violating any precedence or resource constraints.
A_x Active set: The set of activities which started at time $\leq c_x$, either in state X or in some ancestor state of X, and have not finished before c_x. This is the set of activities in progress in state X at time c_x.
dp_x Decision point: This represents the earliest time at which the processing of an activity in the active set A_x is completed. It becomes the current time (i.e. the decision point) for each child state of state X.
K_x Decision set: The set of activities which are not yet completed at time dp_x, but all of whose predecessors have completed at some time $\leq dp_x$. This is the set of candidate activities.
Thus, the root state I, of the search tree is as follows:

$c_I = 0$ = current time of root state.
$F_I = \{\ \}$ = Set of activities completed at c_I, empty set.
$A_I = \{a_{11}\}$ = Set of activities in progress at time c_I, which is only the dummy start activity.
$dp_I = 0$ = the decision point for state I, is the finish time of activity a_{11}.
NR_I = Amounts of non-renewable resources consumed by the completed activities (here, zero for each non-renewable resource).
K_I = Set of activities that become ready when activity a_{11} completes, i.e. the candidate activities.

Let the level of a state X in the search tree be the number of activities in the finished set, F_X. At the time of start of the algorithm, the tree consists of only the root state I at the level = 0, where level zero represents the number of activities completed (which is zero for level 0, where the root state resides). We are interested in developing the next level (level 1), for which we need to expand all of the previous level's state(s), one by one. Subsequently, we are interested in generating the states at all levels up to the last level and obtain the solution.

States get added by selecting the first parent state, which is a partial solution state, at the last completed level, generating all its child states, and proceeding to the next parent state. The decision point dp_x has to be determined first, followed by the set of candidate activities, K_x. A *resource satisfying set* (RSS) is a subset of candidate activities (the decision set), including their assigned modes, that does not cause a resource violation. A *maximal resource satisfying set* (MRS) is such a set of candidate activities, with their assigned modes, to which no other candidate activity in any of its modes can be added without violating a resource constraint. The concept of RSS and MRS is explained below through the problem instance Example 2, shown in Figure 3.

Figure 3: Example 2 Project for Explaining RSS and MRS

On completion of the (dummy) start activity, a_{11}, the candidate activities eligible for scheduling are activities two, three, and four. Note that activity three in mode one, $\{a_{31}\}$, is infeasible as eight units of renewable resource R1 are needed against four units maximum available. Thus, all the feasible **RSS** are: $\{a_{21}\}$, $\{a_{21}, a_{32}\}$, $\{a_{32}\}$, and $\{a_{41}\}$. None of these violates the renewable or non-renewable resource availabilities. Amongst these, $\{a_{21}, a_{32}\}$ and $\{a_{41}\}$ are such RSS with which no other candidate activity can be scheduled due to resource limitations. Hence, these are maximal resource satisfying sets, i.e. **MRS**.

On completion of the (dummy) start activity, a_{11}, the candidate activities eligible for scheduling are activities two, three, and four. Note that activity three in mode one, $\{a_{31}\}$, is infeasible as eight units of renewable resource R1 are needed against four units maximum available. Thus, all the feasible **RSS** are: $\{a_{21}\}$, $\{a_{21}, a_{32}\}$, $\{a_{32}\}$, and $\{a_{41}\}$. None of these violates the renewable or non-renewable resource availabilities. Amongst

these, {a_{21}, a_{32}} and {a_{41}} are such RSS with which no other candidate activity can be scheduled due to resource limitations. Hence, these are maximal resource satisfying sets, i.e. **MRS**. Note that we cannot consider only the maximal resource satisfying sets when non-renewable resources are involved too, as this may lead to the optimal solution being missed. For explanation through a problem instance, consider the Example 3 problem instance shown in Figure 4. When this problem is solved using only the maximal resource satisfying sets, a sub-optimal solution is yielded for the problem. The solution yielded is shown in Figure 5.

Figure 4: Example 3 Project for Explaining why all RSS need Consideration

The wrong solution is yielded due to consideration of only the maximal resource satisfying sets, whereas, the correct optimal solution is yielded when a sub-maximal resource satisfying set (or just resource satisfying set) is considered. The correct optimal solution is shown further in Figure 6.

Figure 5: Wrong Solution Yielded when Considering only MRS

The development of only MRS forces the activity six to be performed in its first mode leaving insufficient non-renewable resource for activity seven to be scheduled in its shorter first mode later. The activity six is not considered in its second mode with activity two, as they are resource infeasible together. Further, when activities {a_{21},a_{61}}

Figure 6: Right Solution Yielded when Considering all RSS

are in progress, {a_{61}} completes first, hence it is not considered for retraction either. A contributing factor to the above dilemma appears to be the large difference in time taken for completion in the two different modes of the activity six.

When we consider all of the resource feasible activity mode combinations, as we do in our algorithm, the optimal solution (shown in Figure 6) is yielded.

Notice that after completion of activity three in its first mode, the activities two and four are in progress in their first modes, i.e. {a_{21},a_{41}}. This set is not a maximal resource satisfying set, in fact, it is only a resource satisfying set (an RSS). Disregarding the possible shift within available slack in non-critical path activities (i.e., {a_{21}}, {a_{51}} and {a_{62}}), this problem has a unique optimal solution!

Hence, we consider all resource feasible single activity and mode sets with which the tree could be expanded; also all two activities and their modes sets; and so on, up to and including all candidate activities with their modes, as long as they are resource feasible. For each of such an RSS, some of which are also MRS, a child state is developed (though some are pruned through the pruning rules described later, while retaining the optimality).

In every child state Y of a parent state X:
$c_Y = dp_X$, the current time of child state.
$F_Y = F_X$ augmented by the activities in A_X that completed at time dp_X;
A_Y = the RSS of X that corresponds to this child state.

The starting times of activities in A_Y are determined as follows. If an activity a_i that is in A_Y also belongs to A_X, and is in progress in the same mode as in A_X, then the starting time of a_i in state Y is the same as its starting time in state X, else its starting time is c_Y. If activity a_i in A_X did not complete at dp_X and now, does not belong to A_Y, it is as if it was never scheduled, and it is treated as retracted, similar to retraction in the algorithm of Demeulemeester and Herroelen (1992). Some problem instances may not be optimally solved, if retraction is not considered. We show this with an example later.

The search tree is generated level-by-level, i.e. in a breadth-first order, and all paths from the root state to a solution state are essentially of length N, the number of activities in the project. At any point, the states at two adjacent levels (one being developed further, and one being generated as new level) are needed to be stored. These states can be maintained in a linked list, thus, making the algorithm very simple. In practice, as more than one activities may complete together in a given state to be expanded, more than two adjacent levels may have to be maintained in memory.

The basic breadth-first algorithm, thus, is as follows.

Algorithm Breadth-first

Preprocessing	remove infeasible modes, redundant resources, and inferior modes
Step 1 (Initialization)	create the root state I at level 0 in the search tree
Step 2 (Loop)	for all levels L from 0 to N-1 do
	for each state X at level L do
Step 3 (Expansion)	determine dp_X
	construct K_X and all the RSSs
	generate a child node corresponding to each RSS
Step 4 (Termination)	Traverse the states at level N and output the complete schedule associated with the state X with minimum makespan at level N

Figure 7: Example 4 Project for Breadth-first Algorithm

Two activities, in selected modes, are compatible only if they can be processed simultaneously, not taking any other activity into account. This implies that these activities must not be related to each other in a successor-predecessor relationship, and when processed in their selected modes, they do not violate any resource constraint. Note that two activities which can be processed simultaneously, may not be compatible in all of their mode combinations with each other due to resource restrictions.

We present below a small example project, Example 4, solved using the above algorithm *without any pruning rules*. Later, we shall present the same problem instance solved again using only the Left Shift rule; using only the One Child Set pruning rule; using only the Dominance pruning rule; and finally with all the three pruning rules. This shall demonstrate the effectiveness of all pruning rules in obtaining the optimal solution while reducing computational effort. Consider the example problem instance Example 4 shown in Figure 7.

The project in Example 4 comprises five activities, two renewable resources, and two non-renewable resources. The availabilities of all resources are shown in the figure. The duration of activities in all of their modes and the corresponding resource requirements are shown as per the legend for each activity in all of its modes.

When the algorithm is applied without any pruning rules, a large number of states (a total of one hundred and thirty three for Example 4 problem instance) is generated. The complete enumeration of all states, level by level in breadth-first order, is given in Table 1 and is described in detail thereafter.

The columns respectively denote: the state number, the level at which the state is generated, the parent state's number, the earliest completion time of an activity in the state, the set of completed activities, and the set of activities in progress (at least one of which is completed at the earliest completion time of the state). Column 7 displays all of the resource feasible activity-mode sets (the RSS) which give rise to the child states of the current state.

Table 1: Breadth-first Search in Example 4 Project without Pruning Rules

(1)	(2)	(3)	(4)	(5)	(6)	(7)
State X	Level	Parent State	Decision Point (dpx)	Completed Actvs (Fx)	In Progress Actvs(Ax)	Resource feasible RSS
S1	0	--	0	--	{a11}	{a21}{a22}{a22,a32} {a31}{a32}
S2	1	S1	2	{1}	{a21}	{a31}{a31,a41} {a31,a42}{a32} {a32,a42}{a41}{a42}
S3	1	S1	4	{1}	{a22}	{a31}{a31,a41} {a31,a42}{a32}
						{a32,a41}{a32,a42} {a41}{a42}
S4	1	S1	4	{1}	{a22,a32}	{a31}{a31,a41} {a31,a42}{a32}{a32,a 41}{a32,a42} {a41}{a42}
S5	1	S1	3	{1}	{a31}	{a21}{a22}
S6	1	S1	6	{1}	{a32}	{a21}{a22}
S7	2	S2	5	{1,2}	{a31}	{a41}{a42}
S8	2	S2	5	{1,2}	{a31,a41}	{a41}{a42}
S9	2	S2	5	{1,2}	{a31,a42}	{a41}{a42}
S10	2	S2	8	{1,2}	{a32}	{a42}
S11	2	S2	8	{1,2}	{a32,a42}	{a42}
S12	2	S2	6	{1,2}	{a41}	{a31}
S13	2	S2	9	{1,2}	{a42}	{a31}{a32}
S14	2	S3	7	{1,2}	{a31}	{a41}{a42}
S15	2	S3	7	{1,2}	{a31,a41}	{a41}{a42}
S16	2	S3	7	{1,2}	{a31,a42}	{a41}{a42}
S17	2	S3	10	{1,2}	{a32}	{a41}{a42}
S18	2	S3	8	{1,2}	{a32,a41}	{a31}{a32}
S19	2	S3	10	{1,2}	{a32,a42}	{a41}{a42}
S20	2	S3	8	{1,2}	{a41}	{a31}{a32}
S21	2	S3	11	{1,2}	{a42}	{a31}{a32}
S22	2	S4	7	{1,2}	{a31}	{a41}{a42}
S23	2	S4	7	{1,2}	{a31,a41}	{a41}{a42}
S24	2	S4	7	{1,2}	{a31,a42}	{a41}{a42}
S25	2	S4	6	{1,2}	{a32}	{a41}{a42}
S26	2	S4	6	{1,2}	{a32,a41}	{a41}{a42}
S27	2	S4	6	{1,2}	{a32,a42}	{a41}{a42}
S28	2	S4	8	{1,2}	{a41}	{a31}{a32}
S29	2	S4	11	{1,2}	{a42}	{a31}{a32}

