Exact Methods for the Preemptive Resource-Constrained Project Scheduling Problem

Sanjay Verma

W.P. No. 2006-03-08
March 2006

INDIAN INSTITUTE OF MANAGEMENT
AHMEDABAD-380 015
INDIA

Exact Methods for the Preemptive Resource-Constrained Project Scheduling Problem

Prof. Sanjay Verma[1]

Abstract

A simple best-first tree search scheme with pruning rules to minimize the completion time (makespan) of the project is described. A project consists of a set of activities partially ordered by precedence constraints. An activity has a given non-negative duration and uses renewable resources such as manpower and machinery. The total number of available units of each resource is constant and specified in advance. A unit of resource cannot be shared by two activities. An activity is ready to be processed only when all its predecessor activities are completed and the numbers of units of the various resources required by it are free and can be allocated to it. Once started, an activity can be interrupted and rescheduled later on without any increase in remaining duration of that activity. Each such part of the activity can be called a segment of that activity. There are no set-up times. The objective is to assign start times to the activities or segment of activities so that the makespan is minimized.

1. Introduction

A *project* consists of a set of activities partially ordered by precedence constraints. An activity has a non-negative duration and uses different types of renewable resources such as manpower and machinery. The total number of available units of each resource type is constant and specified in advance. Two activities cannot simultaneously make use of the same unit of resource. 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 are free and can be allocated to it. In the non-preemptive case, once started an activity is not interrupted and runs to completion. One of the objectives is to minimize the completion time (makespan) of the project. *In the preemptive case which is discussed in this paper, an activity can be interrupted any number of times. However, this preemption is allowed at unit time intervals only.*

Extensive work on resource constrained project scheduling problem can be found in Stinson *et al*. [1978], Christofides *et al*. [1987], Bell and Park [1990], Demeulemeester and Herroelen [1992], and Nazareth *et al.* [1999]. However, very little literature is available on solving project-scheduling problem when activities can be preempted to resume later on, so that some other activities can be executed. Demeulemeester and Herroelen [1999] presented an algorithm for the preemptive case. However, experiments were conducted on standard set of Patterson [1984] only, which are very small compared to the new standard sets, by Kolisch *et al.* [1995]. In Verma [2004] a breadth first algorithm for the preemptive case was explained. The objective of this paper is to present an optimal best-first algorithm to solve the preemptive resource-constrained project scheduling problem, and to show the results on the standard set of Kolisch *et al.* [1995].

Section 2 of the paper introduces the basic terminology and notation, and gives the mathematical formulation of the problem. Section 3 reviews the existing literature. Section 4 explains the operation

[1] The author is thankful to Prof. Amitava Bagchi, Professor in Management Information Systems at Indian Institute of Management Kolkata, Calcutta, India for his valuable guidance. The author thanks Dr. Erik Demeulemeester, from Department of Applied Economics, Operations Management Group Katholieke Universiteit Leuven, Belgium for providing source code of their programs. This paper was possible due to research grant by Research and Publications Unit of IIM Ahmedabad.

of Preempt_BFS with the help of examples. Section 5 explains a Breadth First Implementation of Preempt_BFS called Preempt_BDS. Section 6 details our experimental observations. Preempt_BFS, Preempt_BDS and a very popular algorithm for same problem by Demeulemeester E. & Herroelen W. [1996] are coded in C and executed on a Linux based machine. The results of experiments conducted over standard set of Patterson as well as, that of Kolisch *et al* [1995] are recorded. Section 7 suggests further work on the problem and concludes the paper. Section 8 provides references.

2. Definitions of Terms

1. **Project:** A project consists of **N** activities $a_1, a_2, ..., \mathbf{a_i}, \ldots, a_N$. Activity a_i has duration of $\mathbf{p_i}$ units; this includes the set-up time, processing time and set-down time. We use the Activity-on-Node (AON) convention when referring to projects.

2. **Precedence Constraints:** Activity a_i ($i = 1, \ldots, N$) can start only when all its *predecessor* activities have finished. The predecessors are determined by the technological considerations of the project. An activity a_p is said to be a predecessor of a_i, when a_i cannot start until a_p has finished. This is represented as $a_p < a_i$, where '<' defines the *precedes* relationship. Similarly a_s is said to be a *successor* activity of a_i if a_s cannot begin until a_i has finished. Let *H* denote the set of all the pairs of activities with predecessor and successor relationships.

3. **Resource Types**: M types of renewable resources are assumed to be available. R_j (j=1,.. M) denotes the *total availability in number of units* of resource type j. Activity $\mathbf{a_i}$ requires r_{ij} units of the **j**$^{\mathbf{th}}$ resource.

4. **Resource Constraints:** The total number of units of resource type j used by all the activities in progress at any instant of time should not exceed the total availability R_j of that resource type.

5. **Integrality Condition**: Values of parameters such as activity duration (p_i), resource availability (R_j) and resource requirements (r_{ij}) are *non-negative integers*.

