Propagating Resource Constraints Using Mutual Exclusion Reasoning

Romeo Sanchez[1], Minh B. Do[1], and Jeremy Frank[2]

[1] Arizona State University , Tempe AZ 85287-5406, USA,
rsanchez,binhminh@asu.edu
[2] NASA Ames Research Center, Mail Stop N269-3, Moffet Field, CA 94035-1000, USA,
frank@ptolemy.arc.nasa.gov

Abstract. One of the most recent techniques for propagating resource constraints in Constraint Based scheduling is Energy Constraint. This technique focuses in precedence based scheduling, where precedence relations are taken into account rather than the absolute position of activities. Although, this particular technique proved to be efficient on discrete unary resources, it provides only loose bounds for jobs using discrete multi-capacity resources. In this paper we show how mutual exclusion reasoning can be used to propagate time bounds for activities using discrete resources. We show that our technique based on critical path analysis and mutex reasoning is just as effective on unary resources, and also shows that it is more effective on multi-capacity resources, through both examples and empirical study.

1 Introduction

Scheduling problems are fundamentally concerned with the interaction of temporal constraints and resource constraints. Tasks that require the same resources lead to a choice of precedence, but some choices may be impossible due to the absolute and relative constraints on the timing of activities. A number of techniques have been developed to propagate resource and temporal constraints. Laborie [1] provides a good survey of these results, including Timetabling [2], Edge Finding [3, 4], and Energetic Reasoning [8]. Based on the survey, Laborie developed a technique called the Energy Precedence Constraint (*EC*). This constraint works on resources that are used then released (called discrete resources by Laborie). The idea is to estimate the time required to execute all activities on a single resource by computing the "energy", that is the sum of the duration of the activities times the amount of the resource they require, then computing the minimum duration by dividing the energy by the amount of the resource available.

The *EC* is simple to compute, but results in loose computational bounds in cases where the total capacity of a resource is exceeded by the resource consumption of the activities. We observe that a simple analysis of the activities can lead to the discovery of mutual exclusions among group of activities. This can result in improved bounds on such problems with small cost.

The rest of the paper is organized as follows. Section 2 introduces the preliminaries of our work, and a short description of the energy bound *EB* propagated by the energy constraint procedure. Section 3 will describe the idea of our constraint propagation technique called *CPMB* (critical path plus mutex bounds), and one extension for multiple machine multiple capacity problems, called *GMB* (group mutex bounds). Then, Section 4 presents an example and the possible best and worst case scenarios for the propagation techniques described in this paper. Section 5 will introduce some aspects of our implementation which affect the way resource constraints get propagated. Section 6 will

provide empirical evaluation of our techniques, showing that *CPMB* propagates more constraints than *EB* in presence of mutex information, reducing the execution time and number of nodes generated during search. Finally, Section 7 presents the conclusions and some ideas for future work.

2 Preliminaries

We consider a set of activities A. Each activity uses one of a set of resources R (We can consider activities using multiple resources without loss of generality). Each activity has an earliest start and a latest end time, a fixed duration and a fixed resource requirement. Each resource has a maximum capacity. All activities utilize the resource from the time it begins until the time it terminates (i.e. all resources are discrete in the sense defined by Laborie [1]). There may be precedence constraints between activities; if activity i must precede activity j, then the end of activity i must occur before the beginning of activity j (i.e. RCPSP). Our task is to search for a *Schedule*, by which we mean an order in which the activities execute that does not violate any of the resource capacity constraints or the precedence constraints. Two activities have a resource capacity constraint if they over consume the total capacity of their resource. Also, they have a precedence constraint if they can not be executed simultaneously, in other words if one activity is forced to be before or after the other. In the case of multi-capacity problems, activities can overlap, as long as the resource capacity constraints are not violated. An algorithm to propagate resource constraints on discrete resources is described in [1]. This procedure is called the Energy Precedence Constraint (*EC*). So, given an activity x the energy bound *EB* on the lower bound of the start time of x is calculated as follows:

$$L_B(x) \geq min(MinStart(A_i) + \Sigma(q(A_i) * d(A_i))/Q \quad (1)$$

Where, each A_i corresponds to every activity that is constrained to execute before x. q and d are the resource consumption and duration for each activity respectively, and $MinStart$ refers to the minimum possible start time for the activity (e.g. lower bound). Although this algorithm is very simple to compute, it may result in loose constraint bounds. As an example, consider two tasks A and B that precede some timepoint x. Let $d(A) = 5, d(B) = 10, q(A) = 3, q(B) = 2$. Assume both A and B use the same resource, and assume that the capacity of that resource is 4. Now, the EC tells us that $L_B(x) \geq \frac{5*3+10*2}{4} = 8.75$. But it is easy to see that these tasks can't overlap because of the resource bound, and so $L_B(x) \geq 15$. We will show that a simple analysis of the activities will provide mutual exclusion information that can improve $L_B(x)$ in cases like these with a very small cost. Mutual exclusion information has been widely used in planning to cut down the search or improve heuristics [9]. So, we will show that such information is also useful in the scheduling context.

3 Critical Path Analysis and Mutex Reasoning

We have seen already that the Energy Bound (*EB*) from the Energy Precedence Constraint *EC* described in Section 2 has some limitations. Specifically, it can provide loose computational bounds on the start and end time points of the activities when the total resource capacity is exceeded by their resource consumption. In this section, we will describe a technique to do constraint propagation based on critical path analysis and mutex reasoning. We will call this bound **Critical Path Mutex Bound (CPMB)**.

The precedence graph contains all of the current activities that have a precedence constraint between each other at any given time point. Each node in the graph represents

an activity from the original scheduling problem, and each edge represents a precedence constraint between the activities. Obviously, finding the *longest path* in this precedence graph (in terms of the total time of such path) from the beginning of the schedule to a given activity x leads to the **critical path** for such given task x. The sum of the durations of the activities in the critical path will indicate the lowest bound on the starting time point of x. In other words, x can not start before the bound indicated by its critical path in the precedence graph. It should be noted that propagation of Simple Temporal Networks as described in Dechter Meiri and Pearl [7] accomplishes this. Critical Path Analysis can be used also in the scheduling framework to find new bounds for tasks. These new bounds can help the scheduler to prune search branches that lead to the violation of temporal constraints, or guide the search by combining them with some heuristics.

Our framework will not only have a *precedence graph* but also a *mutual exclusion graph*. The precedence graph contains the activities as nodes and the precedence relations between them as edges. The mutual exclusion graph contains also activities, and mutex relations, which are not only obtained from the precedence information but also from the resource consumption of the activities (two activities are mutexes also if they consume more than the total capacity of their resource). In other words, the edges in the mutual exclusion graph represent all possible mutex relations in our scenario, which include mutexes by precedence and by resource consumption. Using this graph we can find any clique of mutex activities, and use it to augment if possible the critical path. Observe that a part of this graph is static, since the resource consumption of the activities will not change over time. Given this, a subset of the graph is computed just once before the search procedure starts, and it is used during the search process to augment the critical path for any task to get improved bounds. A *clique* in these graphs is a set of vertices in which each vertex has one edge to every other vertex. We can observe easily that any path in the precedence graph is a clique, since each activity on the path has a precedence constraint to every other activity in such a path. We can say then that these activities are mutexes with each other given the precedence constraints. We can retrieve these paths directly from the mutual exclusion graph. The precedence relations form the basis for the mutual exclusion relations in precedence constraint based scheduling. However, we still can augment such cliques by looking at the resource consumption of the activities inside the mutual exclusion graph of a given resource. So, if we have a path P that is a critical path for an activity x, we can augment the lower bound on the starting point of x by considering a set of activities $S \notin P$ before x, such that $\forall i \in S$ and $\forall j \in P$ the resource consumption of i and j exceeds the total capacity of the resource. We can observe that we are augmenting the initial clique formed from precedence relations, with activities from the mutual exclusion graph that are mutexes because of resource consumption with all the activities in such original clique. The same reasoning follows to compute the upper bound of any activity. The resulting bound is what we call **Critical Path Mutex Bound (CPMB)**.