(1)	(2)	(3)	(4)	(5)	(6)	(7)
State X	Level	Parent State	Decision Point (dpx)	Completed Actvs (Fx)	In Progress Actvs(Ax)	Resource feasible RSS
S30	2	S5	5	{1,3}	{a21}	{a41}{a42}
S31	2	S5	7	{1,3}	{a22}	{a41}{a42}
S32	2	S6	8	{1,3}	{a21}	{a42}
S33	2	S6	10	{1,3}	{a22}	{a41}{a42}
S34	3	S7	9	{1,2,3}	{a41}	{a51}
S35	3	S7	12	{1,2,3}	{a42}	{a51}
S36	3	S8	6	{1,2,3}	{a41}	{a51}
S37	3	S8	12	{1,2,3}	{a42}	{a51}
S38	3	S9	9	{1,2,3}	{a41}	{a51}
S39	3	S9	9	{1,2,3}	{a42}	{a51}
S40	3	S10	15	{1,2,3}	{a42}	{a51}
S41	3	S11	9	{1,2,3}	{a42}	{a51}
S42	3	S12	9	{1,2,4}	{a31}	{a51}
S43	3	S13	12	{1,2,4}	{a31}	{a51}
S44	3	S13	15	{1,2,4}	{a32}	{a51}
S45	3	S14	11	{1,2,3}	{a41}	{a51}
S46	3	S14	14	{1,2,3}	{a42}	{a51}
S47	3	S22	11	{1,2,3}	{a41}	{a51}
S48	3	S22	14	{1,2,3}	{a42}	{a51}
S49	3	S15	8	{1,2,3}	{a41}	{a51}
S50	3	S15	14	{1,2,3}	{a42}	{a51}
S51	3	S23	8	{1,2,3}	{a41}	{a51}
S52	3	S23	14	{1,2,3}	{a42}	{a51}
S53	3	S16	11	{1,2,3}	{a41}	{a51}
S54	3	S16	11	{1,2,3}	{a42}	{a51}
S55	3	S24	11	{1,2,3}	{a41}	{a51}
S56	3	S24	11	{1,2,3}	{a42}	{a51}
S57	3	S17	14	{1,2,3}	{a41}	{a51}
S58	3	S17	17	{1,2,3}	{a42}	{a51}
S59	3	S25	10	{1,2,3}	{a41}	{a51}
S60	3	S25	13	{1,2,3}	{a42}	{a51}
S61	3	S18	11	{1,2,4}	{a31}	{a51}
S62	3	S18	10	{1,2,4}	{a32}	{a51}
S63	3	S26	8	{1,2,3}	{a41}	{a51}
S64	3	S26	13	{1,2,3}	{a42}	{a51}
S65	3	S19	14	{1,2,3}	{a41}	{a51}
S66	3	S19	11	{1,2,3}	{a42}	{a51}

(1)	(2)	(3)	(4)	(5)	(6)	(7)
State X	Level	Parent State	Decision Point (dpx)	Completed Actvs (Fx)	In Progress Actvs(Ax)	Resource feasible RSS
S67	3	S27	10	{1,2,3}	{a41}	{a51}
S68	3	S27	11	{1,2,3}	{a42}	{a51}
S69	3	S20	11	{1,2,4}	{a31}	{a51}
S70	3	S20	14	{1,2,4}	{a32}	{a51}
S71	3	S28	11	{1,2,4}	{a31}	{a51}
S72	3	S28	14	{1,2,4}	{a32}	{a51}
S73	3	S21	14	{1,2,4}	{a31}	{a51}
S74	3	S21	17	{1,2,4}	{a32}	{a51}
S75	3	S29	14	{1,2,4}	{a31}	{a51}
S76	3	S29	17	{1,2,4}	{a32}	{a51}
S77	3	S32	15	{1,2,3}	{a42}	{a51}
S78	3	S33	14	{1,2,3}	{a41}	{a51}
S79	3	S33	17	{1,2,3}	{a42}	{a51}
S80	3	S30	9	{1,2,3}	{a41}	{a51}
S81	3	S30	12	{1,2,3}	{a42}	{a51}
S82	3	S31	11	{1,2,3}	{a41}	{a51}
S83	3	S31	14	{1,2,3}	{a42}	{a51}
S84	4	S40	15	{1,2,3,4}	{a51}	--
S85	4	S41	9	{1,2,3,4}	{a51}	--
S86	4	S77	15	{1,2,3,4}	{a51}	--
S87	4	S34	9	{1,2,3,4}	{a51}	--
S88	4	S36	6	{1,2,3,4}	{a51}	--
S89	4	S38	9	{1,2,3,4}	{a51}	--
S90	4	S57	14	{1,2,3,4}	{a51}	--
S91	4	S59	10	{1,2,3,4}	{a51}	--
S92	4	S63	8	{1,2,3,4}	{a51}	--
S93	4	S65	14	{1,2,3,4}	{a51}	--
S94	4	S67	10	{1,2,3,4}	{a51}	--
S95	4	S78	14	{1,2,3,4}	{a51}	--
S96	4	S80	9	{1,2,3,4}	{a51}	--
S97	4	S35	12	{1,2,3,4}	{a51}	--
S98	4	S37	12	{1,2,3,4}	{a51}	--
S99	4	S39	9	{1,2,3,4}	{a51}	--
S100	4	S58	17	{1,2,3,4}	{a51}	--
S101	4	S60	13	{1,2,3,4}	{a51}	--
S102	4	S64	13	{1,2,3,4}	{a51}	--
S103	4	S66	11	{1,2,3,4}	{a51}	--

(1)	(2)	(3)	(4)	(5)	(6)	(7)
State X	Level	Parent State	Decision Point (dpx)	Completed Actvs (Fx)	In Progress Actvs(Ax)	Resource feasible RSS
S104	4	S68	11	{1,2,3,4}	{a51}	--
S105	4	S79	17	{1,2,3,4}	{a51}	--
S106	4	S81	12	{1,2,3,4}	{a51}	--
S107	4	S45	11	{1,2,3,4}	{a51}	--
S108	4	S47	11	{1,2,3,4}	{a51}	--
S109	4	S49	8	{1,2,3,4}	{a51}	--
S110	4	S51	8	{1,2,3,4}	{a51}	--
S111	4	S53	11	{1,2,3,4}	{a51}	--
S112	4	S55	11	{1,2,3,4}	{a51}	--
S113	4	S82	11	{1,2,3,4}	{a51}	--
S114	4	S46	14	{1,2,3,4}	{a51}	--
S115	4	S48	14	{1,2,3,4}	{a51}	--
S116	4	S50	14	{1,2,3,4}	{a51}	--
S117	4	S52	14	{1,2,3,4}	{a51}	--
S118	4	S54	11	{1,2,3,4}	{a51}	--
S119	4	S56	11	{1,2,3,4}	{a51}	--
S120	4	S83	14	{1,2,3,4}	{a51}	--
S121	4	S42	9	{1,2,3,4}	{a51}	--
S122	4	S43	12	{1,2,3,4}	{a51}	--
S123	4	S61	11	{1,2,3,4}	{a51}	--
S124	4	S69	11	{1,2,3,4}	{a51}	--
S125	4	S71	11	{1,2,3,4}	{a51}	--
S126	4	S44	15	{1,2,3,4}	{a51}	--
S127	4	S62	10	{1,2,3,4}	{a51}	--
S128	4	S70	14	{1,2,3,4}	{a51}	--
S129	4	S72	14	{1,2,3,4}	{a51}	--
S130	4	S73	14	{1,2,3,4}	{a51}	--
S131	4	S75	14	{1,2,3,4}	{a51}	--
S132	4	S74	17	{1,2,3,4}	{a51}	--
S133	4	S76	17	{1,2,3,4}	{a51}	--

Optimal Makespan: 6 (from state S88).

The starting state one (state S1) at level zero shows the (dummy) start activity a_{11} in progress and the next decision point as t = 0 (as the dummy start activity starts and ends at t = 0). The candidate activities on completion of the start activity are computed which are activities two and three. The resource feasible RSS arising from these candidate activities are computed next, which are: $\{a_{21}\}$, $\{a_{22}\}$, $\{a_{22},a_{32}\}$, $\{a_{31}\}$, and $\{a_{32}\}$. Each of these RSS is developed into a corresponding state (child states of state S1), and these

child states are numbered in continuity as states S2, S3, S4, S5, and S6. As a total of one activity has so far completed, these child states are appended to the search tree at level one. Level zero's processing is now complete and we proceed to level one.

The first state encountered at level one is state S2 in which one activity, a_{21}, is completing. Its finish time, and subsequent set of candidate activities are first computed. Next, all the feasible RSS are computed, which are $\{a_{31}\}$, $\{a_{31},a_{41}\}$, $\{a_{31},a_{42}\}$, $\{a_{32}\}$, $\{a_{32},a_{42}\}$, $\{a_{41}\}$, and $\{a_{42}\}$. The RSS $\{a_{32},a_{41}\}$ needing twelve units of NR2 is not feasible as the amount consumed in activity $\{a_{21}\}$ is four units leaving a residual of only eleven units. The remaining RSS are then developed into child states of S2 and are appended at the suitable level of the tree (here, level two, as two activities are now finished, namely activities one and two). Each remaining state at level one (here, S3 to S6) is processed one by one before moving to the next level.

Once the states at level one are completely processed, we advance to the next level, i.e. level two, and start processing of states at that level. The newly generated child states appended to the tree at any level are stored in a memory conserving data structure. Hence it is not necessary that all states at subsequent levels are processed in the original order of their generation (this is evident from the column three, depicting the parent state, for level three in Table 1).

All levels are processed, in breadth-first order, till the states at the last level are all generated (here level four, as it is enough to just schedule the dummy finish activity indicating the completion of the project). Note that even in this very small example instance, Example 4, 133 states are generated. Fifty of these, representing complete schedules, are at the last level, one (or more) of which is (are) optimal. To find the optimal makespan, we traverse the states at the last level and pick up the optimal solution(s). The number of states generated reaches millions for instances in even the smallest problem set of twelve activities (set j10) in PSPLIB. However, without pruning rules, this approach generates far too many states, takes much computational time, and needs more memory for even slightly larger problem instances. Hence, the algorithm is augmented with suitable pruning rules which are described below. For convenience, we continue with the same problem instance, Example 4, and describe the implementation of all pruning rules, first one by one, and then all together, to demonstrate their power.

2.2 Pruning Rules

Three pruning rules are used to augment the above algorithm to reduce the computational effort needed. These pruning rules do not compromise the optimality of the solution. The rules are: the Left Shift Rule (LS), the One-Child Set Rule (1C), and the Dominance Pruning Rule (DP). The application of all the rules is illustrated through the problem instance in Example 4. Theoretical proof of optimality of the solution on using these pruning rules are provided later.

2.2.1 Left-Shift Rule (LS)

An activity, in its given mode, which can be started earlier than its assigned start time in current state, without violating any resource or precedence constraints and the start times of already scheduled activities, is a *left shiftable activity*. The LS rule is that if an RSS, say A, at a decision point dp_X, is left shiftable then do not generate any child state of X corresponding to this RSS. Left-shift rule is employed by several scheduling methods. In breadth-first algorithm for regular measures, it is necessary to check only the activities *in progress* for left shiftability, as left shifting activities which are completed before the current decision point, dp_X, will not alter the end time (makespan) of the solution generated from current sub tree in consideration. This version of the left shift rule is the *local left shift rule*. The global left shift rule would check the left shiftability for all time periods (along with completion of predecessor activities) since the start time of the project (i.e. t=0) and involves substantial computational effort. The application of this rule to the same example, Example 4, solved earlier without any pruning rules, is described in Table 2.

Table 2: Breadth-first Search in Example 4 Project with only Left Shift Rule

(1)	(2)	(3)	(4)	(5)	(6)	(7)
State X	Level	Parent State	Decision Point (dpx)	Completed Actvs (Fx)	In Progress Actvs (Ax)	Resource feasible RSS
S1	0	--	0	--	{a11}	{a21}{a22}{a22,a32} {a31}{a32}
S2	1	S1	2	{1}	{a21}	{a31}{a31,a41} {a31,a42}{a32} {a32,a42}{a41}{a42}
S3	1	S1	4	{1}	{a22}	{a31}{a31,a41} {a31,a42}{a32} {a32,a41}{a32,a42} {a41}{a42}
S4	1	S1	4	{1}	{a22,a32}	{a31}{a31,a41}{a31,a42}{a32} {a32,a41}{a32,a42} {a41}{a42}
S5	1	S1	3	{1}	{a31}	{a21}{a22}
S6	1	S1	6	{1}	{a32}	{a21}{a22}
S7	2	S2	5	{1,2}	{a31}	{a41}{a42}
S8	2	S2	5	{1,2}	{a31,a41}	{a41}{a42}
S9	2	S2	5	{1,2}	{a31,a42}	{a41}{a42}
S10	2	S2	8	{1,2}	{a32}	{a42}
S11	2	S2	8	{1,2}	{a32,a42}	{a42}
S12	2	S2	6	{1,2}	{a41}	{a31}
S13	2	S2	9	{1,2}	{a42}	{a31}{a32}
S14	2	S3	7	{1,2}	{a31}	{a41}{a42}
S15	2	S3	7	{1,2}	{a31,a41}	{a41}{a42}
S16	2	S3	7	{1,2}	{a31,a42}	{a41}{a42}
S17	2	S3	10	{1,2}	{a32}	(LS)
S18	2	S3	8	{1,2}	{a32,a41}	(LS)
S19	2	S3	10	{1,2}	{a32,a42}	(LS)
S20	2	S3	8	{1,2}	{a41}	{a31}{a32}
S21	2	S3	11	{1,2}	{a42}	{a31}{a32}
S22	2	S4	7	{1,2}	{a31}	{a41}{a42}
S23	2	S4	7	{1,2}	{a31,a41}	{a41}{a42}
S24	2	S4	7	{1,2}	{a31,a42}	{a41}{a42}
S25	2	S4	6	{1,2}	{a32}	{a41}{a42}
S26	2	S4	6	{1,2}	{a32,a41}	{a41}{a42}
S27	2	S4	6	{1,2}	{a32,a42}	{a41}{a42}
S28	2	S4	8	{1,2}	{a41}	{a31}{a32}
S29	2	S4	11	{1,2}	{a42}	{a31}{a32}

(1)	(2)	(3)	(4)	(5)	(6)	(7)
State X	Level	Parent State	Decision Point (dpx)	Completed Actvs (Fx)	In Progress Actvs (Ax)	Resource feasible RSS
S30	2	S5	5	{1,3}	{a21}	{a41}{a42}
S31	2	S5	7	{1,3}	{a22}	{a41}{a42}
S32	2	S6	8	{1,3}	{a21}	{a42}
S33	2	S6	10	{1,3}	{a22}	(LS)
...and so on. A completely solved example, with all pruning rules applied, follows later.						

