An exact solution procedure for mode identity and resource constrained project scheduling problem

BEHROUZ AFSHAR NADJAFI1, AMIR RAHIMI2
Department of Industrial Engineering
Qazvin Islamic Azad University
Islamic Azad University, Nokhbegan avenue, Qazvin, Iran (P.O.Box: 34185-1416)
IRAN
1afsharnb@alum.sharif.edu, 2 a.rahimi@qiau.ac.ir

Abstract: Mode identity problem refers to a generalization of the multi-mode resource constrained project scheduling Problem (MRCSP). In this paper, we present a depth-first branch and bound algorithm for resource constrained project scheduling problem with mode identity, in which set of project activities is partitioned into disjoint subsets while all activities forming one subset have to be processed in the same mode. The proposed algorithm is extended with some bounding rules to reduce the size of branch and bound tree. Finally, some test problems are solved and computational results are reported.

Key-Words: Project Scheduling, Multi-Mode, Mode-Identity, Branch and Bound

1 Introduction
The standard multi-mode resource constrained project scheduling problem involves the selection of an execution mode for each activity (mode assignment) and the determination of the activity start or finish times such that the precedence and resource constraints are met and the project duration is minimized. In the multi-mode case, all mode-activity assignments are mutually independent; i.e. assigning a mode to one activity \( i \) of a project consisting of a set of \( n \) nonpreemptable activities does not necessarily force any other activity to be processed in a specific mode. In practice, however, situations may occur in which certain activities belong together and must be executed in the same mode [1].

Salewski et al. partitioned the set of all activities into disjoint subsets where all the activities forming one subset have to be performed by the same resources [2]. Time and cost incurred by processing such a subset depend on the resources assigned to it. Salewski et al. refer to the resulting problem as the mode-identity problem, in which objective is to minimize the cost of processing [2]. They prove that the mode identity problem is strongly NP-hard [2].

The literature on solution methods for the mode identity problem is scant. Salewski et al. have developed a parallel regret-based biased random sampling approach, RAMSES, which consists of two stages. In the first stage, priority values are used to assign modes to subsets of activities. In the second stage, a schedule is built using a priority-based parallel scheduling scheme [2]. This paper addresses the resource constrained project scheduling problem with mode identity, in which we consider a project consisting of activities to be scheduled subject to finish-start precedence relations with zero time lags and renewable resource constraints. Objective is to minimize the project duration.

The paper is organized as follows: first, the problem description is presented and the terminology used is clarified. Then, a branch and bound procedure is described. Following that, computational results are reported. Finally, the paper is concluded.

2 Problem Description
The mode identity and resource constrained project scheduling problem (MIRCPSP) involves the scheduling of project activities in order to minimize the project makespan. In this problem setting, the set of project activities is partitioned into \( U \) disjoint subsets while all activities forming one subset have to be processed in the same mode. The project is represented by an AON network where the set of nodes, \( N \), represents activities and the set of arcs, \( A \), represents finish-start precedence constraints with a time-lag of zero. The nonpreemptable activities are numbered from a dummy start activity 1 to the dummy end activity \( n \), and are topologically ordered. We have the notations given in table 1 for MIRCPSP.
Defining variables $x_{jmt}$ as follow:

$$x_{jmt} = \begin{cases} 
0; & \text{if activity } j \text{ is performed in mode } m \text{ and completed in period } t \\
1; & \text{otherwise}
\end{cases}$$

(1)

allows to formulate the mode identity and resource constrained project scheduling problem (MIRCPSP) under the minimum project makespan objective as given in Table 2 (derived from Salewski et al. formulation [2]). The objective in equation 2 minimizes the project duration. It is assumed that the dummy start node and dummy end node can only be processed in a single mode with duration equal to zero. The constraints in equation 3 assure that each activity is assigned exactly one mode and exactly one finish time. The constraint set in equation 4 maintains the mode identity constraints, in which all activities forming one subset have to be performed in the same mode.