To reduce the complexity of finding a critical path we only consider activities involving the same resource. This is reasonable in problems in which each task uses only one resource, as in our scenario. Under this circumstance the number of precedence relations in the graph is small, and the critical path computation remains feasible. The mutex cliques with respect to resource consumption are also limited because they are computed before hand only once, and used during the search to augment critical paths.

One final observation between the *EB* and *CPMB* bounds is that in problems involving unary-capacity resources like unary JSPs, both of the bounds have the same effect in propagating constraints. Given that in unary-JSP the total capacity of a resource is 1 and the total consumption of each activity is also 1, then the formula in [1] will be equal to computing the total durations of the tasks in the current set being evaluated, which in

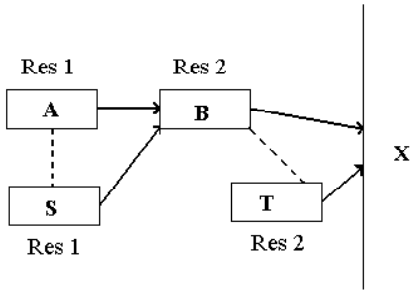

Fig. 1. Group Mutex Bound Example

consequence is equal to the critical path contained in that set. The resulting bound with *EB* will be the same than to the value obtained by the critical path of such a set. We will show some results that support our claim in Section 6. Our motivation is to show that this condition does not hold when we move into multi-capacity problems, where mutex and precedence information can really help us to propagate better constraints.

One of the main limitations of the Energy Bound *EB* is that it will only work for a set of tasks S using the same resource. So, our main motivation is to find a propagation procedure that works not only under this circumstance but also with tasks having mutex relations with each other and not necessary using the same type of resource. Given this, we can analyze bigger sets of tasks using the precedence relations or cliques, which can lead us to better bounds. Although the size of the set S is important, it is even more important to identify the interactions between the tasks in the set. This issue is captured by the mutual exclusion information. So, if we can identify mutex or precedence relations between tasks using multiple resources and time lines, then the bounds will be improved for propagation, as in the case of the CPMB described before. The energy bound will not benefit from this fact. We can see a very simple example of this scenario on Figure 1. In the figure, lines represent precedence relations and dashed lines represent mutex relations. Suppose that A and S are activities using one resource, and B and T are tasks using another resource. We can observe that A and S are mutexes because of their resource consumption, as well as B and T, but *EB* can not find this information since it relies in the precedence relations to calculate its bounds and there are not such relations between these activities. Moreover, the precedence relations in this example are defined between activities in multiple resources. Specifically, there is a precedence relation between A and B, and B and T, but these activities are using different resources and in consequence *EB* can not calculate any bounds. Mutex reasoning then can be very important to improve our constraint propagation. Since we may have multiple groups of activities interacting between multiple resources as in our last example. We need a definition that captures this information, in which we are looking for any bounds based on multiple groups.

Definition 1 (Group Mutex) *Two sets of activities A and B are mutex with each other if $\forall i \in A$ and $\forall j \in B$, i and j are mutex. Given this, we say that a group of sets of activities G is a* **Group Mutex** *iff every set S_g in the group is mutex with all other sets.*

We can see that computing such bound may be very expensive. However, we can compute cheaper forms of GMB. Specifically, we can find a small group mutex G, and

consider the bounds of such a group on different ways. We can also compute approximations of the longest path in those groups to avoid blowing up the search. Although, this particular constraint propagation technique has not been fully implemented, we will show its effectiveness with an example in the next section.

4 Methodology for Computing the Different Bounds

We will show in this section the different ways to compute bounds for propagation. We will illustrate with an example such computation, and we will argue the different methods used in our framework to integrate the algorithms for propagating the resource constraints.

4.1 Developed Example

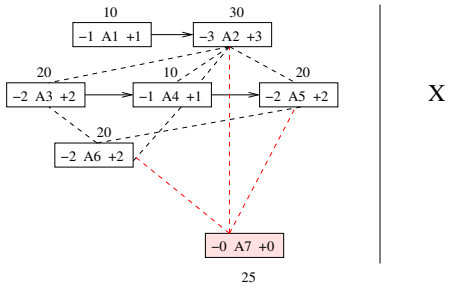

Fig. 2. Propagation of Constraint Bounds

We will show with an example the different bound values obtained by the techniques described in the last section. In Figure 2 there are 7 tasks, which are constrained to occur before a particular task X. Six tasks are competing for the same resource R. In particular, $A_1 ... A_6$ consume different amounts of such resource. Tasks correspond to nodes in the graph. The numbers above the nodes indicate the durations of the activities. We assume that each task releases the same amount of resource used at the end of its execution. Total quantity of the resource R for this example is 3. Precedence constraints are represented by arrows in the graph, and mutex constraints are represented by dashed lines. Notice that any precedence relation implies a mutex relation, but not viceversa. A mutex relation can be based on the resource consumption of the activities, if they exceed the capacity of the resource then they are obviously mutex with each other. In Figure 2, A_2 and A_3 are mutexes by resource consumption. However, the mutex constraints can also be caused by another reason due to a different resource. In our example, we have A_7 mutex with A_2, A_5, and A_6, since A_7 uses another resource that interacts with the first resource [6]. From the discussion in the last section, we can compute the following bounds:

Energy Bound (EB)= ((1*10) + (3*30) + (2*20) + (1*10) + (2*20) + (2*20)) / 3 = 230/3 = 76.66

Critical Path Mutex Bound(CPMB): If we consider the $criticalpath$ of the graph in Figure 2, then the longest precedence path will be A_3, A_4, A_5, with a bound of **Critical Path** = 20+10+20=50. But, now we can do mutex analysis to improve even more

this initial bound given by the critical path. From the mutual exclusion graph, we can see that A_2 is mutex with every activity on the critical path because of resource consumption. Then, the final estimate for *CPMB* is 80.

If we would like to estimate the bound given by *GMB*, there are multiple partitions of the activities in this example. Let's consider two of them. Suppose there are two groups, S_1 and S_2. Let S_1 = A_2, and S_2 = A_1, A_3, A_4, A_5, A_6. The best bound for S_2 is given by a clique of length 60 (A_3, A_5, A_6). So, the **GMB** = 30 + 60 = 90. We can observe that *GMB* can be very useful in problems involving multiple resources. However, finding good partitions of a set of activities in multiples groups remains an open question. Such partitions can be based on paths and can be augmented with additional mutex activities.

We have presented in this section an example which clearly indicates the benefits of using mutex information for propagating constraints bounds on multi-capacity scheduling problems. We have seen that *CPMB* can provide a better estimate than *EB* in problems where there are many activities being mutexes by some reason other than a precedence relation (i.e. resource consumption).

4.2 Possible Search Scenarios

It is important to identify what kind of scenarios can arise during the search, such that each propagation technique can benefit from it. This is the first step in understanding which type of bound reasoning would be more adequate given the current snapshot of the search. Most of the discussion is based on the Energy Bound **(EB)** and critical path plus mutex bounds **(CPMB)**.

Scenario 1 *When we have more precedence constraints forming long critical paths,* CPMB *will provide better bounds than* EB

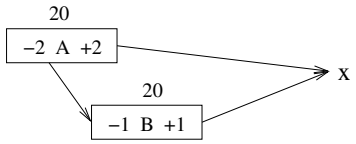

Fig. 3. Critical Path Analysis Scenario

Figure 3 shows an example of such a situation. In this example, tasks A and B are constrained to be before X in the precedence graph. A consumes 2 units of the resource and B 1 unit. The resource has a maximum capacity of 4. Furthermore, the duration of each of the activities is 20. The Energy Constraint will find a bound of *EB*= ((20*2)+(20*1))/4 = 15. This bound is even smaller than the duration of any of the tasks, which is 20. Instead, the critical path analysis *CPMB* will return the bound value of 40, since there is a precedence constraint between A and B forming these two actions the critical path to X.

Scenario 2 *When critical paths are short, it is more likely that we may have a set of tasks consuming the same resource, which can possibly overlap. Therefore, the Energy Constraint procedure may give a better bound.*

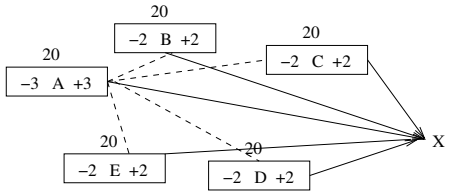

Fig. 4. Energy Constraint Procedure Scenario

We can observe one example of this scenario in Figure 4. In this example all five tasks have a duration of 20, and each one consumes 2 units of the resource excepting A, which consumes 3 units. The capacity of the resource is 4 as in the previous example. In this example, the Energy Constraint will find a bound *EB*=(20*2*4 + 20*3)/4=50. At the same time the critical path analysis alone will find a bound of 20, since there are no precedence relations between the activities all of the paths in the graph are very short. *CPMB* will give a bound of 40, since A is mutex with everybody else. We can observe then that *EB* is better for this particular example. However, notice that if we do group mutex reasoning, the bound obtained will be better than the *EB*. Given that A is mutex with everybody else because of its resource consumption, then A forms a group and the rest of the activities one more. Then, *GMB*=Bound(A)+Bound(B, C, D, E)=20+(20*2*4)/4=60.

Under these scenarios one technique can work better than the other one. The complexity of critical path analysis is compensated by the accuracy of the procedure when mutex information is given. This combination of factors allows *CPMB* to remain competitive with respect to *EB* in the scenarios such as that shown in Figure 4, and to outperform *EB* in the other ones. We will present some empirical results on this intuition on Section 6 by running both techniques in unary-capacity and multi-capacity single resource scheduling problems.

5 Experimental Framework

We have discussed in general form the application of different constraint propagation techniques. In this section, we discuss some details of the applicability of such techniques into our framework, and how the framework may affect the computing procedure or usefulness of the different bound analysis. The characteristics of our implementation related to resources are:

1. The temporal relations between time points are managed by a Simple Temporal Network (STN).
2. Whenever the ordering between two tasks is established, the constraint manager will automatically enforce consistency in the temporal network. When a precedence variable between two tasks is set, the STN will do propagation and enforce new lower/upper bounds on the start/end time points of any involved tasks.
3. We employ a resource manager that assumes that resources are discrete.
4. On top of these structures, we have implemented a simple backtracking algorithm that performs variable and value ordering using heuristics. The search is conducted over the relative order of pairs of tasks that use the same resource; in the case of multicapacity resources, we include an explicit “overlap” choice as well. We use the B-Slack heuristics of [5] for both variable and value ordering.

5. The "overlap" choice is preferred during the search by the heuristics. So, if any two tasks have the opportunity of "overlap" because they do not exceed the resource capacity of the resource, then the heuristics choose this option first, otherwise it computes the B-Slack as described in [5] and picks the best.

The idea behind B-Slack is that we want to choose the variable with the overall minimum temporal slack in its sequencing decision. And a value that provides the maximum slack for the variable. A complete description of the heuristic is found in [5]. Although this heuristic was designed for problems involving single capacity resources, it still helps in our context. Following these specifications, there are some important questions that need to be answered with respect to our design. The first one is how to obtain the set of tasks before/after a given task. We have two choices in our implementation. We can either rely on the values of the decision variables or compute every time the temporal relations from the temporal network.

Another question is when should we call any of the propagation techniques discussed in this article (e.g. *EB*, *CPMB*, etc). The most natural way is to call them whenever we assign a value to a precedence variable (e.g. before or after a given task), such that new lower and upper bounds can be propagated on the start and end time points of the tasks involved in the decision. Another approach is to consider running any constraint propagation technique before assigning any value to the current variable. In other words, if we have an X variable with domain *before,after*, then we consider one assignment to the variable (e.g. before), and run the constraint propagation technique. And, we do the same with the rest of the values of the variable. After the constraint propagation procedure is done, we compute the slacks of the variable again and choose the best value for the decision. The intuition is that by running the constraint propagation procedure first the different slack values of the variable may change, and the heuristics will be able to capture such information. However, we can observe that such a technique is more computationally expensive since it propagates in all values of a given variable. An empirical evaluation of these two techniques will be presented in Section 6.

6 Empirical Evaluation

All the experiments have been run on a $SUNW, Ultra - 80$ workstation running $Solaris$ with $1GB$ of RAM. The tables reflect the solution of the problems running B-Slack based heuristics without any constraint propagation (BS), with Energy Constraint ($BS + EB$), and with Critical Path Mutex Bounds ($BS + CPMB$). In order to show the effect of computing the propagation technique in different ways as explained in the last section, we include a set of experiments running first *EB* and calculating the heuristic next ($EBF + BS$). Observe that we did not include *CPMB* in such analysis because the result of its propagation procedure is the same than the one from *EB* in unary JSPs. All of the problems have been randomly generated. In single machine unary problems, a number of resources and jobs are given to the generator, as well as the range for the durations of the activities. The generator then creates randomly all of the tasks for each job, and their durations. In case of multi-capacity problems, a total capacity is given for each resource, and the generator creates in addition to the information explained above, the resource consumptions for each task in the problem. The problems were solved different number of times considering different horizon values. Once the data is collected, the horizon that seems close to the optimal is chosen to run the final tests in all propagation techniques. The reason for this procedure is that we do not want a very loose horizon because *EB* and *CPMB* would not propagate any constraints.

Table 1 shows the execution time and number of nodes generated by the different techniques. All the running times are in seconds. The problems are single machine

Prob	BS	BS+EB	BS+CPMB	EBF+BS
T1	43.89/289	3.03/90	3.59/90	4.37/90
T2	48.46/284	14.88/131	15.56/131	16.29/131
T3	706.71/3318	8.11/105	8.80/105	9.55/105
T4	74.86/411	4.71/93	5.14/93	6.07/93
T5	21.69/176	7.24/109	7.92/109	8.54/109
P1	948.86/2131	49.34/219	54.09/219	48.35/208
P2	28838/59447	10.16/147	11.55/147	13.99/147
P3	623.54/1482	36.60/227	39.10/227	41.48/227
P4	4834.49/9132	22.28/168	23.95/168	26.42/168
P5	649.12/1636	7.91/147	9.77/147	11.87/147

Table 1. Single Machine Unary Capacity Random Generated Problems

unary capacity, where each activity uses one resource and consumes one unit of that resource. The size of the problems are 6x6 (T set) and 7x7 (P set). We can observe on this table that *EB* and *CPMB* will perform almost equally. In this set of problems there is not additional mutex information that can help *CPMB*. As introduced in Section 3, each activity consumes the total capacity of the resource, because it is a unary problem, and by consequence the bound obtained by *EB* will be the same than the length of the critical path. Since there is no additional mutex information for *CPMB*, this technique will return the length of the critical path, and by consequence *EB* and *CPMB* will propagate the same amount of constraints during the search process. This has been already verified, but it has not been included in the table for space limitations. We can observe that there is a minor difference in the execution time of both techniques, given that *CPMB* is more expensive to compute, but still *CPMB* has an acceptable performance. The last column of Table 1 shows the results by doing constraint propagation first and then using the heuristics to select the next best possible value for a variable. We can observe that the effect of doing the propagation procedure first does not seem to affect the heuristics.

Table 2 shows the execution time and number of nodes in single machine multi-capacity problems. We have two sets of problems, the set S (5x5) and the set P (6X6). These sets of problems includes additional mutex information for *CPMB*, because there are mutex constraints given not only by precedence relations, but also by the resource consumptions between the activities. Recall from Section 3 than *CPMB* uses such mutex information to augment its critical path and get improved bounds. We can observe that the mutex information is in fact very useful, since it helps to reduce the number of nodes and increase the efficiency of the search process. Another observation that we found is that the bigger the scheduling problem the more we can find activities being mutexes by resource consumption. However, finding paths becomes more difficult too. By doing this simple mutex analysis, we have shown that *CPMB* in fact can result in improved bounds for the problems with little cost. We can observe also that *EB* seems to confuse the heuristics in these kinds of problems. In fact, searching using only heuristics is better than using *EB* in some of the problems from Table 2. For *CPMB* this is not the case, because the combination of the heuristics and *CPMB* did not lead to bad performance.

7 Conclusions and Future Work

We have described a technique to propagate resource constraints using mutex reasoning in precedence based scheduling. Such technique called *CPMB* (critical path plus mutex bounds) takes into account a precedence graph, and computes the critical path for any

Prob	BS	BS+EB	BS+CPMB
S1	325.35 / 1967	336.75 / 1970	237.85 / 1390
S2	24.40 / 198	26.11 / 198	6.85 / 86
S3	443.05 / 2775	371.32 / 2227	305.74 / 1835
S4	324.34 / 2154	292.76 / 1785	274.39 / 1653
S5	106.98 / 741	89.93 / 575	56.64 / 341
T1	162.91 / 496	330.92 / 976	121.17 / 380
T2	977.44 / 2568	741.13 / 1916	645.41 / 1701
T3	102.01 / 370	105.59 / 370	69.10 / 259
T4	6686.13 / 20354	7793.10 / 20354	6119.71 / 17569
T5	187.66 / 472	196.61 / 472	138.22 / 327

Table 2. Single Machine Multiple Capacity Random Generated Problems

given activity to propagate, and augment such a path using mutex information based on resource consumption from a mutual exclusion graph. Although, this technique seems very complex, we have decreased the complexity by considering problems in which each activity uses only one resource. We have compared our constraint propagation procedure to a state of the art algorithm, the Energy Constraint *EB*. We have shown that our implementation remains competitive in its worst case scenarios on single machine unary capacity problems, and it results in improved bounds on multi-capacity problems where mutex information is available. Since *EB* works only on problems in which tasks use only one resource, we have proposed a technique based on *CPMB* which will allow to scale up to multiple machine multiple capacity problems. This technique is called *GMB*, group mutex bounds. Specifically, we have demonstrated with some examples the effectiveness *GMB* on these kinds of problems. Complete implementation of this technique and alternative approximations for the calculation of critical paths are being considered as future work.

References

1. P. Laborie.: Algorithms for Propagating Resource Constraints in AI Planning and Scheduling: Existing Approaches and New Results In *Proc. ECP-01*, 2001.
2. B.Drabble and A.Tate.: The Use of Optimistic and Pessimistic Resource Profiles to Inform Search in an Activity Based Planner. In *Proc. AIPS-94*, 1994.
3. J.Carlier and E. Pinson.: A Practical Use of Jackson's Preemptive Schedule for Solving the Job-Shop Problem. In *Annal of Operation Research*, 1990.
4. W. Nuijten.: Time and Resource Constrained Scheduling: A Constraint Satisfaction Approach. PhD Thesis, Eindhoven University of Technology, 1994.
5. S.F. Smith and C. Cheng.: Slack-Based Heuristics for Constraint Satisfaction Scheduling. In *Proc. AAAI-93*, 1993.
6. D. Smith, J. Frank and A. Jonsson.: Bridging the Gap Between Planning and Scheduling. In *Knowledge Engineering Review*, 2000.
7. R. Dechter and I. Meiri and J. Pearl.: Temporal Constraint Networks In *Artificial Intelligence*, vol. 49, 1991.
8. P. Lopez and P. Esquirol.: Consistency enforcing in Scheduling: A general formulation based on energetic reasoning 5th. International Workshop on Project Management and Scheduling *PMS'96*, 1996.
9. X. Nguyen, S. Kambhampati and R. Sanchez.: Planning Graph as the Basis for deriving Heuristics for Plan Synthesis by State Space and CSP Search. To appear in Artificial Intelligence.