The generation of states for level one by processing states at level zero, is identical to the previous explanation. However, while processing states at level one to generate states at level two, we find that the schedules in states S17, S18, S19, and S33 contain activity (or activities) which are left shiftable. Hence, these states are not processed (in practice, these states are not appended to the search tree at all). The remark '(LS)' in the remarks column indicates the application of the left shift rule to a state. The schedule obtained by developing State 4 will be at least as good as one obtained by developing State 17. The comparison of these partial schedules is shown in Figure 8.

Figure 8: Application of Left-shift Rule

In a small example problem instance, the frequency with which such schedules are found, demonstrates the utility of this rule's application in reducing the size of the search tree and saving computational effort. The total number of states generated thus reduces to ninety seven from earlier one hundred and thirty three for this small example problem instance. The final solution is not compromised as any partial solution with a left shiftable activity will not generate a solution superior to another partial solution which contains that activity (in the same mode) already at left and hence finishing earlier. We provide the proof of optimality of the solution when the left shift rule is applied later.

We now explain the application of the one child set rule through the same example, Example 4. First, we describe the one child set rule.

2.2.2 One-Child Set Rule (1C)

If all activities in the parent state have completed, and among the candidate activities, each activity, in all its modes, can be scheduled with all other activities together, in all of their mode combinations without causing a resource violation, then (instead of all resource satisfying sets) we need to consider only the maximal resource satisfying sets (MRS) for generating the child states of this parent state. The concept of MRS is adopted from Nazareth and Bhattacharya (1993) and Nazareth (1995) with some modification. In each MRS set each of the candidate activities is included, in one of its modes. For the pruning rule to apply, all possible sets of mode combinations of all candidate activities should be (renewable and non-renewable) resource feasible. Note

that one (or more) of these candidate activities is the longest in its selected mode. We call this activity in its selected mode the *distinguished member* of the MRS.

Although application of this rule involves substantial computational overhead, the reduction in computational requirement on its application justifies its use, as it is able to prune several partial schedules at early levels which cuts large chunks of the search tree from the need to be generated. As more activities are completed and lower levels in the search tree are reached, non-renewable resources are depleted. In a large number of these partial schedules, it is then not possible to schedule all candidate activities in all of their modes due to reduced residuals of non-renewable resources. Hence, the rule applies less frequently. The power of the rule, too, is diminished, as the size of the search tree pruned at lower levels is relatively smaller. As such, deactivating the rule in the last few levels may yield an improved computational performance, though, we apply the rule at all levels. We solve the same example, Example 4, by applying only the one child set rule. The generation of states with only one child set rule is given in Table 3.

Table 3: Breadth-first Search in Example 4 Project with only 1C Rule

(1)	(2)	(3)	(4)	(5)	(6)	(7)	
State X	Level	Parent State	Decision Point (dpx)	Completed Actvs (Fx)	In Progress Actvs (Ax)	Resource feasible RSS	
S1	0	--	0	--	{a11}	{a21}{a22}{a22, a32}{a31}{a32}	
S2	1	S1	2	{1}	{a21}	{a31}{a31,a41} {a31,a42}{a32} {a32,a42}{a41} {a42}	
S3	1	S1	4	{1}	{a22}	{a31}{a31,a41} {a31,a42}{a32} {a32,a41}{a32,a42} {a41}{a42}	*
S4	1	S1	4	{1}	{a22,a32}	{a31}{a31,a41} {a31,a42}{a32} {a32,a41} {a32,a42}{a41}{a42}	**
S5	1	S1	3	{1}	{a31}	{a21}{a22}	
S6	1	S1	6	{1}	{a32}	{a21}{a22}	
*1C rule applied, hence, process only {a31,a41}{a31,a42}{a32,a41} and {a32,a42}							
**1C rule applied, hence, process only {a31,a41}{a31,a42}{a32,a41} and {a32,a42}							
...and so on. A completely solved example, with all pruning rules applied, follows later.							

The child states at level one for the root starting state are generated similar to previous discussion. However, while processing the level one states and applying the one child

set rule, for states S3 and S4, we generate the child states only for those RSSs which are maximal, i.e. the MRSs. These MRSs are $\{a_{31},a_{41}\}$, $\{a_{31},a_{42}\}$, $\{a_{32},a_{41}\}$, and $\{a_{32},a_{42}\}$. The remark '1C' in the remarks column indicates the application of one child set rule. This further reduces the number of states to be generated. Only ninety three states for the Example 4 problem instance are generated when one child set rule is applied. We provide the proof of optimality of the solution when the one child set rule is applied later. Note that if the one child set rule is not applied, and instead, all RSSs are developed into child states (partial solutions), they lead to such child states which are subsequently pruned by the left-shift rule or by the dominance pruning rule.
Next, we discuss the dominance pruning rule, which, out of the three rules applied, appears to be the most powerful when all the rules are applied only one at a time to the test problem sets. It prunes several partial schedules much earlier, which would have been pruned later by the left shift rule or the one child set rule.

2.2.3 Dominance Pruning Rule (DP)

If at any time during the execution of breadth-first algorithm there are two states X and Y in the search tree such that:

$F_X = F_Y$, i.e., the activities completed are same;
$A_X = A_Y$, i.e. activities in progress and their corresponding modes are same;
the residual of each non-renewable resource, after consumption by all activities completed or allocation to all activities in progress in set X, is same or more than in set Y;
the starting time in state X of each activity in A_X is less than or equal to its starting time in state Y;

then prune state Y from the search tree, as state X dominates state Y.

A state X can dominate a state Y only if they are both at the same level, making the rule easier to implement in a breadth-first progress scheme. By collating sets of activities co mpleted and non-renewable resources consumed in one data structure; the activities in p rogress and their modes in another data structure; and the start times in a yet another dat a structure; all connected suitably through linked lists, the memory requirement in the al gorithm's implementation is substantially reduced. The scheme also enables simpler and speedier implementation of the pruning rule. The application of only the dominance pru ning rule in the Example 4 problem instance is described below. The states developed ar e as shown in Table 4.

Table 4 : Breadth-first Search in Example 4 Project with only DP Rule

(1)	(2)	(3)	(4)	(5)	(6)	(7)
State X	Level	Parent State	Decision Point (dpx)	Completed Actvs (Fx)	In Progress Actvs (Ax)	Resource feasible RSS
S1	0	--	0	--	{a11}	{a21}{a22}{a22,a32}{a31}{a32}
S2	1	S1	2	{1}	{a21}	{a31}{a31,a41{a31,a42}{a32}{a32,a42}{a41}{a42}
S3	1	S1	4	{1}	{a22}	{a31}{a31,a41{a31,a42}{a32}{a32,a41{a32,a42}{a41}{a42}
S4	1	S1	4	{1}	{a22,a32}	{a31}{a31,a41{a31,a42}{a32}{a32,a41{a32,a42}{a41}{a42}
S5	1	S1	3	{1}	{a31}	{a21}{a22}
S6	1	S1	6	{1}	{a32}	{a21}{a22}
S7	2	S2	5	{1,2}	{a31}	
S8	2	S2	5	{1,2}	{a31,a41}	
S9	2	S2	5	{1,2}	{a31,a42}	
S10	2	S2	8	{1,2}	{a32}	
S11	2	S2	8	{1,2}	{a32,a42}	
S12	2	S2	6	{1,2}	{a41}	
S13	2	S2	9	{1,2}	{a42}	
S14	2	S3	7	{1,2}	{a31}	
S15	2	S3	7	{1,2}	{a31,a41}	
S16	2	S3	7	{1,2}	{a31,a42}	
S17	2	S3	10	{1,2}	{a32}	
S18	2	S3	8	{1,2}	{a32,a41}	
S19	2	S3	10	{1,2}	{a32,a42}	
S20	2	S3	8	{1,2}	{a41}	
S21	2	S3	11	{1,2}	{a42}	
S22	2	S4	7	{1,2}	{a31}	(S22 DP by S14)
S23	2	S4	7	{1,2}	{a31,a41}	(S23 DP by S15)
S24	2	S4	7	{1,2}	{a31,a42}	(S24 DP by S16)
S25	2	S4	6	{1,2}	{a32}	(S17 DP by S25)
S26	2	S4	6	{1,2}	{a32,a41}	(S18 DP by S26)
S27	2	S4	6	{1,2}	{a32,a42}	(S19 DP by S27)
S28	2	S4	8	{1,2}	{a41}	(S28 DP by S20)
S29	2	S4	11	{1,2}	{a42}	(S29 DP by S21)
S30	2	S5	5	{1,3}	{a21}	
S31	2	S5	7	{1,3}	{a22}	
S32	2	S6	8	{1,3}	{a21}	
S33	2	S6	10	{1,3}	{a22}	

...and so on. A completely solved example, with all pruning rules applied, follows later.

Table 5: Breadth-first Search in Example 4 Project with all Rules

(1)	(2)	(3)	(4)	(5)	(6)	(7)	
State X	Level	Parent State	Decision Point (dpx)	Completed Actvs (Fx)	In Progress Actvs (Ax)	Resource feasible RSS	
S1	0	--	0	--	{a11}	{a21}{a22}{a22,a32} {a31}{a32}	
S2	1	S1	2	{1}	{a21}	{a31}{a31,a41}{a31,a42 }{a32}{a32,a42}{a41}{a 42}	
S3	1	S1	4	{1}	{a22}{a3 2,a41} {a32,a42}	{a31}{a31,a41} {a31,a42}{a32} {a41}{a42}	*
S4	1	S1	4	{1}	{a22,a32}	{a31}{a31,a41}{a31,a42 }{a32}{a32,a41}{a32,a4 2} {a41}{a42}	**
S5	1	S1	3	{1}	{a31}	{a21}{a22}	
S6	1	S1	6	{1}	{a32}	{a21}{a22}	
S7	2	S2	5	{1,2}	{a31}	{a41}{a42}	
S8	2	S2	5	{1,2}	{a31,a41}	{a41}{a42}	
S9	2	S2	5	{1,2}	{a31,a42}	{a41}{a42}	
S10	2	S2	8	{1,2}	{a32}	{a42}	
S11	2	S2	8	{1,2}	{a32,a42}	{a42}	
S12	2	S2	6	{1,2}	{a41}	{a31}	
S13	2	S2	9	{1,2}	{a42}	{a31}{a32}	
S14	2	S3	7	{1,2}	{a31,a41}	{a41}{a42}	
S15	2	S3	7	{1,2}	{a31,a42}	{a41}{a42}	
S16	2	S3	8	{1,2}	{a32,a41}	(LS)	
S17	2	S3	10	{1,2}	{a32,a42}	(LS)	
S18	2	S4	7	{1,2}	{a31,a41}	(S18 DP by S14)	
S19	2	S4	7	{1,2}	{a31,a42}	(S19 DP by S15)	
S20	2	S4	6	{1,2}	{a32,a41}	{a41}{a42}	
S21	2	S4	6	{1,2}	{a32,a42}	{a41}{a42}	
S22	2	S5	5	{1,3}	{a21}	{a41}{a42}	
S23	2	S5	7	{1,3}	{a22}	{a41}{a42}	
S24	2	S6	8	{1,3}	{a21}	{a42}	
S25	2	S6	10	{1,3}	{a22}	(LS)	
S26	3	S7	9	{1,2,3}	{a41}	(LS)	
S27	3	S7	12	{1,2,3}	{a42}	(LS)	
S28	3	S8	6	{1,2,3}	{a41}	{a51}	
S29	3	S8	12	{1,2,3}	{a42}	(S29 DP by S31)	
S30	3	S9	9	{1,2,3}	{a41}	(S30 DP by S28)	