$$\text{Min } Z = \sum_{t \in EFT_j} t \times n_{j1}$$

(2)

$$\sum_{m=1}^{M_u} \sum_{t \in EFT_j} x_{jmt} = 1 \quad (1 \leq u \leq U)$$

(3)

$$\sum_{m=1}^{M_u} \sum_{t \in EFT_j} x_{jmt} = \sum_{m=1}^{M_u} \sum_{t \in EFT_j} x_{jmt} \quad (1 \leq u \leq U, \forall \in H_u \setminus \{f_u\}, 1 \leq m \leq M_u, |H_u| > 1)$$

(4)

$$\sum_{m=1}^{M_u} \sum_{t \in EFT_j} t x_{jmt} \leq \sum_{m=2}^{M_u} \sum_{t \in EFT_j} (t - d_{jmt}) x_{jmt} \quad (1 \leq u \leq U, \forall \in H_u, 1 \leq m \leq U, \forall \in EPT_j)$$

(5)

$$\sum_{u=1}^{U} \sum_{m=1}^{M_u} \sum_{t \in EFT_j} t R_k x_{jmt} \leq R_k \quad (1 \leq k \leq K, 1 \leq t \leq T)$$

(6)

$$x_{jmt} \in \{0,1\} \quad (1 \leq u \leq U, \forall \in H_u, 1 \leq m \leq M_u, \forall \in EFT_j, \forall \in LFT_j)$$

(7)

Equation 5 denotes the precedence relations-constraints. Equation 6 secure that the per-period availability of the renewable resources is not violated. Finally, equation 7 imposes binary values on the decision variables.

### 3 B&B Algorithm (Precedence Tree)

The precedence tree based approach was originated by Patterson et al. [3] to solve RCPSP and was further refined by Sprecher [4] and Sprecher and Drexl [5] to solve multi-mode case. Hartman and Drexl [6] show that the precedence tree approach by Sprecher and Drexl [5] outperformed the other available branch and bound algorithms with respect to computation times. The precedence tree approach is based on the enumeration of all feasible sequences that correspond with different early-start schedules and the selection of the best amongst these feasible sequences.

**Table 1. Parameters for MIRCPSP**

<table>
<thead>
<tr>
<th>Problem parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>Number of activities indexed by $j$</td>
</tr>
<tr>
<td>$K$</td>
<td>Number of renewable resources indexed by $k$</td>
</tr>
<tr>
<td>$d_{jm}$</td>
<td>Time required to perform activity $j$ in mode $m$</td>
</tr>
<tr>
<td>$H_u$</td>
<td>Specific nonempty subset $u$ of activities</td>
</tr>
<tr>
<td>$U$</td>
<td>Number of disjoint subsets of activities, indexed by $u$</td>
</tr>
<tr>
<td>$T$</td>
<td>Number of time periods, indexed by $t$</td>
</tr>
<tr>
<td>$EFT_j$</td>
<td>Earliest finish time of activity $j$</td>
</tr>
<tr>
<td>$LFT_j$</td>
<td>Latest finish time of activity $j$</td>
</tr>
<tr>
<td>$f_u$</td>
<td>The job with the smallest index of subset $H_u$</td>
</tr>
<tr>
<td>$P_j$</td>
<td>The set of immediate predecessors of activity $j$</td>
</tr>
</tbody>
</table>

**Table 2. Mathematical model for the problem**
In this section, we propose a modified structure of the precedence tree algorithm for MIRCPSP. The procedure is based on the observation that any early-start schedule can be obtained by listing all activities in a sequence such that no successor of an activity is sequenced before its predecessor. Every such sequence corresponds with one early-start schedule by scheduling the different activities as soon as possible in the order of the sequence, but without violating the precedence, resource and mode identity constraints.

