Propagation over the meets temporal constraint

Andrew J. Davenport, J. Christopher Beck, Mark S. Fox

Department of Computer Science and Department of Industrial Engineering,
University of Toronto,
Toronto, Ontario, CANADA

andrewd@ie.utoronto.ca, chris@cs.utoronto.ca, msf@ie.utoronto.ca

Abstract

Many real life scheduling problems have more complicated temporal constraints than that found in the job shop scheduling model. One example which occurs frequently is the "meets" constraint. Although this constraint can be dealt with using techniques developed for the job shop scheduling model, specialised techniques can significantly improve performance. In this paper we present a simple but effective constraint propagation mechanism for tackling scheduling problems with the "meets" constraint between activities in a process plan. On a set of randomly generated benchmark problems we show significant performance gains by using this propagation scheme.

1. Introduction

The determination and utilisation of implied constraints during search by constraint propagation has been found to dramatically improve the performance of many constraint satisfaction and scheduling algorithms (?, ?). Recently a number of specialised propagation techniques have been developed for specific constraints such as the alldifferent constraint (?) and the global cardinality constraint (?), as well as many constraints found in scheduling, such as disjunctive (?, ?) and cumulative resource constraints (?), preemptive scheduling (?) and scheduling with sequence-dependent changeover constraints (?).

The research described in this paper came about as a result of our experience on tackling real world scheduling problems, where more complicated constraints than those found in the job shop model had to be dealt with. The particular issue we discuss in this paper is propagation through the "meets" constraint. The meets constraint between two activities A and B specifies that the end time of activity A must "meet", or be equal to, the start time of activity B (?). The work described in this paper can also be extended to the "meets with offset" constraint, which states that a fixed duration must elapse between the end time of one activity and the start time of another[1].

In this paper we describe a simple but powerful constraint propagation mechanism for deducing implied constraints for scheduling problems with the meets constraint. In our application the meets constraint was used in the modelling of intermediate storage of material between different processing stages. However

[1]This constraint can be modelled either by including the offset directly in the constraint or by creating a new "null" activity of duration equal to the offset, which meets (with no offset) and is met-by the activities to which the meets with offset constraint applies to.

scheduling problems with the meets constraint between activities in a process plan can be found in many industrial applications; in particular where there are dangers of product spoilage or where there are safety considerations to take into account. For example, in semiconductor manufacturing, silicon wafers must be processed as soon as possible after being made in the clean room, to prevent risk of contamination. In the pharmaceutical industry there are safety considerations when volatile materials are created as a result of a reaction at some point in the manufacturing process. Ideally, one wants to process these materials into something less dangerous as quickly as possible. In steel industry scheduling there are many instances where the meets constraint is important. For instance, steel from a furnace goes into an "acid pickle" for cleaning. The steel must then be immediately coated, with either an enamel or zinc coating, to prevent oxidation.

The balance of this paper is organised as follows: after formalising the job shop scheduling problem with meets constraints, we present and illustrate the ideas behind meets propagation in section 3.1. To evaluate the effectiveness of this new technique, we generated a number of benchmark problems with a similar, but simplified, structure to the industrial problem we were trying to solve. We present results and analysis from an empirical study using these problems in section 4. We conclude in section 6.

2. Job shop scheduling with meet constraints

2.1 Definition

The $n \times m$ job-shop scheduling problem with meets constraints is formally defined as follows: Given are a set of n jobs, each composed of m totally ordered activities, and m resources. Each activity A_i within a job requires exclusive use of a single resource R_j for some processing duration dur_i. For this paper we assume that all activity processing durations are greater than zero. There are three types of constraints in this problem:

- precedence constraints between two activities in the same job stating that if activity A is before activity B in the total order then activity A must execute before activity B;
- meets constraints between two activities in the same job stating that if activity A meets activity B then the end time of A must be equal to the start time of B;
- unary resource capacity constraints specifying that no two activities requiring the same resource may execute at the same time.

Jobs have release dates (the time after which the activities in the job may be executed) and due dates (the time by which the last activity in the job must finish). In the decision problem, the release date of each job is 0 and a global due date is D. The problem is to determine whether there is an assignment of a start-time to each activity such that the precedence constraints and resource constraints are satisfied and the maximum finish time of all jobs is less than or equal to D. This problem is more general than the job shop scheduling problem, and thus is NP-hard (?).