Note: Without loss of generality and in consistency with standard practice, it is assumed that:

a) A project has 2 *dummy* activities, a (unique) dummy start activity a_1 and a (unique) dummy finish activity a_N, which are of zero duration and consume no resources (*i.e.*, $p1 = p_N = 0$, $r_{1j} = r_{Nj} = 0$, j, $j = 1,\ldots, M$).

b) The activities are numbered in such a way that no activity has a predecessor with a higher number.

c) Every non-dummy activity has at least one predecessor and at least one successor,

d) In listing the set of predecessor activities of a given activity, only the activities directly preceding need to be listed. If the direct predecessors have completed, all indirect predecessors must also have completed.

A project starts at time $t = 0$ (*i.e.*, $s_1 = 0$). A *schedule* for the project is an assignment of a start time s_i to each activity a_i. An activity is said to be scheduled when it is assigned a start time. The *makespan* ($T = f_N$) of a schedule is the time when the last activity (a_N) finishes. 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.

Given R_j ($j = 1, .. M$), and p_i, *H* and r_{ij} for each a_i, ($i = 1, \ldots, N$, $j = 1, \ldots, M$), our problem is to determine an optimum schedule. In the widely discussed resource-constrained project-scheduling problem (RCPSP), activities once started are executed unto their completion. The problem can be formulated mathematically as follows:

Minimize f_N (1)

subject to the conditions

i) $f_i - f_j \geq p_i \ \forall (a_i, a_j) \in H$; *and* (2)

ii) $\sum r_{ij} \leq R_j$ for each j, $1 \leq j \leq M$, at every integer time instant t, $0 \leq t < f_N$, (3)

where the summation is over all i such that activity a_i is in progress during the time interval [t, t+1).

However, when preemptions are allowed, the activities can be interrupted at any integer time instant and restarted later without any setup cost, i.e. an activity (a_i) with duration (p_i) can be splitted into p_i segments of unit duration where each segment consumes same resources as that of activity a_i and two segments cannot run in parallel. Following formulation is discussed in Demeulemeester and Herroelen [1999].

Let f_{ik} be the completion time of k^{th} unit of activity a_i, where each activity a_i is broken into p_i durations. Let f_{i0} be the earliest start time of the activity a_i. Only finish-start relations with a time lag of zero are allowed, and therefore f_{i0}, equals the latest finish time of all the predecessors of activity a_i. An activity a_i belongs to the set of activities in progress at time t if one of its duration units k = 1,2,.....,p_i finishes at time t. With these, PRCPSP can be formulated as follows:

Minimize $f_{n,0}$ (4)

i) $f_{i,di} \leq f_{j,0} \ \forall (a_i,a_j) \in H$ (5)

ii) $f_{i,k-1} + 1 < f_{i,k}$ for i = 1,.....n and k = 1,......d_i (6)

iii) $f_{1,0} = 0$, and (7)

iii) $\sum r_{ij} \leq R_j$ for each j, $1 \leq j \leq M$, at every integer time instant t, $0 \leq t < f_{N,0}$ (8)

where the summation is over all i such that activity a_i is in progress during the time interval [t, t+1).

The objective function (4) minimizes the makespan by minimizing the earliest start time of activity a_N (dummy activity). In (5) all precedence relationships are satisfied; the earliest start time of an activity a_j cannot be smaller than the finish time of the last unit of duration of its predecessor a_i.(6) specify that the finish time of a portion of activity at least one unit of time more that the completion time of previous portion. (7) specify that the earliest start time of activity a_i is 0. (8) specifies that the resources consumed at any point of time during the duration of the project are not greater that the resources available. It should be noted that with preemption the number of possible solutions increase and therefore the computational complexity of the problem also increases.

3. Literature Review

Significant work has been done in the field of project scheduling. A comprehensive survey of the work that has been done can be found in Herroelen *et al*. [1998]. In RCPSP the objective is to minimize the makespan of the project, where activities have deterministic duration and resource requirements. The resource requirement as well as, resource availability is given and remains constant throughout the duration of the project. Some of the noteworthy publications in this field are by

Stinson *et al*. [1978], Demeulemeester and Herroelen [1992], and Nazareth *et al*. [1999]. Demeulemeester and Herroelen [1992] use a depth first strategy. It generates a search tree, in which the nodes correspond to partial schedules. At each node finish times are assigned to a subset of activities in the project. DH uses the concept of Minimal Delaying Alternatives (MDA). A delayed set consists of all subset of activities that are either in progress or are eligible, the delay of which would resolve the resource conflict in the partial schedule. Pruning rules are also used.

Nazareth *et al*.[1999] have used two strategies, a breadth-first and a best first strategy. The concept of Maximal Resource Satisfying Set (MRS) is used. An MRS is considered out the candidate set. A candidate set is those activities that are available for scheduling. A MRS is a maximal subset of activities that are eligible to be scheduled and does not cause a resource violation; however if another activity belonging to candidate set is added to an MRS, the resulting subset would cause a resource violation. Nazareth *et al*.[1999] also use three pruning rules, namely Dominance Pruning rule, Left shift rule and One child rule.