3.1 Branching Strategy
The procedure begins with starting the dummy start activity 1 with mode 1 at time 0. At each level \( g \) of the branch and bound tree, we determine the set \( SJ_g \) of the already scheduled activities and the set \( EJ_g \) of the eligible activities, that is, those activities the predecessors of which are already scheduled. Then we select an eligible activity \( j \). If activity \( j \) has at least one same subset activity scheduled at previous levels, its execution mode is fixed before, so activity \( j \) has only one allowable mode \( m_g \). Else, we select a mode \( m_j \in \{1, \ldots, M_j\} \) of this activity. Now we compute the earliest precedence feasible start time \( EST_{jg} \) and the earliest resource feasible start time \( EST_{jg} \) so that \( sj \geq EST_{jg} \). Then we branch to the next level. If the dummy end activity \( n \) is eligible, we have found a complete schedule, and finish time of activity \( n, FT_n \), is project duration. In this case, backtracking to the previous level occurs. Here we select the next untested mode. If none exists, we select the next untested eligible activity. Note that if the selected activity \( j \), has at least one same subset activity scheduled at previous levels, it has no untested mode. If we have tested all eligible activities in all allowable modes, we track another step back.

Having discussed all the necessary concepts of the algorithm, we present it with the pseudo-code given in Table 3.

3.2 Bounding Rules
If it can be established that further branching from a node cannot lead to an optimal solution, then the node can be pruned away. While most of the rules are known from the literature, we also present two new rules and adapt some well-known ones for MIRCPSP.

3.2.1 Bounding Rule 1 (Data Reduction)
This bounding rule has originally been proposed by Sprecher et al. [7]. An execution mode \( m_i \) is called non-executable if we have \( \tau_{jm_i k} > \bar{r}_k \) for all \( k \in K \). A mode \( m_{jg} \) is called inefficient if \( d_{jm} \geq d_{jm_{g}}, \) and \( r_{jm,k} \geq r_{jm_{g}k} \) for all \( k \in K \). Here, \( m_{jg} \) is all another modes of the same activity. Hence, non-executable and inefficient modes may be excluded from the project data without losing optimality.

3.2.2 Bounding Rule 2
This bounding rule is based on critical path length. Patterson et al. [3] have employed primal version of this rule in their algorithm for solving the RCPSP. We propose an alternative version for MIRCPSP. Consider an upper bound on the makespan of the project which is e.g. given by the sum of the maximal duration of the activities. If algorithm has found the first or an improved schedule with a makespan \( T \), the upper bound replaced by \( T \). We add the remaining critical path length of the currently scheduled activity to its start time and if this value exceeds or equals the currently best solution, we can dominate the current node in the precedence tree. Note that, computation of critical path length some differs. In computation of critical path length of the currently scheduled activity \( j \), if activity \( j \) has at least one same subset activity at current partial schedule, its fixed mode duration needs to be considered. Else, its minimal duration needs to be considered.

3.2.3 Bounding Rule 3
We drive another critical path based bounding rule for MIRCPSP. This bounding rule has originally been proposed by Sprecher for MRCPS [4]. If an eligible activity cannot be feasibly scheduled in any mode in the current partial schedule without exceeding the upper bound, then no other eligible activity needs to be examined on this level. We add the remaining critical path length of the eligible activity to its start time and if this value exceeds or equals the currently best solution, then no other eligible activity needs to be examined at the current level. The critical remaining path length is computed as preceding bounding rule.

3.2.4 Bounding Rule 4 (Multi Mode Rule)
This bounding rule has originally been proposed by Sprecher et al. [7] for multi mode resource constrained project scheduling. Here, we transfer it for MIRCPSP.
Table 3. Branching Algorithm

Step 1: Initialization
Initialization step sets the level of the precedence tree to 1, \( g = 1 \).
Schedule the first (dummy) activity with the start time of zero, \( j_1 = 1 \), \( m_{j_1} = 1 \), \( s_{j_1} = 0 \).
Initialize the set of the already scheduled activities, \( S_{j_1} = \emptyset \).