2.2 Notation

For an activity, A_i, ST_i is the start time variable, ET_i is the end time variable and STD_i is the discrete domain of possible start times. res_i represents the resource required by activity A_i. est_i and lst_i represent the earliest and latest possible start times, while eft_i and lft_i represent the earliest and latest possible finish times respectively. dur_i is the duration of A_i. We will omit the subscript unless there is the possibility of ambiguity.

Given two activities A and B connected by a precedence or meets constraint ($A \rightarrow B$), activity A is said to be *upstream* of activity B and similarly, activity B is *downstream* of activity A.

3. Meets constraint propagation

3.1 An example

Meets propagation is used when a new constraint is posted (*i.e.*, added to the graph) during the search and one or more of the activities involved in the new constraint has a meets constraint.

For example, Figure 1(a) illustrates meets propagation when a new precedence constraint is posted. Figure 1(a) displays a problem which has two jobs J_1 and J_2, each composed of three activities A_1, A_2, A_3 and A_4, A_5, A_6 respectively. Activities within each job are connected via meets constraints *e.g.*, in J_1 activity A_1 meets A_2 and activity A_2 meets A_3. These constraints are represented by solid arcs in the disjunctive constraint graph[2].

We also have resource capacity constraints between activities in each job, represented by dashed arcs in the disjunctive constraint graph of Figure 1. For instance, activities A_3 and A_6 both execute on the same resource R_2. Since all resources have unary capacity, only one activity may be executing on any one resource at any time. Thus we have the following disjunctive resource constraint on the start times of activities A_3 and A_6:

$$ST_3 + dur_3 \leq ST_6 \vee ST_6 + dur_6 \leq ST_3 \qquad (1)$$

This constraint states that either activity A_3 executes before A_6 or A_6 executes before A_3. At some point in the search we have to make a decision on the sequencing of activities A_3 and A_6 on resource R_2. Let us consider what would happen if we posted the constraint that A_6 executes before A_3 on R_2, *i.e.,*:

[2]A meets constraint between two activities A and B can be dealt with by representing the end time of A and the start time of B by a single interval variable, since they must be equal. This technique does not work for the meets with offset constraint however.

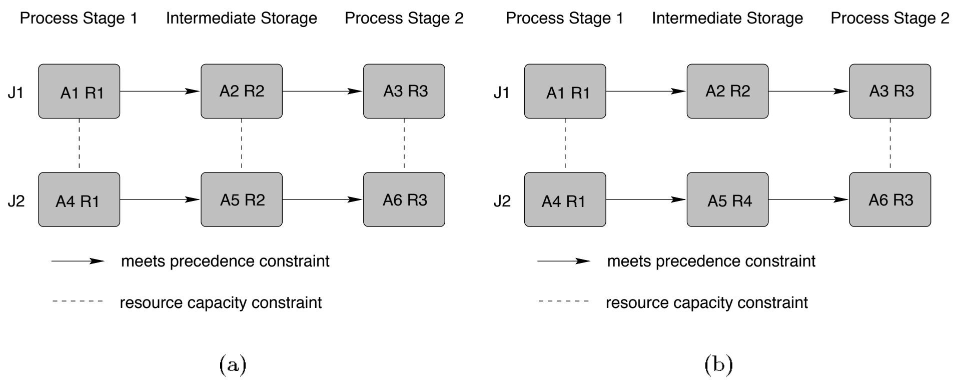

Figure 1: *A disjunctive constraint graph representing an example scheduling problem with the meets constraint. Each node in the constraint graph is characterised by a 2-tuple: (activity name, resource required), e.g., (A1 R1).*

$$ST_6 + dur_6 \leq ST_3 \qquad (2)$$

Since activity A_2 meets A_3, activity A_5 meets A_6, and both A_2 and A_5 require the same resource, it must be the case that A_5 executes before A_2. The reasoning behind this is as follows:

> Activity A_6 executes before activity A_3. Therefore the end time of A_6 must be less or equal to the start time of A_3. Since activity A_5 executes before A_6, the end time of A_5 must be less than the start time of A_3[3]. However, the end time of A_2 is the same as the start time of activity A_3, since these two activities are connected by a meets constraint. Thus the end time of A_5 must be less than the end time of A_2. Since A_2 and A_5 require the same resource and only one activity can execute on the resource at any one time, A_5 must execute before A_2.