However, not enough work has been done in the area of PRCPSP. Davis and Heidorn [1971] suggested and implicit enumeration scheme based on the splitting of activities into sub-activities of unit duration. The algorithm of Demeulemeester and Herroelen [1996] has also been extended for PRCPSP. IN PRCPSP, Demeulemeester and Herroelen (DH) make a distinction between activities and sub-activities. Each activity in the project network is replaced by sub-activities; their number being equal to the duration of the activity. Each sub-activity has duration of 1 and resource requirement equal to the corresponding activity. In Nazareth [1995] it is suggested that it is possible to modify the algorithms discussed in Nazareth *et al.* [1999] for the preemptive case. In this paper, the same idea has been taken further.

4. Preempt_BFS: A Best-First Strategy

Like most other project scheduling methods, Preempt_BFS is a tree-search procedure augmented with pruning rules. The nodes in the search tree are called *states* and correspond to partial schedules, where a *partial schedule* is a schedule of a subset of the N activities that do not violate any of the given precedence and resource constraints. A *complete schedule* is a partial schedule of all the N activities; a state corresponding to a complete schedule is a *solution state*. A state is identified with its partial schedule when no confusion is likely to arise; for clarity, we sometimes refer to the partial schedule corresponding to a state X as schedule(X). The root node of the search tree corresponds to a partial schedule with no activity completed and the dummy start activity a_1 in progress. The following parameters are associated with a state X:

c_X *Current time*: The time of creation of state X.

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, whose segment started at time c_X; this is the set of segments (of activities) in progress in state X at time c_X.

dp_X *Decision point*: The time at which we make consider new activities for scheduling. This becomes the current time of each child state of X.

K_X *Decision set*: The set of activities, which are not yet completed at time c_X but all of whose predecessors have completed at some time $\leq c_X$. These are the ready activities at time c_X. Activities in A_X, also belong to K_X.

4.1 Generation of Root State

The parameters associated with the root state I of the search tree are as follows:

c_I	=	0	=	Current time of the root state I
F_I	=	{ }	=	Set of activities completed at c_I (empty set)
A_I	=	$\{a_1\}$	=	Set of activities in progress at time c_I
dp_I	=	0	=	Finish time of activity a_1
K_I	=	$\{a_1\}$		

Initially the search tree consists only of the root state I. States get added to the tree as follows. Suppose X is a state in the tree that is not a solution state and X is selected for expansion. First the decision point dp_X and candidate set K_X are determined. Let us suppose that K_X is non-empty. A *Maximal Resource Satisfying Set* (MRS) is a maximal subset of K_X that does not cause a resource violation; if another activity belonging to K_X is added to an MRS, the resulting subset would cause a resource violation [Nazareth *et al.* 1999]. In general, K_X has a number of distinct subsets each of which is an MRS; these are not necessarily disjoint. For example, suppose there is only one resource type with a total availability of two units. Also suppose that in schedule(X), activities a_j and a_k are in progress, and a_j finishes at time dp_X but a_k is still in progress. Let a_m and a_n be two other activities that are ready to be scheduled at dp_X, so that $K_Y = \{a_k, a_m, a_n\}$. Let each of a_k, a_m and a_n require one unit of resource. Then, at dp_X the MRSs are $\{a_k, a_m\}$, $\{a_k, a_n\}$ and $\{a_m, a_n\}$. The expansion of X, which takes place at time dp_X, creates a child node corresponding to each MRS. In every child state Y of X

c_Y	=	dp_X
F_Y	=	F_X augmented by the activities in A_X that completed at time dp_X
A_Y	=	the MRS of X that corresponds to this child state

Since the activities are pre-empted, these will be scheduled many times.

4.2 Expansion of a State

Let the *level* of a state X in the search tree be the decision point of the state dp_X. The next state is generated at time $c_Y = dp_X = c_X+1$. However, if the duration of any activity in progress (A_X) is zero than $c_Y = c_X$ (and $dp_X = dp_Y$) and therefore, level of both parent and child states will be the same.

The root state is at level zero; its children are at level one, and so on. In the best-first formulation, the evaluation function for the state Y employs the usual makespan heuristic. A partial schedule is converted to a pseudo-complete one by adding the unfinished activities to the schedule in conformity with the precedence constraints but ignoring the resource constraints. The makespan of this schedule is used as the heuristic estimate of the state. The states are maintained in a priority queue with smaller estimates having higher priority, and in each iteration the state with the highest priority is selected for expansion. In case of ties, states with larger F_X sets get preference; if the F_X sets are equal in size, states with lower dp_X values are preferred. Since the heuristic value underestimates the actual makespan, the first solution state selected yields a schedule of minimum length. We find a state which has activity a_N in progress (or $K_Y = a_N$), it must be a state that will provide an optimal solution. Also, it is important to note that the level of the search tree can never exceed the sum of durations of all the activities.