(1)	(2)	(3)	(4)	(5)	(6)	(7)	
State X	Level	Parent State	Decision Point (dpx)	Completed Actvs (Fx)	In Progress Actvs (Ax)	Resource feasible RSS	
S31	3	S9	9	{1,2,3}	{a42}	{a51}	
S32	3	S10	15	{1,2,3}	{a42}	(LS)	
S33	3	S11	9	{1,2,3}	{a42}	{a51}	
S34	3	S12	9	{1,2,4}	{a31}	(LS)	
S35	3	S13	12	{1,2,4}	{a31}	(LS)	
S36	3	S13	15	{1,2,4}	{a32}	(LS)	
S37	3	S14	8	{1,2,3}	{a41}	{a51}	
S38	3	S14	14	{1,2,3}	{a42}	{a51}	
S39	3	S15	11	{1,2,3}	{a41}	(S39 DP by S37)	
S40	3	S15	11	{1,2,3}	{a42}	(S38 DP by S40)	
S41	3	S20	8	{1,2,3}	{a41}	(S41 DP by S28)	
S42	3	S20	13	{1,2,3}	{a42}	(S42 DP by S31)	
S43	3	S21	10	{1,2,3}	{a41}	(S43 DP by S28)	
S44	3	S21	11	{1,2,3}	{a42}	(S44 DP by S31)	
S45	3	S24	15	{1,2,3}	{a42}	(S45 DP by S33)	
S46	3	S22	9	{1,2,3}	{a41}	(S46 DP by S28)	
S47	3	S22	12	{1,2,3}	{a42}	(S47 DP by S31)	
S48	3	S23	11	{1,2,3}	{a41}	(S48 DP by S37)	
S49	3	S23	14	{1,2,3}	{a42}	(S49 DP by S40)	
S50	4	S33	9	{1,2,3,4}	{a51}	--	
S51	4	S28	6	{1,2,3,4}	{a51}	(S50 DP by S51)	
S52	4	S31	9	{1,2,3,4}	{a51}	--	
S53	4	S37	8	{1,2,3,4}	{a51}	(S52 DP by S53)	
S54	4	S40	11	{1,2,3,4}	{a51}	--	
*1C hence process only {a31,a41}{a31,a42}{a32,a41}{a32,a42}							
**1C hence process only {a31,a41}{a31,a42}{a32,a41}{a32,a42}							

Example 4 Optimal makespan: 6

The algorithm is started, as in earlier explanation, with the starting state S1 at root. Its development at the first level is identical to previous examples. After developing the states of the first level by processing the starting state, we develop the next (second) level. The first state at level one, state S2, is taken and its seven RSSs are developed into states S7 to S13. The state S3 of level one is processed next, and its eight RSSs are developed into states S14 to S21. State S4 of level one is then processed, yielding eight RSSs. The first of its RSS, $\{a_{31}\}$, generates state S22, which is dominated by state S14, hence, pruned. Observe that: (a) activities completed in S22 and S14 are same, (b) activities currently in progress and their modes, in S22 and S14, are same, (c) residual of each non-renewable resource after allocation to activities in progress in state S14 is same or more than corresponding residual in state S22, and (d) the starting time(s) of activities in progres

s in S14 is same or earlier with respect to S22. Thus, state S14 dominates state S22. Hen ce, state S22 may be pruned. By identical comparisons between states S23 and S15, S23 is dominated, and hence, pruned. In this example, all child states of parent state S4 are d ominated and pruned. This shows the power of this rule in reducing computational time in solving a problem instance. We now explain the application of all the three pruning r ules together in a completely worked out example. Once again, consider the same probl em instance, Example 4, as shown in earlier. The development of states in breadth-first order, applying all pruning rules, is as shown in Table 5.

The processing of level zero to develop level one is identical to the previous explanation. However, while processing the level one, we encounter two states, S3 and S4, where the one child set rule applies. Three states at level two are pruned due to left-shift rule, and two due to dominance pruning rule. The effect of pruning rules becomes even more evident at the level three, where six states are pruned by the left-shift rule, and thirteen due to the dominance pruning rule, leaving only four states for further development. This demonstrates the power of the dominance pruning rule. The advantage arising from the DP rule is explained as follows. While generating child states in different branches, as activities are scheduled and completed in different modes, there is a difference between the non-renewable resource amounts consumed. However, when activities are scheduled in same modes, but in a different order, two similar child states X and Y, may be generated from two different parent states. DP avoids repeated processing of such duplicate states by pruning them timely. Additionally, DP prunes some such child states, which at later levels, would have been pruned by LS. By pruning them at an earlier level, substantial computational effort is saved. The pseudocode for pruning rules incorporated version of breadth-first algorithm is as follows.

Algorithm Breadth-first With Pruning Rules

Preprocessing	remove infeasible modes, redundant resources, and inferior modes
Step 1 (Initialization)	create the root state I at level 0 in the search tree
Step 2 (Loop)	for L from 0 to N-1 do
	for each state X at level L do
Step 3 (Expansion)	determine dp_X, completed activities, and child states' level (L_c)
	construct K_X and all the RSSs
Step 3A (1C)	if 1C applies, retain only MRSs with all candidate activities
	for each RSS/MRS do
Step 3B (LSR)	if LSR applies then
	do not generate child state
	else
	generate a child state at L_c for each RSS/MRS
Step 3C (DP)	apply the dominance pruning rule to all states at L_c
Step 4 (Termination)	Traverse the states at level N and output the complete schedule associated with the state X with minimum makespan at level N

At the last level generated, as we schedule the dummy end activity, the tree yields multiple solution states. Not all, but many among these are optimal solution states. These solutions differ by the total amounts of non-renewable resources consumed in the

schedules generated. Even for small problem instances, sometimes hundreds of multiple solutions may be found. It is then possible to search for such an optimal makespan solution, which simultaneously optimizes on the consumption of the first non-renewable resource (or alternatively the second non-renewable resource). As this only requires a re-traversal of the leaf nodes at the last level, the computational effort and time needed is negligible. It is possible to find multiple such solutions, which consume the same least amount of the first non-renewable resource, and then, within these, the solution consuming the least quantity of the second non-renewable resource may also be searched, thus leading to an *exact multi objective non-renewable resource conserving optimal makespan solution*. Note that among several single-objective optimal solutions, the modes in which activities are performed may be different, and so are the critical paths. As we search for higher order multi objective solutions, the number of critical paths increases, which could be of concern to a manager.

The multiple exact single objective optimal solutions, with different activity modes and start times, may be substantially different from any given optimal solution. A Pareto bound analysis between two points representing optimal solutions for two different objectives may fail to yield another multi objective optimal solution even though one exists!

It is noteworthy that the problem instances which have multiple solutions involve more processing in Breadth-first approach as it builds all the optimal solutions up to the last level, while Depth-first approach, once it finds an optimal solution, prunes away even other optimal solutions. When multiple optimal solutions exist, one may be found early in the search tree to the advantage of Depth-first. However, Breadth-first is still able to solve the larger problem instances faster!

Retraction Example: We provide below a small example problem instance which would not yield the correct solution without considering retraction. Consider the problem instance shown in Figure 9.

Figure 9: Example Problem Instance for Retraction

The generation of all states up to level 3 for this instance is shown in Table 6.

Table 6: Breadth-first Search in Example with Retraction

(1)	(2)	(3)	(4)	(5)	(6)	(7)	
State X	Level	Parent State	Decision Point (dpx)	Completed Actvs (Fx)	In Progress Actvs (Ax)	Resource feasible RSS	
S1	0	--	0	--	{a11}	{a21}{a21,a31}{a21,a31,a41} {a21,a41}{a31}{a31,a41}{a41}	*
S2	1	S1	3	{1}	{a21,a31,a41}	{a21}{a21,a41}{a41}{a41,a61} {a41,a62}{a61}{a62}	**

(1)	(2)	(3)	(4)	(5)	(6)	(7)	
State X	Level	Parent State	Decision Point (dpx)	Completed Actvs (Fx)	In Progress Actvs (Ax)	Resource feasible RSS	
S3	2	S2	4	{1,3}	{a21}	{a41}{a41,a61}{a41,a62}{a51} {a51,a61}{a51,a62}{a61}{a62}	
S4	2	S2	4	{1,3}	{a21,a41}	{a41}{a41,a61}{a41,a62}{a51} {a51,a61}{a51,a62}{a61}{a62}	
S5	2	S2	9	{1,3}	{a41}	{a21}{a61}{a62}	
S6	2	S2	8	{1,3}	{a41,a61}	{a21}{a21,a41}{a41}	
S7	2	S2	9	{1,3}	{a41,a62}	{a21}{a61}{a62}	
S8	2	S2	8	{1,3}	{a61}	{a21}{a21,a41}{a41}	
S9	2	S2	13	{1,3}	{a62}	{a21}{a21,a41}{a41}	
S10	3	S3	13	{1,2,3}	{a41}	(S10 DP by S18)	
S11	3	S3	9	{1,2,3}	{a41,a61}	(S11 DP by S19)	
S12	3	S3	13	{1,2,3}	{a41,a62}	(S12 DP by S20)	
S13	3	S3	6	{1,2,3}	{a51}	(S21 DP by S13)	
S14	3	S3	6	{1,2,3}	{a51,a61}	(S22 DP by S14)	
S15	3	S3	6	{1,2,3}	{a51,a62}	(S23 DP by S15)	
S16	3	S3	9	{1,2,3}	{a61}	(S24 DP by S16)	
S17	3	S3	14	{1,2,3}	{a62}	(S25 DP by S17)	
S18	3	S4	9	{1,2,3}	{a41}	{a51}{a51,a61}{a51,a62} {a61}{a62}	#
S19	3	S4	9	{1,2,3}	{a41,a61}	{a51,a81}	
S20	3	S4	9	{1,2,3}	{a41,a62}	{a51}{a51,a61}{a51,a62} {a61}{a62}	##
S21	3	S4	6	{1,2,3}	{a51}	(S21 DP by S13)	
S22	3	S4	6	{1,2,3}	{a51,a61}	(S21 DP by S13)	
S23	3	S4	6	{1,2,3}	{a51,a62}	(S21 DP by S13)	
S24	3	S4	9	{1,2,3}	{a61}	(S21 DP by S13)	
S25	3	S4	14	{1,2,3}	{a62}	(S21 DP by S13)	
S26	3	S5	13	{1,3,4}	{a21}	{a51}{a51,a61}{a51,a62} {a61}{a62}	##
S27	3	S5	14	{1,3,4}	{a61}	{a21}	
S28	3	S5	19	{1,3,4}	{a62}	(S28 DP by S34)	
S29	3	S6	12	{1,3,6}	{a21}	{a41}{a51}	
S30	3	S6	9	{1,3,6}	{a21,a41}	{a21}	
S31	3	S6	9	{1,3,6}	{a41}	{a21}	
S32	3	S7	13	{1,3,4}	{a21}	(S32 DP by S26)	
S33	3	S7	14	{1,3,4}	{a61}	(S33 DP by S27)	

(1)	(2)	(3)	(4)	(5)	(6)	(7)	
State X	Level	Parent State	Decision Point (dpx)	Completed Actvs (Fx)	In Progress Actvs (Ax)	Resource feasible RSS	
S34	3	S7	13	{1,3,4}	{a62}	{a21}	
S35	3	S8	12	{1,3,6}	{a21}	(S35 DP by S29)	
S36	3	S8	12	{1,3,6}	{a21,a41}	(S36 DP by S30)	
S37	3	S8	17	{1,3,6}	{a41}	(S37 DP by S31)	
S38	3	S9	17	{1,3,6}	{a21}	(S38 DP by S29)	
S39	3	S9	17	{1,3,6}	{a21,a41}	(S39 DP by S30)	
S40	3	S9	22	{1,3,6}	{a41}	(S40 DP by S31)	
S41	4	S18	11	{1,2,3,4}	{a51,a61}	{a61}{a62}	
S42	4	S18	11	{1,2,3,4}	{a51,a62}		

*(1C), only {a21,a31,a41} processed.
**(Note retraction of a41 in parent state S2
#(1C), only {a51,a61} and {a51,a62} processed.
##(1C), only {a51,a61} and {a51,a62} processed.
##(1C), only {a51,a61} and {a51,a62} processed.
...and so on. Note that there is only one optimal solution to this problem achieved when considering retraction.
Optimal makespan: 18

Note that a retracted activity is considered as if it had never started. In the multi-mode case of RCPSP, a retracted (or withdrawn) activity (or activities), when scheduled later, may be performed even in different mode(s) compared to their previous mode from which they are now retracted. This allows them to be performed along with such other activities, which were otherwise successors of those activities with which they were earlier being scheduled. This leads to generation of such feasible schedules for a problem instance which would not be generated, if retraction is not considered.

Single Processor Best-first Algorithm: The best-first algorithm differs from the breadth-first in two respects: (a) the order of selection of next state to be expanded, and (b) the termination condition. The optimistic estimates of earliest start time (EST), latest start time (LST), earliest finish time (EFT), and latest finish time (LFT) are made by the Metra Potential Method (MPM), without considering the resource constraints. We use a makespan heuristic for evaluating the forward estimate of the makespan of a partial schedule as follows: a partial schedule is converted into a pseudo-complete schedule by adding the unfinished activities in conformance with the precedence constraints but ignoring the resource constraints, and this estimated makespan is used as the heuristic estimate of the state. A priority queue is maintained to keep the states with smallest makespan estimates first; breaking any ties by considering the finished activities set, F_X, and breaking further ties by the decision point, dp_X, of the state. As the heuristic value underestimates the actual makespan, the first solution state selected yields a schedule of minimum length. Note that though with same makespan, the schedules found by breadth-first and best-first may be different. The algorithm is explained through an example problem instance after its pseudocode below.