Step 2: Compute the set of eligible activities
Increase the level of the precedence tree and update the set of already scheduled activities, \( g = g + 1 \);
\( S_{j_g} = S_{j_g-1} \cup \{ j_{g-1} \} \).
Compute the set of eligible activities (i.e., activities not currently scheduled whose predecessors are already scheduled), \( E_{j_g} = \{ e \in \{ 1, \ldots, n \} \setminus S_{j_g} \} \).
If the last (dummy) activity is eligible \( n \in E_{j_g} \), then store the current solution and go to Step 5.
Else go to Step 3.

Step 3: Select the next activity to be scheduled
If there is no untested activity left in \( E_{j_g} \), then go to Step 5.
Else select an untested activity, \( j_g \in E_{j_g} \).

Step 4: Select a mode and compute the activity start time
If there is no activity same subset \( j_g \) in \( S_{j_g} \), then
Else select an untested mode \( m_{j_g} \in \{ 1, \ldots, M_{j_g} \} \).
Set allowable mode \( m_{j_g} \) for other activities same subset \( j_g \).
Compute the earliest precedence feasible start time, \( EST_{j_g} = \max \{ FT_{i} ; e \in F \} \).
Compute the earliest resource feasible start time \( s_{j_g} = EST_{j_g} \), then go to Step 2.
Else if there is the activity same subset \( j_g \) in \( S_{j_g} \), then
If \( j_g \) once already scheduled at this level, then go to Step 3.
Else set allowable mode \( A_{M_j} \) for \( j_g \) : \( m_{j_g} = M_j \).
Compute the earliest precedence feasible start time, \( EST_{j_g} = \max \{ FT_{i} ; e \in F \} \).
Compute the earliest resource feasible start time \( s_{j_g} = EST_{j_g} \), then go to Step 2.

Step 5: Backtracking
Decrease the level of the precedence tree, \( g = g - 1 \).
If the precedence level is equal to 1, then STOP.
Else go to Step 4.

Within a given schedule, a multi mode left shift is a reduction of an activity finish time without changing the modes or finish times of the other activities, such that the resulting schedule is feasible. A schedule is called tight if no multi mode left shift can be performed. A mode reduction on an activity is a reduction of its mode number without changing its finish time and without changing the modes and finish times of the other activities. A schedule is called mode-minimal if there is no activity a mode reduction can be performed on. Note that there are tight schedules which are not mode-minimal and vice versa. Obviously, if there is an optimal schedule for a given instance, then there is an optimal schedule which is both tight and mode minimal. Assume that no currently unscheduled activity will be started before the finish time of a scheduled activity \( j \) when the current partial schedule is completed.
Note that activity \( j \) has no same subset activity at current partial schedule. If a multi mode left shift or mode reduction of activity \( j \) with resulting mode \( m'_{j} \), can be performed on the current partial schedule, then the current partial schedule needs not be completed.

3.2.5 Bounding Rule 5 (Order Swap Rule)
This bounding rule is proposed by Hartman and Drexel [6]. Denoting the finish time and start time of a scheduled activity \( j \) with \( f_{j} \) and \( s_{j} \), respectively, we consider two activity \( i \) and \( j \) with \( i \not\succ j \) that \( f_{i} = s_{j} \).

Now, an order swap is defined as the interchange of these two activities by assigning new start and finish times \( s'_{j} := s_{i} \) and \( f'_{i} := f_{j} \), respectively. Thereby, the precedence and resource constraints may not be violated, and the modes and starts times of the other activities may not be changed. A schedule in which no order swap can be performed is called order monotonous. Clearly, it is sufficient to enumerate only order monotonous schedules. Assume that no currently unscheduled activity will be started before the finish time of a scheduled activity \( j \) when the current partial schedule is completed. If an order swap on activity \( j \) together with any of those activities that finish at its start time can be performed, then the current partial schedule needs not be completed.