In this simple form the algorithm is very inefficient as too many states get generated. Pruning rules must be employed to cut down the effective branching factor of the search tree. Two pruning rules are used to prune the tree. These are Dominance Pruning rule and One Child Rule and are described below.

Algorithm Preempt_BFS

```
Begin
    create the root state I and insert it in priority queue;                    /* initialize */
    do                                                                          /* loop */
        get current best state X form the priority queue;                       /* select state for expansion */
        if state X is a solution state then output schedule and makespan, and exit;   /*terminate */
        else
            determine the decision point $dp_{X;}$
            construct $K_X$ and all the MRSs;
            if the One-Child Rule applies then generate one child state of X
                corresponding to the singular MRS/* expand with one-child rule /* expand -one-child rule */
            else
                for each MRS
                    construct the corresponding child state;
                    apply the Dominance Pruning Rule;
                    if the child state has not been pruned
                        determine its heuristic estimate;
                        insert it in the search tree as a child of state X;
                        insert it in the priority queue at the appropriate place;
                    end if;
                end for;
            end if-else;
        end if-else;
    while true;                                                                 /* end do */
end algorithm.
```

Two activities a_i and a_j are *compatible* if a_i and a_j can be processed concurrently not taking other activities into account. Two activities cannot be compatible if one is a successor of the other in the precedence relationship. When not so related, their compatibility is determined by their resource requirements and by the total availability of resources.

An MRS A_Y at a decision point dp_X of a state X is *singular* if

i) no activity belonging to A_Y is in progress in X prior to dp_X; *and*

ii) there is an activity a_i in A_Y such that a_i is the longest activity in A_Y, *and* among activities that remain unfinished at time dp_X, those that are compatible with a_i are all included in A_Y.

Activity a_i will be referred to as a *distinguished member* of the singular MRS A_Y.

One-Child Rule: If all activities in progress in a state X complete at the decision point dp_X, *and* if there exists a singular MRS A_Y at that time, then generate only one child state of X, namely the state Y that corresponds to MRS A_Y. If more than one MRS is singular at dp_X, then choose any one arbitrarily. For this child state Y $dp_Y = c_X$ + duration of a_i.

While the One-Child Rule helps to reduce the number of states, the use of a Dominance Pruning Rule makes the pruning much more effective. This rule prunes states that would generate solutions no better than those obtainable from states that remain in the search tree.

Many alternative formulations of the rule are possible, some weaker than others in pruning power. A balance must be struck between effective pruning and implementation overhead. The Dominance Pruning Rule used in Preempt_BFS has the following form:

The Dominance Pruning Rule used in Preempt_BFS has the following form:

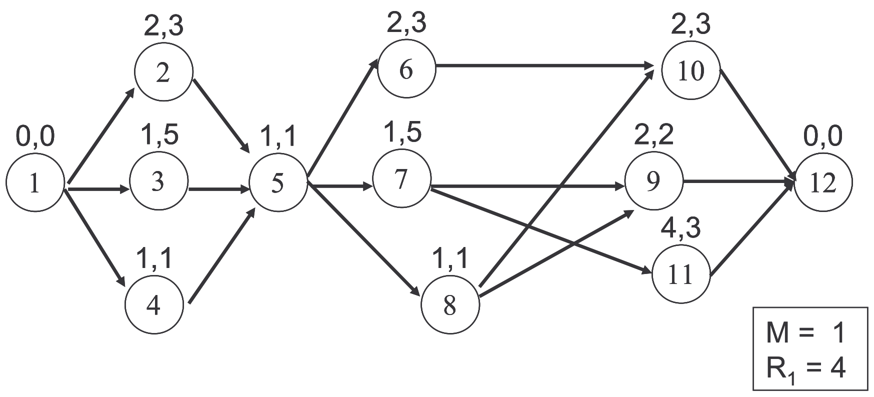

Figure 1: Project for Example 1

Dominance Pruning Rule: If at any time during the execution of Preempt_BFS there are two states X and Y in the search tree such that:

i) $dp_X = dp_{Y;}$

ii) Set of Candidate Activities for state X = Set of Candidate Activities for state Y;

iii) The balance (remaining) time in state X of each activity in set of Candidate activities is less than or equal to the balance time of corresponding activities in state Y at time dp_Y

then prune state Y from the search tree.

Let us define level of a state by its dp_X. Thus a state X can dominate a state Y only if X and Y are at the same level. The algorithm with its pruning rule is shown in the box. Example 1 shows implementation of algorithm with One-Child and Dominance Pruning rule.

Example 1: Consider the project network shown in Figure 1. It has twelve activities numbered 1 to 12. The duration (p_i) and resource requirement (r_i) of each activity is given on the top of the activity. There is one resource type with maximum availability of six. The pre-emptive makespan of the project is 11 as shown in Figure 2(a). Please note that if preemptions were not allowed the makespan would be 12 as shown in Figure 2(b). Table 1 shows the details of the states generated in the search tree.

5 Preempt_BDS: A Breadth-First Implementation

Along with the best-first implementation, a breadth-first version of the algorithm was also implemented. The breadth-first version is different from best-first as the nodes in breadth-first are expanded level-by-level, where a level is defined as the decision point of that particular partial state. Since the expansion of states is done level-by-level, it is enough to keep at most two levels in memory, rather than keeping all the states. This reduces the need for memory significantly and many instances which cannot be solved by best-first may be solved by using breadth-first approach, especially under memory constraints.

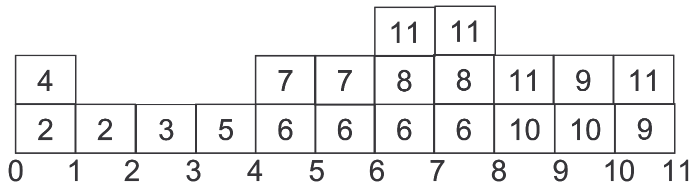

Figure 2(a) : Schedule for Example 1
(Preemption Allowed)

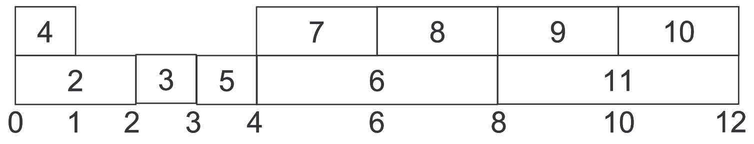

Figure 2(b) : Schedule for Example 1
(Preemption Not Allowed)

Table 1: States Generated in Search Tree for Example 2

State	Parent State	Decision Point dpX	Completed FX	In Progress AX	MRS for Child States	Kandidates Activities for Current State	Balance Times of KX	LB	Remarks
S1	-	0	{}	(1}	{2,4},{3,4}	{1}	{0}	9	One child Rule
S2	S1	1	{1}	{2,4}	{2},{3}	{2,3,4}	{1,1,0}	9	
S3	S2	2	{1,4}	{2}	{3}	{2,3}	{0,1}	10	
S4	S2	2	{1,4}	{3}	{2}	{2,3}	{1,0}	10	
S5	S3	3	{1,2,4}	{3}	{5}	{3}	{0}	10	
S6	S5	4	{1,2,3,4}	{5}	{6,7},{6,8}	{5}	{0}	10	
S7	S6	5	{1,2,3,4,5}	{6,7}	{6,7},{6,8}	{6,7,8}	{3,1,2}	10	
S8	S6	5	{1,2,3,4,5}	{6,8}	*	{6,7,8}	{3,2,1}	11	
S9	S7	6	{1,2,3,4,5}	{6,7}	{6,7}	{6,7,8}	{2,0,2}	10	
S10	S7	6	{1,2,3,4,5}	{6,8}	*	{6,7,8}	{2,1,1}	11	
S11	S9	7	{1,2,3,4,5,7}	{6,8}	{6,8}	{6,8,11}	{1,1,3}	10	
S12	S11	8	{1,2,3,4,5,7	{6,8}	{9,10},{9,11},{10,11}	{6,8,11}	{0,0,2}	10	
S13	S12	9	{1,2,3,4,5,7,8}	{9,10}	{9,10},{9,11},{10,11}	{9,10,11}	{1,1,2}	11	
S14	S12	9	{1,2,3,4,5,7,8}	{9,11}	{9,10},{9,11},{10,11}	{9,10,,11}	{1,2,1}	11	
S15	S12	9	{1,2,3,4,5,7,8}	{10,11}	*	{9,10,,11}	{2,1,1}	11	
S16	S4	3	{1,3,4}	{2}	{5}	{2}	{0}	10	
S17	S16	4	{1,2,3,4}	{5}	{9,10},{9,11},{10,11}	{5}	{0}	10	Dominated by S6
S18	S15	10	{1,2,3,4,5,7,8}	{9,10}	{9,11}	{9,10,11}	{1,0,1}	11	
S19	S15	10	{1,2,3,4,5,7,8}	{9,11}	*	{9,10,11}	{1,1,0}	11	
S20	S15	10	{1,2,3,4,5,7,8}	{10,11}	*	{9,10,11}	{2,0,0}	12	
S21	S14	10	{1,2,3,4,5,7,8}	{9,10}	*	{9,10,11}	{0,1,1}	11	
S22	S14	10	{1,2,3,4,5,7,8}	{9,11}	*	{9,10,11}	{0,2,0}	12	
S23	S14	10	{1,2,3,4,5,7,8}	{10,11}	*	{9,10,11}	{1,1,0}	11	Dominated by S19
S24	S13	10	{1,2,3,4,5,7,8}	{9,10}	*	{9,10,11}	{0,0,2}	12	
S25	S13	10	{1,2,3,4,5,7,8}	{9,11}	*	{9,10,11}	{0,1,1}	11	Dominated by S21
S26	S13	10	{1,2,3,4,5,7,8}	{10,11}	*	{9,10,11}	{1,0,1}	11	Dominated by S18
S27	S18	11	{1,2,3,4,5,7,8,10}	{9,11}	{12}	{9,11}	{0,0}	11	
S28	S27	11	{1,2,3,4,5,7,8,9,10,11}	-	-	{12}	{0}	11	Final Solution

* Not considered for expansion

6. Experimental Observations

In this section we refer to the three algorithms of interest as follows. DH: Algorithm of Demeulemeester and Herroelen, 1997; PBR: Algorithm Preempt_BDR (with all pruning rules); PBS Algorithm Preempt_BDS (with all pruning rules). The experiments were conducted on the problem set of Kolisch *et al*. containing 480 problems. The algorithm was coded in C on Linux and was run on a Pentium IV 1.7 GHz machine with 490 MB RAM. The results are as follows.

i. As shown in Table 2, PBS solves 411 problems in all followed by PBR (376 problems solved) and then DH (337 problems solved).

ii. DH takes least time to solve the problems. However, this is because it solves only easier problems. DH takes 1.26 seconds[2] to solve 337 problems while PBR takes 14.49 seconds to solve 376 problems. PBS solves 411 problems in 10.29 seconds (Table 2).

Table 2: Overall Results on Problem Set of Kolisch *et. al.*

	DH	PBR	PBS
Problems Solved	337	376	411
Average Time	1.26	14.49	10.29
States Generated	46,991	221,046	154,930
States Expanded	4,954	43,015	23,757

iii. DH on average generates 46,991 states for the problems solved. Average number of states generated is highest for PBR with 221,046 states while PBS generates 154,930 states (on average) only. Out of the 46,991 states generated DH expands 4,954 states which are 10.54 % of the states generated. Similarly, PBR expands 43,015 states resulting in expansion ration of 19.46%. PBR has an expansion ration of 15.33% (Table 2).

These 480 problems are categorized on the basis of three parameters *viz*. Network Complexity (NC), Resource Factor (RF) and Resource Strength (RS). NC measures the complexity level of the network as determined from its topology; it depends on the number of arcs and edges in the network. The lower the value of NC, the more activities can be done in parallel, allowing more activities to compete for same resource. RF reflects the average resource requirement of a job. RF = 1 means that each job requests all resources; RF = 0 indicates that no job requests any resource. RS shows the relationship between resource requirement and resource availability.

$$RS = \frac{\text{Resource Availability - Minimum Resource Requirement}}{\text{Maximum Resource Requirement - Minimum Resource Requirement}}$$

The value of RS lies between zero and one. When resource availability just equals the minimum resource requirement, resource constraints are very tight and RS = 0. When resource availability equals the maximum resource requirement, RS = 1. The values of minimum resource requirement and maximum resource requirement are calculated by solving the project scheduling problem ignoring resource constraints.

iv. More problems are solved with increase in NC by all the three algorithms (Table 3). PBR solves one problem less than DH when NC is equal to 1.5 (PBR = 107, DH = 108), however for NC values of 1.8 and 2.1 PBR solves much more problems compared to DH (DH: from 107 solved for NC = 1.8 to 122 solved for NC = 2.1, PBR: from 124 solved for NC = 1.8 to 145 solved for NC = 2.1).

[2] All timings, number of states generated as well as expanded are averages.

v. For NC = 1.5 and NC = 2.1 PBS takes less time compared to PBR. However for NC = 1.8 PBR takes less time than PBS (Table 3).

vi. Time taken for by DH and PBR decreases with increase in NC. For PBS time taken reduces from 0.13 second of NC = 1.5 to 0.05 second for NC = 1.8 but then increases to 2.87 for NC = 2.1(Table 3).

vii. If the ratio of number of states generated to number of states expanded is considered, at NC = 1.5 DH has the most favourable ratio of 5.9% followed by PBS at 13.8% and then by PBS at 17.3%. At NC = 2.1 also the ratio remains in favour of DH. However at NC = 1.8, the ratio is favourable to PBS at 13.8% (Table 4).

Table 3: Impact of NC in Problems by Kolisch *et al.*

	No. of Problems Solved			
	1.5	1.8	2.1	Total
DH	108	107	122	337
PBR	107	124	145	376
PBS	126	135	150	411
Total	**160**	**160**	**160**	**480**
	Average Time Taken			
	1.5	1.8	2.1	Total
DH	0.78	0.04	2.75	1.26
PBR	18.27	10.53	15.10	14.49
PBS	13.32	12.17	6.05	10.29
	No. of States Generated			
	1.5	1.8	2.1	Total
DH	62,052	3,488	71,814	46,991
PBR	264,824	201,759	205,235	221,046
PBS	170,888	181,730	117,406	154,930
	No. of States Expanded			
	1.5	1.8	2.1	Total
DH	3,689	654	9,846	4,954
PBR	45,850	40,176	43,351	43,015
PBS	23,506	25,102	22,757	23,757

Table 4: Ratio of States Expanded to States Generated for change in NC

	1.5	1.8	2.1
DH	5.9%	18.8%	13.7%
PBR	17.3%	19.9%	21.1%
PBS	13.8%	13.8%	19.4%

viii. Number of problems solved decrease for all the methods with increase in RF. While for RF =0.25 DH could solve only 116 problems PBR and PBS solved all 120 problems. For all values of RF PBS always solved maximum instances, followed by PBR and DH (Table 5).

ix. With increase in RF, time taken to solve the problems always increase for PBR. For PBS and DH time taken increase in general but for RF = 0.75 time taken is less than time taken for RF = 0.5. For DH this reduction from RF = 0.5 to RF= 0.75 is significant (Table 5).

x. If the ratio of number of states generated to number of states expanded is considered, at RF = 0.25 PBS has the most favourable ratio of 17.55% followed by PBR at 19.53% and then by DH at 23.16%. At RF = 1.0 the ratio is in favour of DH at 7% followed by PBS (11.32%) and PBR (17.05%) (Table 6).

Table 5: Impact of RF in Problems by Kolisch *et al.*

	No. of Problems Solved				
	0.25	0.5	0.75	1	Total
DH	116	82	76	63	337
PBR	120	108	83	65	376
PBS	120	112	97	82	411
	Average Time				
	0.25	0.5	0.75	1	Total
DH	0.07	0.57	0.05	5.80	1.26
PBR	1.61	12.65	24.25	28.89	14.49
PBS	0.80	8.99	7.57	29.19	10.29
	No. of States Generated				
	0.25	0.5	0.75	1	Total
DH	3,947	27,266	5,505	201,969	46,991
PBR	34,708	277,486	304,794	364,338	221,046
PBS	16,498	187,885	146,719	322,216	154,930
	No. of States Expanded				
	0.25	0.5	0.75	1	Total
DH	914	7,382	894	14,131	4,954
PBR	6,745	58,037	60,953	62,111	43,015
PBS	2,895	35,549	25,191	36,485	23,757

Table 6: Ratio of States Expanded to States Generated for change in RS

DH	23.16%	27.07%	16.24%	7.00%
PBR	19.43%	20.92%	20.00%	17.05%
PBS	17.55%	18.92%	17.17%	11.32%

Table 7: Impact of RS in Problems by Kolisch *et al.*

	No. of Problems Solved				
	0.2	0.5	0.7	1	Total
DH	29	78	110	120	337
PBR	71	73	112	120	376
PBS	68	105	118	120	411
	Average Time Taken				
	0.2	0.5	0.7	1	Total
DH	1.45	1.05	2.73	0.00	1.26
PBR	36.33	12.82	17.27	0.00	14.49
PBS	22.74	23.39	1.93	0.00	10.29
	No. of States Generated				
	0.2	0.5	0.7	1	Total
DH	65,660	89,297	63,277	52	46,991
PBR	644,251	216,420	192,589	25	221,046
PBS	493,157	249,403	33,462	49	154,930
	No. of States Expanded				
	0.2	0.5	0.7	1	Total
DH	19,467	5,843	5,846	51	4,954
PBR	139,721	39,861	29,828	25	43,015
PBS	99,169	24,110	4,095	49	23,757

xi. With increase in RS, number of problems solved increase sharply. For RS = 1, all the three methods solve all 120 problems. Fro RS = 0.2, PBR solves maximum number of problems followed by PBS and DH (PBR = 71, PBS = 68, DH = 29) (Table 7).

xii. For RS = 0.2, number of states generated are maximum for PBR followed by PBS and DH. However the states generated reduce sharply as RS increases. For RS = 1, PBR has least number of states generated followed by PBS and DH (Table 7). Number of states expanded also reduces with increase in RS. This reduction is significant for PBR followed by PBS and DH.

xiii. Table 8 shows that if the ratio of number of states generated to number of states expanded is considered, at RS = 0.2 PBS has the most favourable ratio of 20.1% followed by PBR at 21.7% and then by DH at 29.6%. At RS = 1.0 both PBR and PBS expand all the states generated while DH expand 98.1% of the states. However, it is at lower values of RS, it is beneficial to have lesser number of states generated as well as expanded.

Table 8: Ratio of States Expanded to States Generated for change in RS

DH	29.6%	6.5%	9.2%	98.1%	10.5%
PBR	21.7%	18.4%	15.5%	100.0%	19.5%
PBS	20.1%	9.7%	12.2%	100.0%	15.3%

The analysis so far considers all the solved problems. However, it is quite obvious that PBR and PBS may seem to perform poor in terms of average time, states generated and states expanded because they solve more and therefore difficult instances. Therefore, now we analyse common instances solved by all three problems.

xiv. In all 315 problems were commonly solved by all the three algorithms (Table 9).

Table 9: Common Problems Solved by All Three Methods

NC	1.5	1.8	2.1	Total
Problems Solved	95	103	117	315

xv. 95 of these 315 common problems were for NC = 1.5, 103 were for NC = 1.8 and 117 were for NC = 2.1 (Table 9).

xvi. As shown in Table 10, out of these 315 problems, PBS (0.06 second) takes least time followed by DH (1.12 seconds) and then PBR (6.49 seconds).

Table 10: Average Time for Common Problems

	NC				
	1.5	1.8	2.1		Total
DH	0.13	0.05	2.87		1.12
PBR	9.71	6.38	3.97		6.49
PBS	0.10	0.05	0.05		0.06
	RF				
	0.25	0.5	0.75	1	Total
DH	0.07	0.56	0.04	5.51	1.12
PBR	0.47	1.10	15.80	15.91	6.49
PBS	0.05	0.10	0.06	0.04	0.06
	RS				
	0.2	0.5	0.7	1	Total
DH	1.50	0.19	2.85	0.00	1.12
PBR	1.82	6.36	15.22	0.00	6.49
PBS	0.19	0.13	0.07	0.00	0.06

xvii. PBS gives best time performance for all values of NC followed by DH and PBR.

xviii. With increase in RF, time taken may seem to behave erratically. This behaviour is due to the fact that the difficulty of instances increase slightly problems get solved in more time, but if the difficulty increase beyond a certain level lesser problems get solved, even though it may be observed that time taken has reduced.

xix. With increase in RS time taken for PBS reduces. Time take by PBR first increases with increase in RS up to 0.7 and then reduces to zero. However, it should be noted that whenever the average time increases, number of problem solved also increases implying that more difficult problems are being solved (Table 10).

xx. Table 11 shows that out of 416 problems solved for 278 problems there is no reduction in makespan when compared against non-preemptive case. For 56 problems makespan reduced by 1 unit time. Similarly reduction was 2, 3, 4, 5 and 6 for 52, 21, 7, 1 and 1 problem respectively.

Table 11: Decrease in the optimal project length if preemption is allowed Problems by Kolisch *et al*.

Decrease in Makespan	0	1	2	3	4	5	6	Total
Count of Problems	278	56	52	21	7	1	1	416

7. Conclusion

The algorithm Preempt_BFS is a simple algorithm based on best-first search. The pruning rule is also simple and therefore, easy to implement. However, newer pruning rules may be devised to improve the performance of the algorithm. The algorithm is compared to Demeulemeester and Herroelen [1996] and it can be shown that it gives better results.

8. References

Bell C. E. & Park K. (1990): "Solving resource-constrained project scheduling problems by A* search", *Naval Research Logistics, Vol 37, February 1990, pp 61-84.*

Christofides N., Alvarez-Valdes R. & Tamarit J. M. (1987): "Project scheduling with resource constraints: A branch and bound approach", *European Journal of Operational Research,* Vol 29, 1987, pp 262-273.

Davis E. W., and Heidorn G.E. (1971): "An algorithm for optimal project scheduling under multiple resource constraints", *Management Science*, Vol 17, No 12, August 1971, pp 803-816.

Demeulemeester E. & Herroelen W. (1992): "A branch-and-bound procedure for the multiple resource constrained project scheduling problem", *Management Science,* Vol 38, No 12, December 1992, pp 1803-1818.

Demeulemeester E. & Herroelen W. (1996): "A efficient optimal solution for the preemptive resource-constrained project scheduling problem", *European Journal of Operational Research,* Vol 90, Issue 2, April 1996, pp 334-348.

Demeulemeester E. & Herroelen W. (1997): "New benchmark results for the resource constrained project scheduling problem"*, Management Science,* Vol 43, No 11, November 1997, pp 1485-1492.

Kolisch R, Sprecher A, and Drexl A (1995), "Characterization and generation of a general class of resource-constrained project scheduling problems", *Management Science*, 41, 1693-1703.

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

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

Patterson J. H. (1984): "A comparison of exact approaches for solving the multiple constrained resource project scheduling problem", *Management Science,* Vol 30, No 7, 1984 pp 854-867.

Stinson J. P., Davis E. W. & Khumawala B. M. (1978): "Multiple resource-constrained scheduling using branch and bound", *AIIE Transactions,* Vol 10, No 3, September 1978, pp 252-259.

Verma S. (2004), "An Optimal Breadth-First Algorithm for the Preemptive Resource-Constrained Project Scheduling Problem", Joint 2nd World Conference and 15th Annual POMS Conference (POM2004), Cancún (Mexico), Apr. 30-May 3.