Thus we can now post the implied constraint:

$$ST_5 + dur_5 \leq ST_2 \qquad (3)$$

3.2 Propagation after precedence constraint posting

When posting precedence constraints we have four cases for meets propagation: upstream and downstream propagation when the neighboring activities are on

[3]We assume all activity durations are greater than zero.

the same resource and when they are on different resources. The example above is in the case of upstream propagation to activities which share a resource. More formally, the propagation rule for this case is:

$$\begin{aligned}\forall a, b, c, d \quad & a \text{ meets } b \wedge c \text{ meets } d \wedge \text{resource}(a) = \text{resource}(c) \\ & \wedge\, d \text{ before } b \\ & \qquad\qquad \rightarrow c \text{ before } a \end{aligned} \tag{4}$$

Rule 4 states that if the neighboring upstream activities (connected via meets constraints) must execute on the same resource, then from the posting of a precedence constraint we can infer the sequence of the upstream pair.

A similar rule can be formulated for downstream propagation:

$$\begin{aligned}\forall a, b, c, d \quad & a \text{ meets } b \wedge c \text{ meets } d \\ & \wedge\, \text{resource}(b) = \text{resource}(d) \wedge c \text{ before } a \\ & \qquad\qquad \rightarrow d \text{ before } b \end{aligned} \tag{5}$$

Rules 4 and 5 are only applicable when the neighboring activities require the same unary capacity resource. However, even if the neighboring pair (*e.g.,* A_2 and A_5 in Figure 1(a)) did not execute on the same resource we can still deduce a necessary relationship between them. Consider the example temporal network illustrated in Figure 1(b). Here the intermediate storage activities require different storage resources. In this case, after posting the constraint that A_6 executes before A_3 on R_2, as in our previous example, we can now deduce that A_5 finishes at or before the end of A_2; that is the finish time of A_5 must equal to or before the finish time of A_2:

$$ST_5 + dur_5 \leq ST_2 + dur_2. \tag{6}$$

This reasoning leads to the following propagation rule which is a weaker form of Rule 4:

$$\begin{aligned}\forall a, b, c, d \quad & a \text{ meets } b \wedge c \text{ meets } d \wedge \text{resource}(a) \neq \text{resource}(c) \\ & \wedge\, d \text{ before } b \\ & \qquad\qquad \rightarrow c \text{ finishes before or at } a \text{ finishes} \end{aligned} \tag{7}$$

In a similar way, when propagating downstream (*e.g.,* after posting A_1 executes before A_4), we can infer a relationship between the start-times of the neighboring activities (*e.g.,* A_2 and A_5) even if they do not share a resource. After posting the precedence constraint, we can further post that that A_2 must start at or before A_5 starts: the start-time of A_2 must be less than or equal to the start-time of A_5. In terms of a propagation rule, we have the following, weakened form of Rule 5:

$$\begin{aligned} \forall a, b, c, d \quad & a \text{ meets } b \wedge c \text{ meets } d \\ & \wedge \text{resource}(b) \neq \text{resource}(d) \wedge c \text{ before } a \\ & \quad \rightarrow d \text{ starts before or at } b \text{ starts} \end{aligned} \tag{8}$$

3.3 Propagation after posting other temporal constraints

In above section, we presented four meets propagation rules that can be used when a precedence constraint is added to the evolving constraint graph. These rules, themselves, results in further constraint postings. In fact, three types of temporal constraints are posted by the above rules:

- precedence – Rules 4 and 5
- ends at or before end – Rule 7
- starts at or before start – Rule 8

Generalizing from the above rules, we can, under some conditions,continue meets propagation after posting either of the latter two constraint types.

3.3.1 Propagation after posting finishes at or before

With analogy to the propagation rules after precedence constraint posting, we have four cases for meets propagation after a finishes at or before constraint has been posted: upstream and downstream propagation when neighboring activities do and do not share a resource requirement.

Consider once again the example temporal network illustrated in Figure 1(b) after we have posted the constraint A_5 finishes at or before the end of A_2. Can we determine from this constraint any implied sequencings of the activities A_1 and A_4 upstream of the ones involved in this constraint? This would depend upon the durations of activities A_2, A_4 and A_5. If the combined durations of A_4 and A_5 are greater than that of A_2 then the end time of activity A_4 must be less than that of A_1, since A_1 meets A_2 and A_4 meets A_5. Since A_1 and A_4 require the same resource we can then deduce that activity A_4 executes before A_1. Thus we can now post:

$$ST_4 + dur_4 \leq ST_1 \tag{9}$$

The propagation rule that allows this deduction is as follows:

$$\begin{aligned} \forall a,b,c,d \quad & a \text{ meets } b \wedge c \text{ meets } d \wedge \text{resource}(a) = \text{resource}(c) \\ & \wedge\, d \text{ finishes before or at } b \text{ finishes} \\ & \wedge\, \text{duration}(c) + \text{duration}(d) > \ \text{duration}(b) \\ & \qquad\qquad \rightarrow c \text{ before } a \end{aligned} \tag{10}$$

Similarly, consider activities A_3 and A_6 downstream of the finishes at or before constraint. Because A_5 finishes at or before the end of A_2 it must be the case that the start-time of A_6 must start at or before the start-time of A_3. The disjunctive resource constraints prevents A_6 and A_3 from overlapping, therefore it must be the case that A_6 must execute before A_3. This reasoning results in another propagation rule:

$$\begin{aligned} \forall a,b,c,d \quad & a \text{ meets } b \wedge c \text{ meets } d \\ & \wedge\, \text{resource}(b) = \text{resource}(d) \wedge c \text{ finishes before or at } a \text{ finishes} \\ & \qquad\qquad \rightarrow d \text{ before } b \end{aligned} \tag{11}$$

Rules 10 and 11 do not apply when the neighboring activities do not execute on the same resource. For example, we can not, with these rules, deduce a relationship between this if the activities A_1 and A_4 in Figure 1(b) do not execute on the same resource. In this case, however, if the duration of A_5 is greater than that of A_2 we can still deduce that A_4 finishes before A_1 finishes (which may aid further upstream propagation), allowing us to post:

$$ST_4 + dur_4 \leq ST_1 + dur_1 \tag{12}$$

The new propagation rule, a weakened form of Rule 10 therefore, is:

$$\begin{aligned} \forall a,b,c,d \quad & a \text{ meets } b \wedge c \text{ meets } d \wedge \text{resource}(a) \neq \text{resource}(c) \\ & \wedge\, d \text{ finishes before or at } b \text{ finishes} \\ & \wedge\, \text{duration}(d) > \ \text{duration}(b) \\ & \qquad \rightarrow c \text{ finishes before or at } a \text{ finishes} \end{aligned} \tag{13}$$

Downstream propagation with different resources is again quite similar based on a variation of Rule 11:

$$\begin{aligned} \forall a,b,c,d \quad & a \text{ meets } b \wedge c \text{ meets } d \\ & \wedge\, \text{resource}(b) \neq \text{resource}(d) \wedge c \text{ finishes before or at } a \text{ finishes} \\ & \qquad \rightarrow d \text{ starts before or at } b \text{ starts} \end{aligned} \tag{14}$$

It should be noted that the weaker rules (*e.g.,* Rules 10 and 14) are not useful by themselves–their effect on the pruning of domains is equivalent to that achieved by usual arc-B-consistency temporal propagation (?). However, with the meets propagation rules that we have introduced here, it is possible that subsequent meets propagation will results in the deduction of new, stronger constraints that temporal arc-B-consistency alone would not have been able to find.

3.3.2 Propagation after posting starts at or before

Our final set of propagation rules arise from adding the starts at or before constraints to our graph as is done in Rules 8 and 14. The reasoning here is analogous to that for the rules that have come before and so we leave the explicit statement of the reasoning as an exercise.

After posting of a starts at or before constraint, we have the following four propagation rules that maybe applicable:

- Upstream propagation, same resources

$$\begin{aligned}\forall a,b,c,d \quad & a \text{ meets } b \wedge c \text{ meets } d \wedge \text{resource}(a) = \text{resource}(c) \\ & \wedge\, d \text{ starts before or at } b \text{ starts} \\ & \qquad\qquad \rightarrow c \text{ before } a \end{aligned} \tag{15}$$

- Downstream propagation, same resources

$$\begin{aligned}\forall a,b,c,d \quad & a \text{ meets } b \wedge c \text{ meets } d \\ & \wedge\, \text{resource}(b) = \text{resource}(d) \wedge c \text{ starts before or at } a \text{ starts} \\ & \wedge\, \text{duration}(a) + \text{duration}(b) > \text{duration}(c) \\ & \qquad\qquad \rightarrow d \text{ before } b \end{aligned} \tag{16}$$

- Upstream propagation, different resources

$$\begin{aligned}\forall a,b,c,d \quad & a \text{ meets } b \wedge c \text{ meets } d \wedge \text{resource}(a) \neq \text{resource}(c) \\ & \wedge\, d \text{ starts before or at } b \text{ starts} \\ & \qquad \rightarrow c \text{ finishes before or at } a \text{ finishes} \end{aligned} \tag{17}$$

- Downstream propagation, different resources

$$\begin{aligned}\forall a,b,c,d \quad & a \text{ meets } b \wedge c \text{ meets } d \\ & \wedge\, \text{resource}(b) \neq \text{resource}(d) \wedge c \text{ starts before or at } a \text{ starts} \\ & \wedge\, \text{duration}(a) > \text{duration}(c) \\ & \qquad \rightarrow d \text{ starts before or at } b \text{ starts} \end{aligned} \tag{18}$$

3.4 Summary

The meets propagation rules and the cases in which they are applicable are summarized in Table 1. These rules can be applied every time we post a new temporal constraint on activities executing on the same resource. The worst case time complexity of meets propagation is $O(n^2)$ in the number of activities in the problem, since we may have to examine all pairs of activities to determine if the rules apply.

	Upstream		*Downstream*	
Constraint Added	*Same resource*	*Different resource*	*Same resource*	*Different resource*
before	(4)	(7)	(5)	(8)
finishes	(10)	(13)	(11)	(14)
starts	(15)	(17)	(16)	(18)

Table 1: *Summary of propagation rules*

4. Empirical evaluation

4.1 Benchmark problems

In order to evaluate the effectiveness of meets constraint propagation, we generated a range of benchmark problems. We wanted to approximate, but simplify the structure of the industrial problem we were looking at. To this end, the problems in our problem set had three process stages, that is three activities per job. The first and third stages represent a manufacturing process, while the second stage represents an intermediate storage stage for the products being manufactured. In each job there are meets constraints between the first and second stages and the second and third stages. There are only a limited number of unary capacity resources in this simplified problem, each of which have to be shared by the activities in different jobs. Thus there are disjunctive resource constraints between activities requiring the same resource.

We generated problems from 20 to 40 jobs, with 100 problems at each problem size. We determined the makespan for each problem in the following way: we first determined its Taillard lower bound (TLB) for job shop scheduling, as given in (?). We then generated one problem set where each problem had a makespan of 1.3$\times$ its TLB and another problem set where the makespan was set to 1.5$\times$ its TLB. The problems were not guaranteed to be soluble. In fact for some problems (even at $1.5 \times TLB$) initial constraint propagation immediately detected infeasibility before search commenced.

All experiments were run using NumODO, a constraint-based scheduling system developed at the University of Toronto and Numetrix Limited (?). Pseudocode for the outline scheduling algorithm is given in Algorithms 1 and 2. In all experiments we used the VarHeight texture measurement to make heuristic

```
procedure Scheduling
1   finished := false;
2   while (finished = false)
3       perform constraint propagation;
4       if (constraint propagation finds no commitments) then
5           make heuristic activity sequencing commitment;
6       if (reached dead-end) then
7           backtrack;
8       else
9           arc-B-consistency temporal propagation;
10      if (all-activities-sequenced or CPU limit reached) then
11          finished := true;
```

Algorithm 1: The outline scheduling algorithm

```
procedure Constraint-Propagation
1   while (new constraints implied) do
2       Meets-Propagate-Upstream;
3       Meets-Propagate-Downstream;
4       Constraint-Based-Analysis;
5       EdgeFinding;
```

Algorithm 2: Constraint propagation

decisions (line 5 of Algorithm 1) (?). For constraint propagation on resource constraints (Algorithm 2) we used constraint-based analysis (?, ?) and edge-finding (?)[4] in addition to meets constraint propagation.

We experimented with two backtracking techniques: chronological backtracking and limited discrepancy search (LDS) (?). Previous results have suggested that LDS can significantly outperform chronological backtracking on job shop scheduling problems (?, ?). Thus in total we compared four algorithms:

- NumODO with chronological backtracking and meets propagation
- NumODO with chronological backtracking, no meets propagation
- NumODO with limited discrepancy search and meets propagation
- NumODO with limited discrepancy search, no meets propagation

For each algorithm we set a CPU time bound of 10 minutes to solve each problem on a HP 100 MHz 9000/712 running HPUX 9.05, after which search was terminated if no solution had been found.

[4]We implemented edge finding as described in (?) for job shop scheduling, although it would be possible to improve the performance of this in the presence of meets constraints.

4.2 Results

Results for three stage problems (three activities per job) are presented in Tables 2–6 for $1.3 \times TLB$, and in Tables 7–11 for $1.5 \times TLB$.

All search statistics are presented only for the problems solved by each algorithm. If a problem was not solved within the CPU time bound we do not include that problem in the search statistics.

4.3 Discussion of results

Adding meets propagation to NumODO, either with chronological backtracking or limited discrepancy search, always resulted in more problems being solved within the CPU time bound. Furthermore, although using meets propagation resulted in solving more problems within the CPU time bound, this was done with a significantly lower mean CPU time, significantly less heuristic commitments and significantly less backtracks than when meets propagation was not used.

Ranking algorithms by the number of problems solved gives us the following:

1. NumODO with limited discrepancy search and meets propagation
2. NumODO with chronological backtracking and meets propagation
3. NumODO with chronological backtracking, no meets propagation
4. NumODO with limited discrepancy search, no meets propagation

The fact that when there is no meets propagation, chronological backtracking outperforms limited discrepancy search is surprising. What we find here is that the difference between LDS with and without meets propagation is much greater than the difference for chronological backtracking. We believe this can be explained in the following way, with reference to Figure 1(a). Let's say that in the problem state represented by the constraint graph in this figure we post a precedence constraint stating that activity A_3 executes before activity A_6 on resource R_2. Then later on in the search we try to post the constraint that activity A_4 executes before A_1 on resource R_1. These two constraints together lead to a contradiction, since $A_3 \rightarrow A_6$ implies that $A_1 \rightarrow A_4$. This would be deduced by meets propagation, however if meets propagation was not being used the above scenario could occur. LDS would do the wrong thing here. By reversing this latest commitment (as would occur with chronological backtracking) the search could proceed, but LDS might go back to somewhere earlier in the search and reverse a completely unrelated commitment. LDS with meets propagation would not get into this situation, which we believe explains why with meets propagation LDS is more effective than chronological backtracking. Chronological backtracking is also able to quickly escape from such deadends.

Algorithm	*CPU Time*	*Meets*	*Backtracks*	*Heuristic*	*Solved*
c + m	15.51	434.37	65.76	160.01	70
c	8.48	0.00	24.15	84.89	74
l + m	3.24	158.22	6.62	74.11	72
l	5.66	0.00	12.08	256.37	73

Table 2: *Mean 10 jobs , 3 stages, 3 resources,* $1.3 \times TLB$

Algorithm	*CPU Time*	*Meets*	*Backtracks*	*Heuristic*	*Solved*
c + m	12.18	202.35	8.06	92.84	63
c	17.51	0.00	9.18	117.28	61
l + m	12.83	555.22	8.73	263.51	77
l	30.27	0.00	22.75	1083.83	72

Table 3: *Mean 15 jobs, 3 stages, 3 resources,* $1.3 \times TLB$

Algorithm	*CPU Time*	*Meets*	*Backtracks*	*Heuristic*	*Solved*
c + m	12.55	296.86	0.62	147.34	56
c	41.66	0.00	12.33	219.47	55
l + m	20.26	649.88	2.42	320.03	67
l	100.62	0.00	44.07	3147.73	55

Table 4: *Mean 20 jobs, 3 stages, 3 resources,* $1.3 \times TLB$

Algorithm	*CPU Time*	*Meets*	*Backtracks*	*Heuristic*	*Solved*
c + m	38.97	476.00	1.37	237.13	38
c	114.47	0.00	12.14	307.31	36
l + m	65.57	1610.89	3.67	794.37	54
l	88.00	0.00	10.52	1911.38	29

Table 5: *Mean 25 jobs, 3 stages, 3 resources,* $1.3 \times TLB$

Algorithm	*CPU Time*	*Meets*	*Backtracks*	*Heuristic*	*Solved*
c + m	49.58	697.33	0.21	347.71	42
c	242.90	0.00	19.78	420.33	36
l + m	72.32	1214.19	1.04	603.31	52
l	55.93	0.00	4.26	1154.83	23

Table 6: *Mean 30 jobs, 3 stages, 3 resources,* $1.3 \times TLB$

Algorithm	*CPU Time*	*Meets*	*Backtracks*	*Heuristic*	*Solved*
c + m	8.72	200.43	31.55	95.64	84
l	1.85	0.00	2.94	58.79	82
l + m	1.75	118.57	2.45	54.02	84
l	27.75	0.00	88.06	1308.64	81

Table 7: *10 jobs, 1.5 times TLB*

Algorithm	*CPU Time*	*Meets*	*Backtracks*	*Heuristic*	*Solved*
c + m	4.50	166.74	0.74	83.21	86
c	18.68	0.00	17.82	160.53	88
l + m	3.63	172.64	0.09	85.85	88
l	45.37	0.00	55.73	2001.52	56

Table 8: *15 jobs, 1.5 times TLB*

Algorithm	*CPU Time*	*Meets*	*Backtracks*	*Heuristic*	*Solved*
c + m	14.62	310.34	0.74	154.82	76
c	70.46	0.00	19.50	257.51	80
l + m	13.37	441.86	1.05	215.41	85
l	29.44	0.00	14.81	954.88	32

Table 9: *20 jobs, 1.5 times TLB*

Algorithm	*CPU Time*	*Meets*	*Backtracks*	*Heuristic*	*Solved*
c + m	25.46	469.34	0.00	234.63	70
c	239.78	0.00	27.63	394.11	73
l + m	27.99	586.31	0.35	289.47	77
l	38.46	0.00	3.59	716.95	22

Table 10: *25 jobs, 1.5 times TLB*

Algorithm	*CPU Time*	*Meets*	*Backtracks*	*Heuristic*	*Solved*
c + m	54.60	662.96	0.79	331.23	71
c	448.93	0.00	32.48	487.52	61
l + m	48.70	772.08	0.26	384.59	78
l	76.32	0.00	4.31	1217.15	13

Table 11: *30 jobs, 1.5 times TLB*

5. Future work

Many industrial scheduling problems bear little resemblance to the job shop scheduling model, as used as a basis for the work described in this paper. Our next step is to extend the meets constraint propagation scheme to deal with multi-capacity resources and activities with alternate resources.

6. Conclusions

In this paper we have presented a simple but effective constraint propagation mechanism to deduce implied sequencing of activities in scheduling problems with the meets constraint. Such constraints arise frequently in industrial scheduling problems. We have shown significant performance improvement when using this constraint propagation scheme on a set of benchmark problems designed to model a real world industrial problem we are currently tackling.

Acknowledgements

This research was in part funded by the Natural Science and Engineering Research Council of Canada, Numetrix Limited, the IRIS Research Network, the Manufacturing Research Corporation of Ontario and Digital Equipment of Canada.

We would like to thank Ed Sitarski and Rob Morenz for their valuable comments on this paper.