3.2.6 Bounding Rule 6
This bounding rule has been developed by Demeulemeester and Herroelen [8] for the RCPSP and generalized by Sprecher et al. [7] to the multi mode case. Here we transfer it for MIRCPSP.

Consider an eligible activity \( j \) no mode of which is simultaneously performable with any currently unscheduled activity in any mode. If the earliest feasible start time of each other eligible activity in any mode is equal to the maximal finish time of the currently scheduled activities, then \( j \) is the only eligible activity that needs to be selected for being scheduled on the current level of the branch and bound tree. Note that, if activity \( j \) has at least one same subset activity at current partial schedule, its only execution mode is fixed before. This notion is valid for any unscheduled activity too.

Also, if activity \( j \) has no same subset activity at current partial schedule, simultaneous execution of activity \( j \) with same subset activities, needs to be checked.

3.2.7 Bounding Rule 7
Due to the structure of the precedence tree, algorithm may enumerate one schedule several times. To avoid duplicate consideration of a schedule, Hartman and Drexel [6] have proposed a bounding rule to exclude duplicate enumeration for multi mode RCPSP, which we adapt it for MIRCPSP.

Consider two activities \( i \) and \( j \) scheduled on the previous and on the current level of the branch and bound tree, respectively. If we have \( s_{i} = s_{j} \) and \( i \succ j \), then the current partial schedule needs not to be completed.

3.2.8 Bounding Rule 8
This bounding rule has proposed by Patterson et al. [3] to avoid duplicate consideration of a schedule in RCPSP. Here, we adapt it for MIRCPSP.

Consider two activities \( i \) and \( j \) scheduled on the previous and on the current level of the branch and bound tree, respectively. If we have \( s_{j} < s_{i} \), then the current partial schedule needs not to be completed.

3.2.9 Bounding Rule 9 (The Cutset Rule)
This bounding rule stores information about already evaluated partial schedules. If it can proven that any solution obtained from the current partial schedule cannot be better than a solution obtainable from a previously evaluated partial schedule the information of which has been stored, then backtracking may be performed. Defining a cutset of a partial schedule \( PS \) as the set of activities scheduled in \( PS \), Sprecher and Drexel [5] proposed this rule for their algorithm for the MRCPSP. Here, we adapt this rule for MIRCPSP.

Let \( PS \) denote a previously evaluated partial schedule with cutset \( CS(PS) \) and maximal finish time \( f_{\text{max}}(PS) \). Let \( PS \) be the current partial schedule considered to be extended by scheduling some activity \( j \) with start time \( s_{j} \). If we have \( CS(PS) = CS(PS) \) and \( s_{j} \leq f_{\text{max}}(PS) \), then \( PS \) does not need to be completed.

Aware that, the use of the cutset rule simultaneously with the bounding rules 4, 5, 7 and 8 may cause the optimal solution to be missed. Thus, in use of the cutset rule, information about already evaluated partial schedules dominated with the bounding rules 4, 5, 7 and 8 needs not be stored.

4 Computational Results
In order to validate the proposed branch and bound method for the MIRCPSP, a problem set, consisting of 90 problem instances, was generated by the project generator ProGennx which has been developed by Drexel et al. [9], using the parameters given in table 4.
Table 4. The parameter settings for the problem set