Algorithm Best-first With Pruning Rules

Preprocessing	remove infeasible modes, redundant resources, and inferior modes
Step 1 (Initialization)	create the root state I at level 0 in the search tree, add to heap
Step 2 (Loop)	while (states in heap and solution not obtained) do get best state from heap determine earliest finish time from activities in progress determine activities finished, generate candidate set develop RSS, build child states applying pruning rules append to heap
Step 3 (Termination)	determine and output solution schedule

Now we present the application of Best-first algorithm for regular measures with an example, first to the same example problem as is presented for the Breadth-first algorithm, i.e. Example 4, and then for another problem in which application of all pruning rules in Best-first algorithm is visible. Consider the same problem as shown in example for Breadth-first algorithm. The states generated by Best-first monotone heuristic algorithm are given in Table 7 and explained thereafter.

Table 7: Best-first Search in Example 4 Project with Pruning Rules

(1)	(2)	(3)	(4)	(5)	(6)	(7)		
State X	Parent State	Completed Actvs (Fx)	In Progress Actvs (Ax)	Finish Time	Lower Bound	Number Finished	Resource feasible RSS	
--		--	{1a}	0	--	--	{a21}{a22}{a22,a32}{a31}{a32}	
S2	S1	{1}	{a21}	2	6	1	{a31}{a31,a41}{a31,a42}{a32}{a32,a42}{a41}{a42}	
S3	S1	{1}	{a22}	4	6	1		
S4	S1	{1}	{a22,a32}	4	6	1		
S5	S1	{1}	{a31}	3	6	1		
S6	S1	{1}	{a32}	6	6	1		
*								
S7	S2	{1,2}	{a31}	5	6	2	{a41}-LS{a42}-LS	#
S8	S2	{1,2}	{a31,a41}	5	6	2		
S9	S2	{1,2}	{a31,a42}	5	6	2		
S10	S2	{1,2}	{a32}	8	6	2		
S11	S2	{1,2}	{a32,a42}	8	6	2		
S12	S2	{1,2}	{a41}	6	6	2		
S13	S2	{1,2}	{a42}	9	6	2		
S16	S8	{1,2,3}	{a41}	6	6	3		
S17	S8	{1,2,3}	{a42}	12	6	3	{a51}	
S18	S16	{1,2,3,4}	{a51}	6	6	4		

Optimal makespan: 6

* State with smallest lower bound, followed by largest number of activities finished, followed by earliest finish time is selected from heap for processing first (here State S2). Resource feasible RSS of S2 are given in Col-8 of S2. Its child states, shown below, are added to the heap.

Both states are pruned due to the left shift rule.

The Best-first algorithm with all pruning rules solved the above example problem in just eighteen states. The starting state, S1, reveals five child states, S2 to S6. The state S2 is selected for generation of child states by the Best-first selection criteria, which is, (a) lowest lower bound estimate, and breaking ties with (b) largest number of activities completed, and breaking further ties with (c) earliest finish time of any activity from the activities in progress.

Child states of state S2, i.e. states S7 to S13, are generated and appended to the heap by the chosen criteria as stated above. State S7, which fits in the heap at the top, is now

selected for expansion, and the heap updated. However, both the RSS generated are seen to be pruned by the left shift rule. Hence, the next state from the heap is selected for expansion, which is state S8. The child states revealed are states S16 and S17, of which, the state S16 fits at the top of the heap. State S16 is selected for expansion, now, from the top of the heap, and it reveals a complete solution which is the obtained optimal solution. The above example shows the power of Best-first algorithm in solving these problems faster than Breadth-first. To demonstrate the application of all pruning rules in the Best-first algorithm, we consider an example problem instance.

Consider the example problem, Example 5, shown in Figure 10 below. Table 8 shows the development of the solution to this example problem instance using Best-first algorithm, in which, each pruning rule is applicable at least once. An explanation of the progress of the algorithm to this problem follows thereafter.

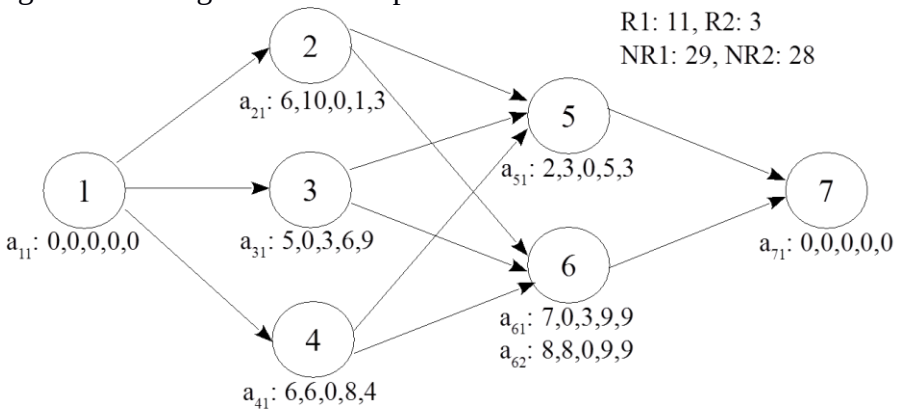

Figure 10: Example 5 Project for Best-first, Regular Measure Demonstrating All Pruning Rules

Table 8: Best-first Search in Second Example 5 Project with Pruning Rules

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	
State X	Parent State	Completed Actvs (Fx)	In Progress Actvs (Ax)	Finish Time	Lower Bound	Number Finished	Resource feasible RSS	
S1	--	--	{1a}	0	--	--	{a21}{a21,a31} {a31}{a31,a41} {a41}	
S2	S1	{1}	{a21}	6	13	1	{a31}{a31,a41} {a41}	*
S3	S1	{1}	{a21,a31}	5	13	1	{a21}{a41}	
S4	S1	{1}	{a31}	5	13	1	{a21}{a41}	
S5	S1	{1}	{a31,a41}	5	13	1	{a21}{a41}	
S6	S1	{1}	{a41}	6	13	1	{a21}{a21,a31} {a31}	**

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	
State X	Parent State	Completed Actvs (Fx)	In Progress Actvs (Ax)	Finish Time	Lower Bound	Number Finished	Resource feasible RSS	
S7	S3	{1,3}	{a21}	6	18	2	(pa-S3 {a41} LS) (pa-S5 {a21} LS)	
S10	S5	{1,3}	{a41}	6	18	2	{a21}	
S15	S7	{1,2,3}	{a41}	12	13	3	{a51}{a51,a61} {a51,a62} {a61}{a62}	#
S16	S15	{1,2,3,4}	{a51,a61}	14	19	4	{a61}{a62}	
S17	S15	{1,2,3,4}	{a51,a62}	14	19	4	{a61}{a62}	
S18	S10	{1,3,4}	{a21}	12	13	3	{a51}{a51,a61} {a51,a62} {a61}{a62}	$
S21	S17	{1,2,3,4,5}	{a61}	21	20	5		
S22	S17	{1,2,3,4,5}	{a62}	20	20	5		
S23	S16	{1,2,3,4,5}	{a61}	19	19	5	{a71}	
(S21 pa-S17 DL by S23)(S24 pa-S16 {a62} DM By S22)								
S25	S23	{1,2,3,4,5,6}	{a71}	19	19	6		
Optimal makespan: 19								
*(1C, process only {a31,a41})								
**(1C, process only {a21,a31})								
# (1C, process only {a51,a61}{a51,a62})								
$ (1C, process only {a51,a61}{a51,a62})								

The Best-first algorithm is started with the first state, S1, prepared. The RSS generated are $\{a_{21}\}$, $\{a_{21},a_{31}\}$, $\{a_{31}\}$, $\{a_{31},a_{41}\}$, and $\{a_{41}\}$, which are respectively developed into the states S2, S3, S4, S5, and S6. While each of these states is generated, there forward estimates (lower bounds or LB) using the shortest possible modes of all activities, and ignoring the resource constraints are also computed and these are appended to a heap according to their LB, breaking ties as already explained earlier.

The state S3, which appears at the top of the heap is developed first. The feasible RSS are $\{a_{21}\}$ and $\{a_{41}\}$. $\{a_{21}\}$ leads to the generation of state S7 which is appended to the heap at its right place, while S8 by $\{a_{41}\}$ is pruned using the left shift rule.

The next state is taken from top of the heap, which is S5. The RSS generated are $\{a_{21}\}$ and $\{a_{41}\}$. The state S9 developed with $\{a_{21}\}$ is left shiftable and hence, pruned. State S10 generated from the RSS $\{a_{41}\}$ is appended to the heap at its right place.

The state at top of heap is taken again, which is S4. Two RSS are generated, $\{a_{21}\}$ and $\{a_{41}\}$. Both the states that these lead to, S11 and S12, are left shiftable, and hence pruned. State S6 from top of the heap next taken yields the RSS $\{a_{21},a_{31}\}$, developed into state S13, which is also left shiftable. The state S14, from RSS $\{a_{21},a_{41}\}$ of the next state S2 taken from top of the heap is also left shiftable.

State S7 is now at the top of the heap and is taken for processing. The feasible RSS generated is $\{a_{41}\}$. State S15 is developed as the child state of state S7, and appended to the heap. Due to its estimated lower bound and number of activities completed, it arrives at the top of the heap, and hence, also becomes the immediately next state to be developed further. The RSS generated are $\{a_{51}\}$, $\{a_{51},a_{61}\}$, $\{a_{51},a_{62}\}$, $\{a_{61}\}$, and $\{a_{62}\}$. Applying the one child set rule, only the RSS $\{a_{51},a_{61}\}$ and $\{a_{51},a_{62}\}$ are developed into their child states, which are respectively, states S16 and S17. Both of these are appended to the heap at their respective places.

The next state at the top of the heap is taken for processing, which is the state S10. The RSS generated are again $\{a_{51}\}$, $\{a_{51},a_{61}\}$, $\{a_{51},a_{62}\}$, $\{a_{61}\}$, and $\{a_{62}\}$, and applying the one child set rule, only the RSS $\{a_{51},a_{61}\}$ and $\{a_{51},a_{62}\}$ are developed into their child states S19 and S20. However, both of these are pruned by states S17 and S18, respectively, applying the dominance pruning rule.

Now, the next state at the top of the heap is taken for development, which is state S17. The feasible RSS $\{a_{61}\}$ and $\{a_{62}\}$ are developed into its child states, which are respectively, S21 and S22, both of which are also appended to the heap at their appropriate places. The state now at the top of the heap is S16 which is taken for processing. The RSS generated are $\{a_{61}\}$ and $\{a_{62}\}$ giving rise to states S23 and S24. While appending the state S23 to the heap, state S21 is dominated by it and hence removed. On the other hand, while appending the state S24 to the heap, state S22 dominates it, and hence S24 is deleted too. State S23, which now at the top of the heap, is taken for processing. The RSS generated is $\{a_{71}\}$, which leads to the development of state S25, which reveals the optimal makespan for the project scheduling problem, i.e. nineteen (19).

We have demonstrated the application of all the pruning rules of the Best-first monotone heuristic in Example 5. It also shows how the Best-first is rapidly able to compute the optimal solution.

Multiple Solutions and Exact Multi Objective Solutions: The breadth-first algorithm generates many complete schedules at the last level. A large number of the complete schedules are sub-optimal, however, several optimal solutions are generated by breadth-first. Trivial multiple solutions for regular measures can be generated from a single schedule, for example, by shifting the activities on slack paths within available slack so as not to violate resource constraints.

Our breadth-first algorithm generates multiple (non-trivial) optimal solutions. The activities in these multiple optimal solutions are not only performed in varying sets of modes, the quantities of non-renewable resources consumed are different too. The critical path, and the number of critical paths in various solutions may also be different. We present a small example to demonstrate this ability of the Breadth-first algorithm. Consider the problem instance Example 6 shown in Figure 11.

Figure 11: Multiple Solutions and Multi-Objective Solution through Example 6

Note that activity-mode a_{51} is infeasible hence it is removed from consideration during pre-processing, and a_{52} is renamed as a_{51}. Breadth-first yields three optimal makespan solutions for this example, all of which are shown in Figure 12, each of which has the same optimal makespan, yet a different consumption of the non-renewable resources NR1 and NR2, which is also indicated.

Figure 12: Multiple Schedules and Multi Objective Solution for Example 6

The last shown solution with an optimal span (optspan) seventeen time units and consumption of fifteen units and twenty units of non-renewable resources one and two respectively, is the ***exact multi objective optimal solution***. As our generation scheme appends the new child states in order of their decreasing consumptions of non-renewable resources, and upon finding two states with same consumption of the first non-renewable resource, ordering them with respect to the next non-renewable resource and so on; we are able to search for an optimal makespan solution which also consumes the least amount of the first non-renewable resource. Among multiple such solutions, if they exist, we are further able to search for a solution which simultaneously optimizes the consumption of the second non-renewable resource and so on. Thus, breadth-first algorithm is capable of delivering an exact multi objective solution, where the minimization of non-renewable resource(s) consumed is the second (and third, ...) objective. As long as the number of solutions obtained does not reach one, further optimization among the available solutions is possible. With the reduction in consumed resources, naturally, multiple critical paths emanate. A non-renewable resource, whose consumption is to be minimized, may be identified by any one of multiple possible options, such as: (a) unit price of non-renewable resources, (b) a given priority order of conserving non-renewable resources, (c) a weighted priority measure of all non-renewable resources, and so on (note that all of these may lead to a different multi objective optimal solution). If, with the second objective too, multiple solution points with respect to activity modes and their start times (i.e. multiple dual objective optimal solutions in the feasible hyper-space) are obtained in the leaves of the B&B tree, it is further possible to optimize on next chosen objective (such as, the total cost of another non-renewable resource consumed based on its unit price).

Availability of multiple optimal solutions to choose from is in correspondence with project scheduling needs of managers, as often reduction in resource consumption is an

important simultaneous managerial goal. Though the computational requirement would rise, it is possible in our algorithm to alter the pruning rules and generate all feasible optimal solutions of a problem instance, among which, as desired, multi objective optimal solutions can be located. The ability of the algorithm to scale up in multi-processor implementation makes this approach very promising for future research.

3 THEORETICAL RESULTS

In this section we provide the proof of optimality of the breadth-first algorithm, first when applied without any pruning rules, and then with each of the pruning rules added one by one. For convenience, we refer to the breadth-first algorithm without any pruning rules as BD; One-Child Set Rule as 1C; Left Shift Rule as LS; and Dominance Pruning Rule as DP. Thus, BD+DP refers to the breadth-first algorithm with only the Dominance Pruning Rule and no other rule; and similarly, BD+DP+1C represents the breadth-first algorithm with the DP and 1C rules, but without the LS rule; and so on. Our goal is to prove that BD+DP+1C+LS yields an optimal solution.

We use the following terminology for the proofs: An *optimal schedule* is a complete schedule of minimal makespan. A complete schedule is *left-aligned* if it contains no left shiftable activities. A partial schedule is optimizable if it is an optimal schedule or an ancestor (as viewed in the search tree as a state) of an optimal schedule. A state is *optimal* (or *optimizable*) if its partial schedule is *optimal* (or *optimizable*). We first look at BD without any pruning rules.

Theorem 1: BD generates *all* left-aligned complete schedules. It may, in addition, generate some complete schedules that are not left-aligned.

Proof: Since: (a) the method of generation of states is exhaustive, and (b) no states are pruned, and (c) the activities are scheduled as early as possible consistent with the precedence and resource constraints, it follows that BD generates all left-aligned complete schedules. Due to retraction, some schedules that are not left-aligned are additionally generated. □

Corollary 1: BD generates an optimal schedule.

Proof: There is a left aligned schedule that is optimal. By Theorem 2.1 BD generates all left-aligned schedules, so it produces an optimal schedule. □
We now present proof that BD+DP produces an optimal schedule.

Lemma 1: If a state X is pruned by state Y during the execution of BD+DP and X is optimizable, then so is Y.

Proof: Let state Y dominate state X during the execution of BD+DP. Further, let state X be optimizable. Since state Y dominates state X, the activities finished in both states are same, i.e. $F_Y = F_X$; the activities in progress and their modes in both states are same, i.e. $A_Y = A_X$; activities in A_Y do not start later than corresponding activities in state X; and

residual of each non-renewable resource in state Y is same or greater than in state X. BD would generate both of the states X and Y. As X is optimizable, it has a successor state X' which is optimal, and is generated by BD.

Now, consider the decision points t_1, t_2,..., and the corresponding RSSs along the path in the search tree of BD, from the partial schedule in state X to the optimal schedule in state X'. Start from state Y instead of state X, and choose the same RSSs at the same time instants t_1, t_2,..., retaining each activity's start time and mode, and obtain schedule Y'. Schedule Y' has no precedence or resource conflicts and the same makespan as schedule X', hence it is an optimal makespan. There may be intervals of time in schedule Y' when no activities are in progress, i.e., it may contain left shiftable activities. Take each left shiftable activity in schedule Y', say in order of activity number, and retaining its mode, shift it left as far as possible without introducing any precedence or resource conflict. Let the resulting state be Y". Y" is optimizable and will be generated by BD+DP as a successor of Y, so Y is optimizable. □

Theorem 2: BD+DP produces an optimal schedule.

Proof: This follows immediately by Lemma 2.1, as whatever states get pruned, the search tree of BD+DP will always contain an optimizable state. □
We now add the One-Child Set Rule to BD+DP.

Theorem 3: BD+DP+1C produces an optimal schedule.

Proof: BD+DP yields an optimal schedule (by Theorem 2.2). Let us suppose that One-Child Set Rule is applicable when state X is expanded at decision point dp_X. Note that the number of MRSs eligible for retention here would be a product of all candidate activities' modes, each MRS including each candidate activity, while the total number of RSSs (including MRSs) would be much larger.

Suppose we generate child states corresponding only to all MRSs. Let these child states be m_1, m_2, m_3,.... Let the additional states which would have been generated using all RSSs, be s_1, s_2, s_3,.... Let state s_1 represent a state where exactly one activity, say a_i, is scheduled later than in a child state generated by the MRS m_1, and all activities scheduled in s_1 and m_1 are in correspondingly same modes. Thus a_i finishes later in s_1 as compared to m_1. Now, if s_1 generates an optimal schedule, and as a_i starts earlier in m_1, without any precedence and resource violation, m_1 must also generate an optimal schedule. Hence, we need not expand the child state corresponding to s_1. Postponing the processing of activities, which can be scheduled earlier, to a later time, with the same consumption in resources, can not lead to a shorter schedule. Note that activity a_i in state s_1 represents a left shiftable activity with respect to state m_1. The case for two or more activities straightaway follows. Thus, as for each RSS, we have a corresponding MRS generating a child state in which at least one activity, in the same mode, starts earlier than in child state generated by the said RSS, all the MRSs together shall produce any of the optimal solution(s), which would have been produced by any of the

RSSs. Hence, for an optimal solution in the above situation, we need to consider only the MRSs. □

In Theorem 2.4 below, we present the proof of optimality of a solution when the left shift rule is applied. This rule is widely used in tree-search procedures for regular measures, including the depth-first approach.

Theorem 4: BD+DP+1C+LS produces an optimal schedule.

Proof: BD+DP+1C yields an optimal schedule by Theorems 2.2 and 2.3. Thus, for a solution by BD+DP+1C+LS to be optimal, it suffices to show that it is as good as one by BD+DP+1C.

Note that we consider child states corresponding to all RSSs. The set of resource feasible MRSs is a subset of these RSSs. Now, consider a partial schedule whose child states are under development. Let this state be the parent state X. There is a set of activities which is completed at or before the decision point dp_X, the earliest finish time of an activity in progress in the parent state X. Let the candidate set of activities for child states of parent X be represented by C. C includes the activities in progress in X but not completed at dp_X.

As stated earlier, in generating the child states, all renewable and non-renewable resource feasible sets (the RSSs) are considered for the parent state X. Let one child state generated, state Y, be such that in it an activity, a_i in mode m_i, is left shiftable, i.e., it can be started at a time < dp_X, without violating any resource and precedence constraints and without affecting the start time of any other activity in progress. Let time t denote the earliest such time when activity a_i, in mode m_i, can be scheduled. Since a_i is ready at t, it belongs to one or more RSSs at t in its resource feasible mode m_i, and thus, is included in one (or more) RSS, for which a child state has been generated, say some state Z. If an optimal schedule is generated by a child state of parent X scheduling activity a_i in its mode m_i, at $t \geq dp_X$, then a schedule with at least the same makespan shall also be generated by state Z. Hence, state Y may be pruned. □

The optimality of best-first using the rules DP, 1C and LS can be established similarly.

4 EXPERIMENTAL OBSERVATIONS

We compare and present the results of our breadth-first and best-first algorithms with the most competitive exact algorithm for MM-RCPSP by Sprecher and Drexl (1998) which uses the depth-first approach. We refer to these algorithms as BRDMM, BSTMM, and DEPSDMM respectively. In BRDMM and BSTMM, all the three pruning rules described above have been applied. In DEPSDMM, we apply all the rules except the global left shift rule and the cut set rule II, as suggested by Sprecher and Drexl (1998), both being computationally too expensive.

Computational Machines: The development and initial experiments were carried out on a desktop machine (details provided below), while the tests were carried out using one CPU on a node in a high performance compute-cluster (HPCC) at the Physical Research Laboratory, Ahmedabad. The HPC has twenty-one nodes, each node with sixteen CPUs. Each compute-node is a collection of four boards, each with four Quad-Core AMD® Opteron™ Processor 8360 SE with 2511.578 MHz clock speed. At each node a total of 64 GB shared DDR2 SDRAM, 677 MHz is available, though in practice far less is used. The size of L1 cache is 128 KB, L2 cache is 512 KB, and L3 cache is 2048 KB. The core speed is 2500 MHz, integrated memory controller speed is 2000 MHz, and the system bus speed is 1000 MHz. The operating system used on the access server (the head node) is Red Hat Enterprise Linux 5 (RHEL 5), while on the compute nodes its light weight variant (or thin version without GUI for computational purposes) is deployed. All the algorithms were coded in C and compiled using Intel C Compiler without using any compile time parallelization or optimization directives. The details of computational machines used are as follows.

(A) Details of developmental and experimental operating systems used:
Operating System : (a) Fedora 11, 12, and 13 for development and testing.
: (b) RHEL 5 for experiments.
Compiler : GNU C Compiler (gcc), Intel C Compiler (icc).
Analysis tools : valgrind, Kcachegrind.

(B) Details of the developmental and testing platform (desktop) used:
Vendor_id : Intel Corp.
Model name : Intel® Pentium® D CPU 3.00GHz
CPU cores : 2 (only one CPU is deployed in the algorithms explained)
Core Speed (MHz) : 3000 (Max 4000 MHz)
L1 Cache Size (KB) : 2 x 16
L2 Cache Size (KB) : 2 x 1024
L3 Cache Size (KB) : Not provided.
System Bus Speed (MHz) : 533

(C) Details of compute-cluster platform used:
The computational experiments were conducted on a compute-cluster with sixteen processors, organized as four Quad-Core boards using AMD processors. The essential details are as given below.
Vendor_id : AuthenticAMD.
Model name : Quad-Core AMD® Opteron™ Processor 8360 SE
CPU cores : 4 (only one CPU is deployed in the algorithms explained)
Core Speed (MHz) : 2500
L1 Cache Size (KB) : 128
L2 Cache Size (KB) : 512
L3 Cache Size (KB) : 2048
System Bus Speed (MHz) : 1000
CMOS : 65nm SOI

IM Controller Speed (MHz) : 2000
Virtualization : Yes
Siblings : 4
Bogomips : 5026.04
TLB size : 1024 4K pages.
Address sizes : 48 bits physical, 48 bits virtual.

Standard Problem Sets: We use the established benchmark test problem sets from the PSPLIB for our experiments. The PSPLIB sets j10, j12, j14, j16, j18, and j20 have known optimal solutions, however, the largest problem set, j30, with thirty two activities in each problem instance, including two dummy activities (start and end), only has heuristic/metaheuristic based solutions known. Each of these problem sets includes random problem instances generated with varying resource factor (for both, renewable and non-renewable resources either 0.5 or 1.0) and resource strength (for both, renewable and non-renewable resources from 0.2, 0.5, 0.7 or 1.0) combinations. The infeasible problem instances generated were removed from the problem sets. No B&B

Table 9: Summary of Results of Computational Experiments on Desktop

	% Solved	Tot CPU time(s)	Mean time(s)	Std dev	Max time(s)	Solved <5min	Solved <10min	Solved <15min	Solved <30min
Problem set: PSPLIB j10 set (total number of problem instances 536)									
BRDMM	100%	1267	2.4	0.2	35.8	536	--	--	--
BSTMM	100%	155	0.3	0.03	8.7	536	--	--	--
CPLEX	100%	712	1.3	6.4	157.3	536	--	--	--
DEPMM	100%	955	1.8	0.1	27.6	536	--	--	--
Problem set: PSPLIB j12 set (total number of problem instances 547)									
BRDMM	100%	3591	6.6	0.5	148.9	547	--	--	--
BSTMM	100%	406	0.7	0.1	13.5	547	--	--	--
CPLEX	100%	979	1.8	7.8	189.3	547	--	--	--
DEPMM	100%	211947	387.5	28.9	8787	442	+62	+23	+17
Problem set: PSPLIB j14 set (total number of problem instances 551)									
BRDMM	99.80%	47977	87.2	7.9	1678.3	537	10	3	
BSTMM	100%	4133	7.5	1	273.4	551	--	--	--
CPLEX	100%	4512	8.2	84.4	2080.1	550	0	0	0
DEPMM	Unable to solve any problem within 30 minutes per problem								

Note: DEPSDMM is represented as DEPMM in the table. As the Desktop machine used had less (only 2 GB) RAM, larger problem sets could not be solved. CPLEX uses both of the two processors on the desktop machine.

exact solution algorithm has been able to solve these problem instances optimally beyond the size of set j12 in a reasonable time. The heuristic/metaheuristic approaches

too have not been able to solve all problem instances when activities considered are more than fourteen.

The runs were taken with varying wall clock time limits, such as, 300 seconds (5 min) for smaller problem instances and up to 1800 seconds (30 min) for larger problem instances. An instance was counted as solved only if the algorithm terminated with an optimal solution in the allocated time. All the three algorithms yielded an optimal solution on termination. Our single processor breadth-first and best-first tree-search algorithms are the first to solve even larger problem instances exactly in a reasonable time.

Single Processor Breadth-first, Best-first and Depth-first Results: The entire set j10 from PSPLIB was solved by all the three algorithms optimally and on average breadth-first is slowest on these small problem instances, followed by depth-first, while the fastest is best-first. It appears that the time taken in processing the pruning rules is computationally expensive for smaller problem instances because of which breadth-first approach becomes slower compared to other approaches.

However, the benefit of investment of time in the pruning rules mentioned above appears substantial in the next larger set j12, where breadth-first performs better than depth-first, though best-first remains at top. The depth-first algorithm is unable to solve all the problem instances in a limited run time of ten minutes per instance, while Breadth-first and Best-first solve all problem instances within ten minutes each. A summary of all results in experiments on a Desktop machine and in HPCC are provided in Table 9 and Table 10.

The Desktop machine had only 2 GB RAM, and the experiments were carried out with a RAM limit of 1792 MB leaving 256 MB for the operating system. Due to this limitation none of the larger problem sets could be solved. To arrive at complete results for the set j12, it was experimented without any time limit for the Depth-first algorithm, and the break-up of numbers of problems solved along with time taken are shown in Table 9. In the set j14, Breadth-first is unable to solve any problem on the desktop machine either due to RAM availability limitation or exceeding the set time limit of 30 minutes per problem instance. However, Best-first solves a substantial number of these problems.

On the HPCC, all algorithms solve the problem set j10 within five minutes of total run time. However, much more time is needed from the set j12 onwards. Depth-first solves only the j10 set faster than Breadth-first, while Best-first is clearly the fastest. A rapid and significant jump in average computational time is observed for both Breadth-first and Depth-first as the problem instance's size increases, while Best-first appears to be influenced less. Problem set j14 onwards Depth-first is able to solve only a few of the problem instances within ten minutes, though both, breadth-first and best-first solve all within this time. The advantage of computational time consumed in pruning rules in

Breadth-first, which causes it to be slower than Depth-first in the smallest set j10, appears significant in larger problem sets, making it faster.

In the set j16, breadth-first is unable to solve all problem instances within ten minutes, however best-first continues to solve all of these and most of even the j18 set within ten minutes. A small number of problem instances from the set j18 seems to take longer time in solving with best-first. Thus, both our algorithms solve much larger problem instances on a single processor implementation in reasonable time. Very few problem instances are solved with a maximum run time limit of fifteen minutes in the set j20, though, notably some instances of even the set j30 are solved within fifteen minutes of run time. Our algorithm appears to be the first exact tree-search algorithm to solve any problem instances in the set j30 in a reasonable time. All known bounds or solutions generated for the j30 set prior to our algorithm are through metaheuristic or other inexact approaches. displays the results of performance of our algorithms on the HPCC using one processor.

Table 10: Summary of Results of Computational Experiments on HPCC

	% Solved	Tot CPU time(s)	Mean time(s)	Std dev	Max time(s)	Solved <5min	Solved <10min	Solved <15min	Solved <30min		
Problem set: PSPLIB j10 set (total number of problem instances 536)											
BRDMM	100%	299	0.6	0.8	6.1	536	--	--	--	--	--
BSTMM	100%	47	0.1	0.2	2.3	536	--	--	--	--	--
CPLEX	100%	712	1.3	6.4	157.3	536	--	--	--	--	--
DEPMM	100%	228	0.4	0.7	7.9	536	--	--	--	--	--
Problem set: PSPLIB j12 set (total number of problem instances 547)											
BRDMM	100%	971	1.8	3.1	26.9	547	--	--	--	--	--
BSTMM	100%	120	0.2	0.5	4	547	--	--	--	--	--
CPLEX	100%	979	1.8	7.8	189.3	547	--	--	--	--	--
DEPMM	100%	15851	29	63.7	589.8	539	8	--	--	--	--
Problem set: PSPLIB j14 set (total number of problem instances 551)											
BRDMM	100%	17841	32.4	70.6	463.8	537	14	--	--	--	--
BSTMM	100%	1515	2.8	9.9	118.5	551	--	--	--	--	--
CPLEX	100%	4512	8.2	84.6	2080.1	550	0	0	0	1	--
DEPMM	38.80%	38703	180.8	147.9	521.8	164	50	0	--	--	--
Problem set: PSPLIB j16 set (total number of problem instances 550)											

BRDMM	99.10%	192071	350.5	1011.6	11698	434	43	18	28	14	8
BSTMM	100%	8041	14.6	40.3	360	548	2	--	--	--	--
CPLEX	100%	4514	8.2	75.3	1838	549	0	0	0	1	--
Problem set: PSPLIB j18 set (total number of problem instances 552)											
BRDMM	64.10%	472785	1335.6	1623.3	6974.7	116	58	23	77	42	38
BSTMM	98.20%	87099	160.4	707.4	11731	494	17	7	15	7	2
CPLEX*	99.80%	12308	22.3	165.2	2310.3	548	0	0	1	3	
Problem set: PSPLIB j20 Set (total number of problem instances 554)											
BRDMM	33.40%	252796	1366.5	1715.4	7080	54	35	21	32	19	24
BSTMM	88.60%	356797	706.5	3032.1	30297	389	38	22	32	4	6
CPLEX	100%	17299	31.2	182.9	2230	548	0	0	3	1	--
Problem set: PSPLIB j30 set (total number of problem instances 640)											
BRDMM	4.70%	15972	532.4	521.7	1711	13	8	2	7	0	0
BSTMM	10.20%	31386	482.9	509.9	1719	36	8	4	17	0	0

*CPLEX could not solve one problem in this set.

Note: DEPSDMM is represented as DEPMM in the table. CPLEX results shown are as obtained in runs on the desktop machine.

Table 11: Problem Instances with the Largest Number of States Obtained using Breadth-first

In problem instance	States Generated	States Explored	States Dominated	States Deleted	States Removed
For the problem set PSPLIB j10:					
j1019_5	**17,416,684**	237,539	6,690,486	363,724	556,136
j1062_10	14,191,387	**269,258**	5,974,158	370,491	590,403
j1055_1	15,542,073	260,793	**6,796,004**	**410,051**	**616,002**
Note: In set j10, in the same problem instance, the largest number of states dominated, deleted, and removed is found.					
For the problem set PSPLIB j12:					
j1251_10	**34,255,709**	653,955	**16,355,118**	986,054	1,497,558
j1219_3	28,231,171	**698,292**	12,307,604	**1,056,126**	**1,594,634**
Note: The largest number of states generated and dominated is found in the problem instance j1251_10, while the largest numbers of states explored, deleted, and removed is also found in the problem instance j1219_3.					
For the problem set PSPLIB j14:					
In problem instance	**States Generated**	**States Explored**	**States Dominated**	**States Deleted**	**States Removed**
j1455_5	**251490330**	**3939909**	124638289	**5753504**	**9032916**
j1446_5	231255764	3342564	**132374587**	5057043	7742838
Note: The largest number of states generated, explored, deleted, and removed are found in the same problem instance, i.e. j1455_5.					
For the problem set PSPLIB j16:					
j1647_6	**1584311474**	**7618709**	**636208223**	**18306916**	**24874085**
Note: All the largest numbers of states generated, explored, dominated, deleted, and removed are found in the same problem instance, i.e. j1647_6.					
For the problem set PSPLIB j18:					
j1850_3	**2946868090**	11624006	**1141962866**	**27209130**	**37429162**
j1847_2	1368580836	**12322763**	696850240	21709808	31794921
Note: The largest number of states generated, dominated, deleted, and removed is found in the same problem instance.					
For the problem set PSPLIB j20 (from problems solved in 120 minutes):					
j2044_2	**1754568734**	9023362	**659423602**	**21446638**	**29354623**
j2046_5	771246560	**9936944**	412764264	13820466	22048856
Note: The largest number of states generated, dominated, deleted, and removed is found in the same problem instance.					
For the problem set PSPLIB j30 (from problems solved in 120 minutes):					
j3031_6	**6189725029**	7053502	**2504759505**	17136021	23418600
j3029_2	4184151142	**11296791**	1578857738	14652327	24483610
j3029_4	4149619341	10605901	1414568659	**18000582**	**27593550**
Note: The largest number of states generated and dominated is found in the same problem instance j3031_6, while the largest number of states deleted and removed is found in the problem instance j3029_4.					

We also observe that the problem instances taking larger time to solve are also the ones which have a large number of available optimal solutions (from which multi objective solutions can be found). It appears that if only one optimal makespan solution exists for a problem instance, then the last few levels of the tree are quite rapidly processed. However, if the problem instance has multiple single objective (makespan) optimal solutions, then many more branches need to be evaluated and processed in the tree even at the last few levels (i.e. the tree is larger or broader even at these levels). Hence, greater the number of *different* optimal makespan schedules for a problem instance,

longer is the time taken by the Breadth-first algorithm to solve it. Two problem instances in the set j16 (j1618_ 9 and j1623_ 5), which were not solved within 120 minutes, when run without time limit, were solved in 22917 seconds (about 6.37 hours) and 37583 seconds (about 10.44 hours), and yielded 467 and 333 optimal makespan solutions respectively!

The largest number of states generated when solving by the Breadth-first algorithm using all three pruning rules in the PSPLIB problem set j10 is for the instance j1019_5, which is 17,416,684 states; and in problem set j12 is for the instance j1251-10, which is 34,255,709 states. In Best-first algorithm, using all three pruning rules, the largest number of states generated in problem set j10 is for the instance j1062_10, which is 8,004,393; and in problem set j12 is for the instance j1240_9, which is 10,700,108. For all the problem sets from PSPLIB, we provide in tables below, the largest number of states generated, explored, dominated (a newly developed child state pruned as dominated by an already existing state by the Dominance Pruning rule), deleted (an existing state dominated by a new child state now generated), and removed (a parent state removed to conserve memory, as all child states of this parent state are pruned by the pruning rules). The largest figures obtained are indicated in bold font in Table 11.

In Table 12, we present the numbers of states generated by the Best-first algorithm. The largest figures obtained are indicated in bold font in the table.

Table 12: Problem Instances with the Largest Number of States Obtained using Best-first

In problem instance	**States Generated**	**States Explored**	**States Dominated**	**States Deleted**	**States Removed**
For the problem set PSPLIB j10:					
j1062_10	**800,4393**	**105,397**	**3,383,340**	**302,346**	**81,957**
Note: The largest number of states generated, explored, dominated, deleted, and removed is found in the same problem instance.					
For the problem set PSPLIB j12:					
j1240_9	**10,700,108**	209,432	**6,722,593**	**436,497**	**209,158**
j1235_2	7,846,486	**221,539**	4,128,090	191,954	145,846
Note: The largest number of states explored is found in a different problem instance, i.e. j1235_2, and all the other largest numbers of states are found in the same problem instance j1240_9.					
For the problem set PSPLIB j14:					
j1439_2	**186065445**	**2150247**	**91357911**	**2762042**	**2317348**
Note: The largest number of all figures is found in the same problem instance, i.e. j1439_2.					
For the problem set PSPLIB j16:					
j1647_6	**864471559**	3076614	**366124329**	**9326773**	4653106

In problem instance	States Generated	States Explored	States Dominated	States Deleted	States Removed
j1637_3	321793242	**4398419**	177707312	4366496	3781753
j1635_8	603130992	3672012	304294627	8392050	**6328589**
Note: The largest number of states generated, dominated, deleted are found in the same problem instance j1647_6, while the largest number of states explored is found in the instances j1637_3 and removed in j1635_8.					
For the problem set PSPLIB j18 (from problem instances solved in 120 minutes):					
j1834_2	**4609491105**	13156891	**2243535572**	**48244017**	16349704
j1837_1	1302898364	**19507176**	670536970	16479579	13992127
j1839_9	2184580116	15587632	1167184861	24995359	**21263830**
Note: The largest number of states generated, dominated, and deleted is found in the same problem instance j1834_2.					
For the problem set PSPLIB j20 (from problems solved in 120 minutes):					
j2048_9	**3060847833**	7157750	**1478656085**	35522008	8070131
j2042_10	2496103758	**10675392**	1305062706	23429340	11839840
j2014_4	1551929189	4318412	628307423	**35875626**	2519364
j2034_1	1633433279	10316459	891251469	23998752	**15619142**
Note: The largest number of states generated and dominated is found in the same problem instance j2048_9.					
For the problem set PSPLIB j30 (from problems solved in 120 minutes):					
j3029_7	**4912496538**	9359656	**1604225024**	**22654339**	**11976806**
j3029_2	4125869694	**9636760**	1498937670	11498230	8418726
Note: The largest number of states generated, dominated, deleted and removed is found in the same problem instance j3029_7.					

We briefly discuss the generalization of the algorithms to other regular measures in the following section.

Generalization to Other Regular Measures: The breadth-first scheme for regular measures can be modified to cater to problem sets with more restrictions, such as, activities with a non-zero ready time. However, minor changes to the pruning rules would be necessary in order to solve problems with an objective of minimizing the mean flowtime given by mft =/ (N – 2), where b_i is the ready time for activity a_i. The 1C rule can not be used for this objective, as it may yield sub-optimal solutions. Further, the DP has to be altered to a weaker form as follows: If at any time during the execution of breadth-first there are two states X and Y in the search tree such that:

$F_X = F_Y$, i.e., the activities completed are same;

$A_X = A_Y$, i.e. activities in progress and their corresponding modes are same;

the residual of each non-renewable resource, after consumption by all activities completed or in progress in set X, is same or more than in set Y;

the starting time in state X of each activity in A_X is less than or equal to its starting time in state Y;

$|$ a_i is in $F_X \leq |$ a_i is in F_Y ;

then prune state Y from the search tree, as state X dominates state Y.

This ensures retention of only left-aligned schedules making the LS rule unnecessary, though retaining the rule helps pruning some states earlier and saving overall computational effort. The generalization to due date based measures, such as minimization of maximum tardiness and minimization of number of tardy jobs, where early completion has no penalty, is also straight forward.

Maximum tardiness, T_{max}, is computed as $T_{max} = \max \{0, f_i - d_i : 1<i<N\}$, where d_i is the due date for the activity a_i. It represents the amount of delay in the most delayed of all non-dummy activities/jobs from their respective due dates. The application for this measure too requires removal of the 1C rule, and modification to the DP rule as follows: If at any time during the execution of breadth-first there are two states X and Y in the search tree such that:

$F_X = F_Y$, i.e., the activities completed are same;

$A_X = A_Y$, i.e. activities in progress and their corresponding modes are same;

the residual of each non-renewable resource, after consumption by all activities completed or in progress in set X, is same or more than in set Y;

the starting time in state X of each activity in A_X is less than or equal to its starting time in state Y;

$\max \{0, f_i - d_i \mid a_i \text{ is in } F_X\} \leq \max \{0, f_i - d_i \mid a_i \text{ is in } F_Y\}$;

then prune state Y from the search tree.

The best-first algorithm can be similarly adopted for regular measures.

5 SUMMARY

In this paper we present the algorithms breadth-first and best-first, along with pruning rules, for solving the MM-RCPSP with renewable and non-renewable resources for regular measures of performance (makespan). Both of our approaches solve the MM-RCPSP optimally. Proof of optimality for the makespan version of breadth-first has been provided in detail. The ability of breadth-first approach to generate multiple optimal solutions, and among these, yield an exact multi objective optimal solution is described. Experimental results of tests on standard problem sets from the PSPLIB are presented along with comparison of performance with the best known exact solution tree-search algorithm of Sprecher and Drexl (1998) and with CPLEX. Both Breadth-first and Best-first are faster than Depth-first.

6 REFERENCES

Alcaraz, J., C. Maroto, and R. Ruiz, 2002. A New Genetic Algorithm for the Multi-Mode Resource-Constrained Project Scheduling Problem. Dpto de Estadistica e Investigacion Operativa, Universidad Politecnica de Valencia, Spain.

Alcaraz, J., C. Maroto, and R. Ruiz, 2003a. Multi-Mode Resource-Constrained Project Scheduling Problem: An Advanced Genetic Algorithms. The Fifth Metaheuristics Internatinal Conference, Kyoto, Japan.

Alcaraz, J., C. Maroto, and R. Ruiz, 2003b. Solving the Multi-Mode Resource-Constrained Project Scheduling Problem with Genetic Algorithms. The Journal of the Operational Research Society, 54(6): 614-626.

Ballestin, F., and Blanco, R., 2011. Theoretical and practical fundamentals for multi-objective optimisation in resource-constrained project scheduling problems. Computers & Operations Research, 38(1): 51-62.

Berman, E. B., 1964. Resource Allocation in a Pert Network under Continuous Activity Time-Cost Functions. Management Science, 10(4): 734-745.

Bouleimen, K. and H. Lecocq, 2003. A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. European Journal of Operational Research, 149(2): 268-281.

Chyu, C.C., Chen, A.H.L., and Lin, X.H., 2005. A Hybrid Ant Colony Approach to Multi- mode Resource-Constrained Project Scheduling Problems with non-renewable types. *1st* International Conference on Operations and Supply Chain Management.

De Reyck, B., E. Demeulemeester, and W. Herroelen, 1998. Local search methods for the discrete time/resource trade-off problem in project networks. Naval Research Logistics, 45(6): 553-578.

De Reyck, B. and W. Herroelen, 1999. The multi-mode resource-constrained project scheduling problem with generalized precedence relations. European Journal of Operational Research, 119(2): 538-556.

Demeulemeester, E. and W. Herroelen, 1992. A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. Management Science, 38(12): 1803-1818.

Drexl, A. and J. Gruenewald, 1993. Non-Preemptive Multi-Mode Resource Constrained Project Scheduling. IIE Transactions, 25(5): 74-81.

Erenguc, S. S., T. Ahn, and D.G. Conway, 2001. The resource constrained project scheduling problem with multiple crashable modes: An exact solution method. Naval Research Logistics, 48(2): 107-127.

Hartmann, S., 2001. Project Scheduling with Multiple Modes: A Genetic Algorithm. Annals of Operations Research, 102(1): 111-135.

Hartmann, S. and A. Drexl, 1998. Project Scheduling with Multiple Modes: A Comparison of Exact Algorithms. Networks, 32: 283-297.

Heilmann, R., 2001. Resource–constrained Project Scheduling: A Heuristic for the Multi–mode Case. OR-Spektrum, 23(3): 335-357.

Heilmann, R., 2003. A Branch-and-bound Procedure for the Multi-mode Resource-constrained Project Scheduling Problem with Minimum and Maximum Time Lags. European Journal of Operational Research, 144(2): 348-365.

Józefowska, J., M. Mika, R. Rózycki, G. Waligóra, and J. Weglarz, 1999. Solving the Multi-Mode Resource-Constrained Project Scheduling Problem by Simulated Annealing. Seventh International Workshop on Project Management and Scheduling.

Józefowska, J., M. Mika, R. Rózycki, G. Waligóra, and J. Weglarz, 2001. Simulated Annealing for Multi-Mode Resource-Constrained Project Scheduling. Annals of Operations Research, 102(1): 137-155.

Kolisch, R. and A. Sprecher, 1997. PSPLIB - A project scheduling problem library : OR Software - ORSEP Operations Research Software Exchange Program. European Journal of Operational Research, 96(1): 205-216.

Kolisch, R., A. Sprecher, A. Drexl, 1995. Characterization and Generation of a General Class of Resource-Constrained Project Scheduling Problems. Management Science, 41(10): 1693-1703.

Leachman, R., 1980. Multiple Resource Leveling in Construction Systems through Variation of Activity Intensities. Retrieved May 23, 2008 from:
http://stinet.dtic.mil/oai/oaiverb=getRecord&metadataPrefix=html&identifier=ADA089988

Leachman, R., A. Dincerler, and S. Kim, 1990. Resource-Constrained Scheduling of Projects with Variable-Intensity Activities. IIE Transactions, 22(1): 31-40.

Mika, M., Waligóra, G., and J. Weglarz, 2008. Tabu search for multi-mode resource-constrained project scheduling with schedule-dependent setup times. European Journal of Operational Research, 187(3): 1238-1250.

Mori, M. and C. Tseng, 1997. A genetic algorithm for multi-mode resource constrained project scheduling problem. European Journal of Operational Research, 100(1): 134-141.

Nazareth T., and Bhattacharya S., 1993. A breadth-first search scheme to solve the resource constrained project scheduling problem, Proc IFIP 93 on Systems Modelling and Optimizaiton, Compeigne, France, July 5-9, 641-644.

Nazareth, T., 1995. New Algorithms for the Multiple-Resource Constrained Project Scheduling Problem, Unpublished Fellow Programme Thesis. Indian Institute of Management, Calcutta.

Nazareth T., Verma S., Bhattacharya S., & Bagchi, A., 1999. The multiple resource constrained project scheduling problem: A breadth-first approach. European Journal of Operational Research, 112(2): 347-366.

Patterson, J. H., 1984. A Comparison of Exact Approaches for Solving the Multiple Constrained Resource, Project Scheduling Problem. Management Science, 30(7): 854-867.

Peteghem, V. V. and M. Vanhoucke, 2008. A Genetic Algorithm for the Multi-Mode Resource Constrained Project Scheduling Problem. Working Papers of Faculty of Economics and Business Administration, Ghent University Ghent University, Belgium. Retrieved May 23, 2008 from:
http://www.FEB.UGent.be/fac/research/WP/Papers/wp_08_494.pdf

Peteghem, Vincent Van, and Mario Vanhoucke, 2010. A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem. European Journal of Operational Research, 201(2): 409-418.

Reddy, J. P., Kumanan, S., & Chetty, O. K. (2001). Application of Petri nets and a genetic algorithm to multi-mode multi-resource constrained project scheduling. The International Journal of Advanced Manufacturing Technology, 17(4), 305-314.

Sabzehparvar, M. and S. Seyed-Hosseini, 2008. A mathematical model for the multi-mode resource-constrained project scheduling problem with mode dependent time lags. The Journal of Supercomputing, 44(3): 257-273.

Schirmer, A., 1996. New Insights on the Complexity of Resource-Constrained Project Scheduling-Two Cases of Multi-Mode Scheduling. Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, 391. Retrieved May 23, 2008 from:
http://citeseer.ist.psu.edu/46793.html

Shan, M., Q. Hong, and W. Juan, 2007. Multi-Mode Multi-Project Scheduling Problem for Mould Production in MC Enterprise. Wireless Communications, Networking and Mobile Computing, 2007 (WiCom 2007), International Conference on: 5311-5315.

Shan, M., J. Wu, and D. Peng, 2007. Particle Swarm and Ant Colony Algorithms Hybridized for Multi-Mode Resource-constrained Project Scheduling Problem with Minimum Time Lag. Wireless Communications, Networking and Mobile Computing, 2007 (WiCom 2007), International Conference on: 5893-5897.

Sprecher, A. and A. Drexl, 1998. Solving Multi-mode resource-constrained project scheduling by a simple, general and powerful sequencing algorithm. European Journal of Operational Research, 107(2): 431-450.

Sprecher, A., Kolisch, R., and A. Drexl, 1995. Semi-active, active, and non-delayschedules for the resource-constrained project scheduling problem. European Journal of Operational Research, 80(1): 94-102.

Sprecher, A., Hartmann, S., and A. Drexl, 1997. An exact algorithm for project scheduling with multiple modes. OR Spectrum, 19(3): 195-203.

Talbot, F. B., 1982. Resource-Constrained Project Scheduling with Time-Resource Trade offs: The Non-preemptive Case. Management Science, 28(10): 1197-1210.

Ulusoy, G. and S. Sahin, 1998. Three Different Payment Programs for the Multi-Mode Resource Constrained Project Scheduling Problem with Discounted Cash Flows: A Genetic Algorithm Approach. Retrieved May 23, 2008 from:
http://www.mathematik.uni-osnabrueck.de/research/OR/pms2000/abstract/ulusoy90.tr.ps

Ulusoy, G., S. S. Funda, and S. Sahin, 2001. Four Payment Models for the Multi-Mode Resource Constrained Project Scheduling Problem with Discounted Cash Flows. Annals of Operations Research, 102(1-4).