<table>
<thead>
<tr>
<th>Control Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of activities</td>
<td>10, 12, 14</td>
</tr>
<tr>
<td>Number of execution modes</td>
<td>2, 3, 4</td>
</tr>
<tr>
<td>Job subset strength (JSS)</td>
<td>0.5, 0.6</td>
</tr>
<tr>
<td>Activity durations</td>
<td>[1,10]</td>
</tr>
<tr>
<td>Number of initial activities</td>
<td>[1,3]</td>
</tr>
<tr>
<td>Number of terminal activities</td>
<td>[1,2]</td>
</tr>
<tr>
<td>Maximal number of predecessors</td>
<td>3</td>
</tr>
<tr>
<td>Maximal number of successors</td>
<td>3</td>
</tr>
<tr>
<td>Coefficient of network complexity (CNC)</td>
<td>1.5</td>
</tr>
<tr>
<td>Resource factor (RF)</td>
<td>1</td>
</tr>
<tr>
<td>Resource strength (RS)</td>
<td>0.5</td>
</tr>
<tr>
<td>Number of resource types</td>
<td>2</td>
</tr>
<tr>
<td>Activity resource (per period) demand</td>
<td>[1,10]</td>
</tr>
</tbody>
</table>

The indication \([x,y]\) means that the value is randomly generated in the interval \([x,y]\). Resource availability is assumed to be constant over time. For each combination of parameters (number of activities, number of execution modes and job subset strength), 5 problem instances was generated, for which, have no non-executable or inefficient mode.

The resource factor RF reflects the average portion of resource required per activity. The resource strength RS reflects the scarceness of the resource. The job subset strength JSS reflects the average number of activity subsets which have to be processed in the same mode.

From table 5 it can be observed that all 90 problems can be solved to optimality within the allowed time limit. It is apparent that the average computation time as well as the standard deviation of the computation time are increasing function of the number of activities and number of execution modes. Table 5 also reveals that an increase in the job subset strength JSS from 0.5 to 0.6 leads to a decrease in the problem complexity, measured by the average as well as by the standard deviation CPU-time.

5 Summary and Conclusions

This paper reports on an exact branch and bound procedure for mode identity and resource constrained project scheduling problem, in which set of activities is partitioned into disjoint subsets while all activities forming one subset have to be processed in the same mode. The objective is to schedule the activities, in order to minimize the project duration. Depth first branching is based on precedence tree approach. Nine rules are used for node fathoming; including seven bounding rules adapted from the literature, and two new rules. Finally, the new branch and bound procedure is used for solving some test problems and computational results demonstrate that the proposed branch and bound is in fact capable of solving problems in acceptable time.

Table 5. Computational results on the MIRCPSP

<table>
<thead>
<tr>
<th>Job subset strength (JSS)</th>
<th>JSS=0.5</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of execution modes</td>
<td>Number of activities</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>n=10</td>
<td>μ</td>
<td>σ</td>
<td>μ</td>
<td>σ</td>
<td>μ</td>
<td>σ</td>
<td>μ</td>
<td>σ</td>
<td>μ</td>
<td>σ</td>
<td>μ</td>
<td>σ</td>
<td>μ</td>
</tr>
<tr>
<td></td>
<td>n=12</td>
<td>1.15</td>
<td>0.55</td>
<td>2.03</td>
<td>1.43</td>
<td>3.95</td>
<td>1.98</td>
<td>0.46</td>
<td>0.28</td>
<td>1.42</td>
<td>0.92</td>
<td>1.93</td>
<td>1.53</td>
<td>1.15</td>
</tr>
<tr>
<td></td>
<td>n=14</td>
<td>1.78</td>
<td>1.73</td>
<td>6.56</td>
<td>1.96</td>
<td>13.90</td>
<td>2.16</td>
<td>1.13</td>
<td>0.34</td>
<td>3.28</td>
<td>1.22</td>
<td>7.28</td>
<td>1.92</td>
<td>1.78</td>
</tr>
</tbody>
</table>

We have coded the branch and bound procedure in the MATLAB version 7.4. The problem set has been solved under windows XP professional on a personal computer with an Intel Core2Dou, 2.5GHz processor and 3GB memory. Table 5 represents the average and the standard deviation CPU-time, in seconds, for a different number of activities, number of execution modes and JSS. (The limit on the computational time value was set to 1 minute).
References:


