

## Scheduling Alternative Activities

**J. Christopher Beck**

ILOG, S.A.  
9, rue de Verdun, B.P. 85  
F-94253 Gentilly Cedex FRANCE  
cbeck@ilog.fr

**Mark S. Fox**

Department of Mechanical and Industrial Engineering  
University of Toronto  
Toronto, Ontario, CANADA, M5S 3G9  
msf@eil.utoronto.ca

### Abstract

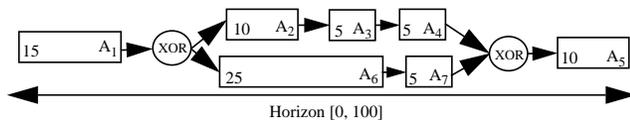
In realistic scheduling problems, there may be choices among resources or among process plans. We formulate a constraint-based representation of alternative activities to model problems containing such choices. We extend existing constraint-directed scheduling heuristic commitment techniques and propagators to reason directly about the fact that an activity does not necessarily have to exist in a final schedule. Experimental results show that an algorithm using a novel texture-based heuristic commitment technique together with extended edge-finding propagators achieves the best overall performance of the techniques tested.

### Introduction

Scheduling problems addressed in the constraint-directed scheduling literature typically have a static activity definition: each activity *must* be scheduled on its specified resource(s). It is common, however, in real-world scheduling problems to have a wider space of choices. There may be multiple process plans for the production of an inventory, or an activity may have a number of resources on which it could execute. In this paper, we show how both these alternatives (*alternative process plans* in the former case and *alternative resources* in the latter) can be addressed by explicit representation of and reasoning about *alternative activities*.

A simple example is shown in Figure 1. The activity network represents a choice of paths from activity  $A_1$  to activity  $A_5$ : the intervening activities may be  $A_2$ ,  $A_3$ , and  $A_4$ , or  $A_6$  and  $A_7$ . We describe the “XOR” nodes in detail in the body of this paper. For now it is sufficient to understand that they represent alternative paths in the activity network and, in a final solution, only one of the alternative paths can be present. Part of the scheduling problem, then, is to decide if  $A_2$ ,  $A_3$ , and  $A_4$  will execute, or if  $A_6$  and  $A_7$  will execute.

The literature provides two general approaches to alternative activities. The first (Nuijten, 1994; Le Pape, 1994;



**Figure 1. A Process Plan with Alternatives. The duration of each activity is shown in the lower left corner of the activity.**

Baptiste and Le Pape, 1995) represents alternatives with a variable in the constraint representation. While much work within this approach depends on propagation techniques to significantly prune the search space, to our knowledge, none of the sophisticated propagators developed over the past few years have been extended to directly reason about activities that may not exist in a solution. The second approach is that of multiple alternative decomposition (MAD) where the alternatives are not represented as internal variables, but rather as separate activities and process plans (Saks et al., 1993; Kott and Saks, 1998). While texture-based heuristics have been applied to such a decomposition, propagators and backtracking have not yet been integrated with the overall heuristic approach.

Our approach to scheduling alternative activities builds on the literature. We explicitly represent the alternative activities following the MAD approach, and in addition, we employ constraint mechanisms to directly reason about the relationships among activities that may not exist in a solution. This allows us to adapt existing texture-based heuristics and propagation techniques.

### Probability of Existence

As may be anticipated from Figure 1, the addition of new types of temporal nodes is central to our novel techniques. Before addressing those issues, however, we turn to the representation of the fact that an activity may not exist in a final solution to a scheduling problem.

Following the knowledge-based approach to scheduling (Fox, 1983; Fox, 1986), our first task in addressing problems with alternative activities is to represent that an activity may not exist in a final solution. The *probability of existence* (PEX) of an activity, therefore, is the estimated probability at a search state that an activity will exist in a solution. The PEX value of an activity is in the range  $[0, 1]$  with 1 indicating that an activity will certainly exist and 0 indicating that it will certainly not. Returning to Figure 1, neither  $A_1$  nor  $A_5$  have alternatives; they must exist in a solution and so their PEX values are 1. Without additional information, each of the alternative paths between  $A_1$  and  $A_5$  is equally likely. The initial PEX values for all other activities, therefore, is 0.5.

With the addition of PEX variables to each activity, we face two challenges. First, we need to maintain consistent PEX values among activities related by temporal con-

straints. PEX propagation must ensure that if we make a PEX commitment, such as assigning the PEX value of  $A_3$  to 1, the PEX values of the related activities would be appropriately reset. In our example, the PEX variables of  $A_2$  and  $A_4$  should also be set to 1 and the PEX variables of  $A_6$  and  $A_7$  should be set to 0. Second, we need to also maintain a consistent temporal network given that some of the activities may not execute in a solution. Returning to the original state in Figure 1 (*i.e.*, before assigning the PEX of  $A_3$  to 1), standard temporal propagation would derive the start time window of  $A_1$  to be  $[0, 45]$ . It is possible, however, for  $A_1$  to start as late as time 55 if the path through  $A_2$ ,  $A_3$ , and  $A_4$  is chosen. Clearly, we need to modify the temporal propagation to account for the PEX values. In addition, we would like to be able to discover when a particular path is no longer consistent. Assume that the two alternatives in Figure 1 are still possible and that through some other scheduling decision the earliest start time of  $A_1$  is increased to 46. In such a situation, we would like to detect that  $A_6$  and  $A_7$  cannot be consistently executed.

### Limitations on the PEX Implementation

Each activity in the temporal graph has a PEX variable; however, the variable is not a true domain variable in the usual CSP sense. In a solution, all PEX values must be either 0 or 1, while during search each PEX variable should have a single value representing the current estimated probability of existence, rather than a domain of values. In a constructive search, if, in search state,  $S$ , a PEX variable is set to either 0 or 1, that value must be maintained in all search states below  $S$ . If a PEX variable has not been assigned to either 1 or 0, it varies during search on the domain  $[0,1]$  according to the PEX propagation algorithm.

There are a number of limitations that we have placed on the PEX representation in order to simplify the implementation. In particular, we make the following assumptions:

- The only commitments that can be made on a PEX variable are to assign it to either 0 or 1. PEX propagation will reset the PEX values of other activities appropriately.
- Each choice that remains to an alternative is equally likely: we assume that we do not have any external knowledge that biases the alternatives.

### Adding PEX to the Temporal Network

Reasoning with PEX variables is achieved partly via the addition of XorNodes to the temporal network. A XorNode represents an exclusive-or constraint among a set of sub-graphs which may themselves contain XorNodes. If a XorNode is present in the final solution then one and only one of the nodes directly connected upstream (resp. downstream) can be present. If a node directly connected upstream (downstream) to a XorNode exists in a solution, so must the XorNode.

As noted above, for the purposes of this paper, we further limit the XorNode behaviour by specifying that all nodes directly connected upstream (or downstream) must have equal PEX values. If a XorNode has a PEX value of  $x$ , and  $k$

upstream and  $l$  downstream links, the PEX value of each directly connected upstream node must be  $x/k$  and the PEX value of each directly connected downstream node must be  $x/l$ . The only exception to the rule of even division is when the upstream or downstream node has a PEX variable assigned to 0 or 1. If a neighboring (wolog) upstream node already has a PEX value of 0, it is not included in PEX propagation: the PEX value at the XorNode is simply divided among the upstream nodes whose previous PEX were greater than 0. If the neighboring (wolog) upstream node has a PEX value of 1, all the other directly linked upstream nodes must have a value of 0.

### Propagating PEX

The basic PEX propagation is achieved through the behavior at each temporal node. At an activity,  $A$ , all the non-XorNodes directly linked to  $A$  must have the same PEX value as  $A$ . For XorNodes, the sum of the PEX values for all nodes directly connected upstream must be equal to the sum of all nodes directly connected downstream which in turn must be equal to the PEX value of the XorNode.

Initial PEX propagation begins with a temporal network where PEX values are unassigned and consistently assigns the PEX variables. If a node has either no upstream or no downstream links it can not be subject to a XorNode and therefore must be present in the final solution. The initial PEX propagation algorithm begins by assigning a PEX value of 1 to all these activities. Based on a topological ordering of the activities in the graph, found with a depth-first search, these initial values are propagated throughout the graph. The complexity of the initial propagation given  $n$  nodes and  $m$  temporal constraints is  $O(\max(m, n))$ . Space limitations preclude presentation of the full algorithm here. See (Beck, 1999) for more details.

Incremental PEX propagation starts with a consistent network and some change to a PEX value. The change is represented by the addition of a new unary constraint assigning a PEX variable to either 0 or 1. Again, space precludes full presentation of the algorithm; however, an example serves to illustrate the process. Figure 2 presents a temporal network. Assuming  $X_1$  and  $X_7$  both have an initial PEX value of 1 and that, after initial propagation, we add the unary PEX constraint assigning the PEX value of activity  $A_1$  to 1, Table 1 shows the PEX values of a subset of nodes before and after incremental PEX propagation.

As shown in Figure 2, XorNodes can be nested: the alternative represented by  $X_3$  and  $X_4$  is nested within the alternative represented by  $X_2$  and  $X_6$  which is nested within the alternative represented by  $X_1$  and  $X_7$ . The initial step of the PEX propagation is to identify the inner-most XorNodes relevant to the node whose PEX is being assigned. Having identified these nodes ( $X_3$  and  $X_4$ ) we propagate from  $A_1$  upstream to  $X_3$  and downstream to  $X_4$ . This propagation assigns a PEX value of 1 to both  $X_3$  and  $X_4$ . We then propagate downstream along all paths from  $X_3$  to  $X_4$  following the same rules as with initial propagation. In this case, the propagation sets the PEX of  $A_2$  to 0.

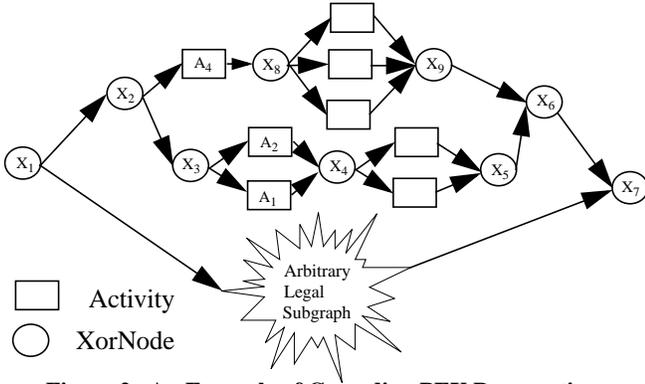


Figure 2. An Example of Cascading PEX Propagation.

The PEX values of  $X_3$  and  $X_4$  have been reset and therefore we continue propagation. This continuation is a *cascade* of PEX propagation. The only difference from the first round of incremental propagation is the nodes that form the starting point. Rather than starting with  $A_1$ , we now start with the inner-most XorNodes from the previous iteration. In our example, we begin with  $X_3$  and identify the inner-most XorNode upstream ( $X_2$ ). Starting from  $X_4$ , we identify the inner-most downstream XorNode ( $X_6$ ). As with the initial iteration, we now reset the PEX values of the identified XorNodes (to 1 in this case) and PEX propagate downstream from  $X_2$  to  $X_6$ . The second cascade of PEX propagation will reassign the PEX value of  $X_5$  (to 1), the PEX values of the activities between  $X_4$  and  $X_5$  (to 0.5), and the PEX values of all nodes along any path from  $X_2$  to  $X_9$  (to 0). One more cascade assigns the PEX values of all nodes in the “Arbitrary Legal Subgraph” to 0.

At worst, PEX propagation will require  $O(n)$  cascades where  $n$  is the number of temporal nodes in the graph, since, in the extreme case, there can be  $O(n)$  nested alternatives. Each cascade incurs a worst-case complexity of  $O(\max(n, m))$ . Therefore, the overall worst-case complexity is  $O(\max(n^2, nm))$ .

## Temporal Propagation with PEX

There are three differences in temporal propagation when XorNodes are present in the network.

Node	Initial PEX Values	PEX Values After Commitment
$X_1$	1.0	1.0
$X_2$	0.5	1.0
$X_3$	0.25	1.0
$A_1$	0.125	1.0
$A_2$	0.125	0
$X_4$	0.25	1.0
$X_5$	0.25	1.0
$X_6$	0.5	1.0
$X_7$	1.0	1.0

Table 1: PEX Values for a Subset of the Nodes in Figure 2.

1. Propagation through a XorNode is different than propagation through an activity.
2. When there are nodes with  $PEX < 1$  in the graph, a state where an activity has an empty domain may not be a dead-end, but may indicate an implied PEX commitment.
3. Temporal propagation is done after PEX propagation.

**Temporal Propagation through a XorNode.** The temporal semantics of a XorNode require that it start at or after the finish time of at least one of its upstream neighbors and it must end at or before the start time of at least one of its downstream neighbors. During downstream propagation, therefore, the XorNode sets the lower-bound on its start time based on the minimum earliest finish time of its upstream neighbors. This start time is then propagated further downstream. During upstream propagation, the analogous process is followed.

A pair of XorNodes representing an alternative enforce an interval of time between themselves. This interval is the length of the shortest path between them (where path length is computed as the sum of the minimum durations of the temporal nodes on the path).

**Deriving Implied PEX Commitments.** In standard temporal propagation, if it is discovered that a variable’s domain has been emptied, a dead-end is derived. When PEX variables are present, emptying a domain may indicate that a particular PEX variable must have a value of 0. When a domain is emptied in temporal propagation, the PEX value of the temporal node with the emptied domain is examined. If the PEX value is less than 1, the node is marked to indicate that the PEX value has been determined to be 0 and temporal propagation does not continue from that node. After temporal propagation, a separate algorithm asserts unary PEX commitments on the marked nodes, and then the usual PEX and temporal propagation is done.

If the domain of a temporal node with PEX of 1 is emptied, a dead-end is derived as in the standard temporal propagation. To see why this is the case, imagine emptying the start time domain of an activity,  $A$ , with a PEX value of 1. By definition, it can not have any enclosing XorNodes. Assume that the presence of some activity,  $B$ , with a PEX value of less than 1 caused the empty domain. The temporal propagation from  $B$  to  $A$  must occur through one of the XorNodes enclosing  $B$ . If that propagation empties  $A$ , then it must be that case that not only  $B$  but *all* its alternatives are inconsistent with  $A$ . Therefore, it is a true dead-end.

**Temporal Propagation after PEX Propagation.** After PEX propagation, temporal propagation must be performed, once for each PEX propagation cascade, to re-establish a temporally consistent network. Returning to our example in Figure 2, recall that after assigning activity  $A_1$  a PEX value of 1, three cascades of PEX propagation were performed:  $X_3$ - $X_4$ ,  $X_2$ - $X_6$ , and  $X_1$ - $X_7$ . Starting from the first cascade, temporal propagation must proceed upstream from the first upstream XorNode ( $X_3$ ) and downstream from the first downstream XorNode ( $X_4$ ). It is then necessary to perform temporal propagation upstream from the second upstream XorNode ( $X_2$ ) and downstream from the

second downstream XorNode ( $X_6$ ) and finally from the third pair of XorNodes ( $X_1$  and  $X_7$ ).

## PEX and Heuristics

With the ability to represent and propagate the fact that some temporal nodes may not be present in a final schedule, it is necessary to extend heuristic search techniques. Here we extend three heuristics to incorporate PEX: SumHeight, CBASlack, and LJRand.

**SumHeightPEX.** The SumHeight heuristic (Beck et al., 1997b) analyzes the constraint graph (via a texture measurement) to identify the critical points at which a commitment should be made. A commitment is then made to attempt to decrease the criticality. The constraint graph analysis begins a probabilistic estimate of the demand of each activity for each resource. The individual demand,  $ID(A, R, t)$ , is (probabilistically) the amount of resource  $R$ , required by activity  $A$ , at time  $t$ . It is found as follows, for all  $est_A \leq t < lft_A$  ( $est_A$ ,  $lft_A$ ,  $dur_A$ , and  $STD_A$  are, respectively, the earliest possible start time, the latest possible start time, the duration, and the domain of possible start times for activity  $A$ ):

$$ID(A, R, t) = \frac{\min(t, lft_A) - \max(t - dur_A + 1, est_A)}{|STD|} \quad (1)$$

The individual demands for each activity on a resource are then summed to give an estimate of the aggregate demand for a resource over time. The most critical resource and time point is, by definition, the one with highest aggregate demand. Given critical resource  $R^*$  and time point  $t^*$ , the SumHeight heuristic identifies the two activities that contribute the most individual demand to  $R^*$  at  $t^*$  that are not already linked by a path of temporal constraints. This activity pair is heuristically sequenced.<sup>1</sup>

PEX values are incorporated into the SumHeight heuristic by modifying the individual demand and by widening the commitments that can be made. The modification to incorporate PEX into the individual demand,  $ID_{PEX}(A, R, t)$  multiplies by  $A_{PEX}$ , the PEX value of activity  $A$ :

$$ID_{PEX}(A, R, t) = A_{PEX} \times ID(A, R, t) \quad (2)$$

This modification has the effect of vertically scaling the individual activity demand curve. If the PEX value of  $A$  is 0.5, the individual demand at any time point  $t$ , is half what the demand would be if the PEX value were equal to 1. This modification fits with our semantic interpretation of individual demand to be the probabilistic demand of an activity at a time point. Because we interpret a PEX value as an activity's probability of existence, an activity that has only a 50% likelihood of existing has half the probabilistic demand of an identical activity that will definitely exist.<sup>2</sup>

Using  $ID_{PEX}$ , the individual demands are calculated and aggregated, and the resource and time point with highest criticality is identified. We, then, identify three activities:

- The activity,  $A$ , with highest individual demand for the  $R^*$  at  $t^*$  with PEX value,  $A_{PEX}$ ,  $0 < A_{PEX} < 1$ .
- The pair of activities,  $B$  and  $C$ , that are not sequenced, with PEX values of 1, and with the highest individual demand for  $R^*$  at  $t^*$ . Assume the individual demand of activity  $B$  at  $t^*$  is greater than or equal to that of  $C$ .

A heuristic commitment is found by comparing the individual demand for  $A$  at  $t^*$  with that of  $B$ . If  $A$  is higher, it is the most critical activity and since it has a possibility of not existing, the heuristic commitment removes it from the schedule by setting its PEX value to 0. If  $B$  is of higher criticality, then it is necessary to sequence  $B$  and  $C$  to reduce criticality. We use the sequencing heuristics presented in (Beck et al., 1997b). If a PEX commitment is retracted via a complete retraction technique, we post its opposite, setting the PEX value of  $A$  to 1. Similarly, if the sequencing commitment is retracted, we post the reverse sequence.

**CBASlackPEX.** The CBASlack heuristic (Smith and Cheng, 1993; Cheng and Smith, 1997) identifies the pair of activities on the same resource that have the minimum biased-slack measurement, and sequences them so as to preserve the maximum amount of slack.

To adapt the CBASlack heuristic, we calculate the biased-slack only for activities with a PEX value greater than 0. The following three conditions then apply to a pair:

1. If both activities have a PEX value of 1, post the sequencing constraint that preserves the most slack.
2. If one activity,  $A$ , has a PEX value of 1 and the other,  $B$ , has a PEX value of less than 1, the greatest amount of slack is preserved by setting the PEX value of  $B$  to 0.
3. If both activities have a PEX value of less than 1, the greatest amount of slack is preserved by setting the PEX value of the activity with the longest duration to 0.

If any commitment is retracted, we post its opposite (the other sequence for case 1, or setting the PEX value to 1 in cases 2 and 3) to guarantee a complete search.

**LJRandPEX.** The LJRand heuristic (Nuijten, 1994) finds the smallest earliest finish time of all the unscheduled activities and identifies the set of activities which can start before this time. One activity in this set is selected randomly and scheduled at its earliest start time. When backtracking, the earliest start time of the activity is updated to the minimum earliest finish time of all other activities on that resource.

Our modification of LJRand to incorporate PEX variables, LJRandPEX, performs the following steps:

1. Find the smallest earliest finish time of all unscheduled activities with PEX greater than 0.
2. Identify the set of activities with PEX values greater than 0 that can start before the minimum earliest finish time.
3. Randomly select an activity,  $A$ , from this set.
4. Assign  $A$  to its earliest start time and assign  $A_{PEX}$  to 1.

The alternative commitment, should backtracking undo the commitment on activity  $A$ , is to update the earliest start

1. See (Beck et al., 1997b) for details of the sequencing heuristics.

2. This is the same modification to individual demand used in the KBLPS scheduler to incorporate a notion similar to PEX into texture-based heuristics (Carnegie Group Inc., 1994).

time of  $A$  to the minimum earliest finish time of all other activities with  $PEX > 0$  on the same resource as  $A$ . The alternative does not contain a PEX commitment: subsequent commitments can still assign  $A_{PEX}$  to either 1 or 0.

### The Information Content of the Heuristics

SumHeightPEX uses the actual PEX value while LJRandPEX and CBASlackPEX only use the PEX variable as a three-value variable: 0, 1, or neither-0-nor-1. We expect that because the texture-based heuristics take into account the information represented by the value of the PEX variable, they will outperform the other heuristics. We do not incorporate the PEX value more deeply into the non-texture heuristics in order to evaluate this expectation.

### PEX and Edge-Finding

Edge-finding exclusion (Nuijten, 1994) and not-first/not-last (Nuijten, 1994; Baptiste and Le Pape, 1996) are powerful propagation techniques. Based on an analysis of the time windows of activities, they are, in some situations, able to deduce constraints implied by the current search state. While the algorithms embody different implication rules and find different implied constraints, they depend on all activities having to be present in a final solution.

Edge-finding exclusion and not-first/not-last can clearly be used with activities with a PEX value of 1. Imagine the situation, however, where all activities on a resource but one,  $A$ , have a PEX value of 1. If a dead-end is found by edge-finding, we can soundly infer that activity  $A$  can not execute and set  $A_{PEX}$  to 0. If edge-finding derives new unary temporal constraints on  $A$ , they can be soundly asserted: if  $A$  is to execute, it must be consistent with the rest of the activities that must execute on the same resource. If edge-finding derives unary temporal constraints on activities other than  $A$ , they must be discarded. It is possible for  $A$  not to execute, so we can not soundly constrain the activities that have PEX values of 1.

This reasoning leads to the PEX-edge-finding algorithm that uses the usual edge-finding algorithms as sub-routines. First, the usual edge-finding is run on all activities with PEX values of 1. Then, for each activity with a PEX value between 0 and 1, its PEX value is temporarily set to 1 and the edge-finding algorithms are run again. Any new constraints are filtered as described above. Given that the standard edge-finding worst-case complexity is  $O(n^2)$ , the PEX-edge-finding worst-case complexity is  $O(n^3)$ . It is possible that this time-complexity can be improved and it is certainly the case that the average time performance can be improved by specializing the code.

### Empirical Evaluation

The empirical evaluation of the PEX techniques focuses on problems with alternative process plans, and problems with both alternative process plans and alternative resources. The PEX techniques presented are applicable, without extension, to both types of problems: alternative resources are treated as a nested alternative within a process plan.<sup>3</sup>

The primary goal of the evaluation is to determine whether using the extra information represented by PEX values results in better heuristic commitments and overall search performance. A second goal is the evaluation of the PEX edge-finding techniques.

The six algorithms used in the experiments are summarized in Table 2. The statistical analysis<sup>4</sup> compares the three heuristics with each other both with and without PEX-edge-finding. We evaluate the use of PEX-edge-finding in three conditions corresponding to each of the heuristics. Each algorithm is run until it finds a solution or until a 20 minute CPU time limit has been reached in which case failure is reported. The machine used is a Sun UltraSparc-IIi, 270 Mhz, 128 M memory, running SunOS 5.6.

### Experiment 1: Varying the Number of Alternatives

We constructed four problem sets with a varying maximum number of alternatives in each process plan. Each problem begins with an underlying 10X10 job shop problem. For a problem with a maximum of  $M$  alternatives per process plan, we generate  $10M$  jobs. These jobs are then transformed into 10 process plans with alternatives. For each job we randomly choose the number of alternatives,  $k$ , uniformly from  $[0, M]$ . We then combine randomly chosen portions of the next  $k$  jobs with our original job to produce a single process plan with  $k$  alternatives. The combination process randomly chooses the to-be-combined portion and the location in the original process plan where the alternative is inserted. Each path between a pair of XorNodes representing an alternative must have the same number of activities and that number must be greater than 1.

Sets of problems with a maximum of 1, 3, 5, and 7 alternatives per process plan were generated. Each set contains

Algorithm	Heuristic	Propagators	Retraction Technique
SumHeight-PropPEX	SumHeight-PEX	All <sup>a</sup>	Chronological Backtracking
SumHeight-PEX	SumHeight-PEX	Non-PEX Propagators <sup>b</sup>	Chronological Backtracking
CBASlack-PropPEX	CBASlack-PEX	All	Chronological Backtracking
CBASlack-PEX	CBASlack-PEX	Non-PEX Propagators	Chronological Backtracking
LJRand-PropPEX	LJRandPEX	All	Chronological Backtracking
LJRandPEX	LJRandPEX	Non-PEX Propagators	Chronological Backtracking

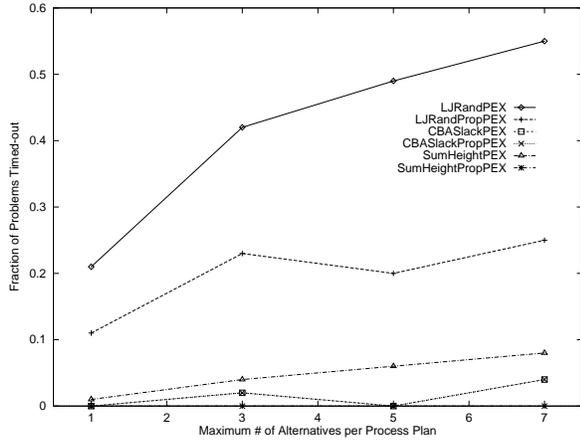
**Table 2: The Six Algorithms Used in the Experiments.**

a. Temporal propagation, PEX edge-finding exclusion, PEX edge-finding not-first/not-last, and CBA.

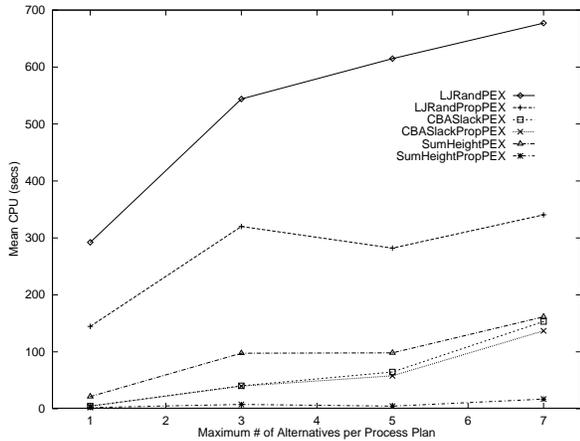
b. Temporal propagation, edge-finding exclusion, edge-finding not-first/not-last, and CBA.

3. See (Davenport et al., 1999) for an application of PEX techniques to pure alternative resource problems.

4. We measure statistical significance with the bootstrap paired-t test (Cohen, 1995) with  $p \leq 0.0001$  (unless otherwise noted).



**Figure 3. The Fraction of Problems in Each Problem Set for which Each Algorithm Timed-out.**



**Figure 4. The Mean CPU Time in Seconds for Each Problem Set.**

120 problems. The problem generation method results in problems such that a solution has exactly the same number of executing activities as the underlying job shop problem. Prior to scheduling, however, each problem has a different number of activities depending on the random generation. Table 3 shows the characteristics of the problem sets.

The proportion of the problems for which each algorithm times out is shown in Figure 3 while the mean CPU time for each algorithm is displayed in Figure 4. Statistical analysis of the number of timed-out problems indicates that there is no significant difference between SumHeightPEX and CBASlackPEX regardless of the use of PEX-edge-finding.

Problem Set (Maximum Number of Alternatives)	Activities Per Problem		
	Min	Mean	Max
1	116	131.6	156
3	142	171.8	200
5	172	204.7	251
7	167	224.2	280

**Table 3: The Characteristics of the Problems in Experiment 1.**

SumHeightPEX and CBASlackPEX significantly outperform LJRandPEX in both propagation conditions. In terms of the usefulness of PEX-edge-finding, SumHeightPropPEX and LJRandPropPEX time-out on significantly fewer problems than SumHeightPEX and LJRandPEX respectively. There is no significant difference in performance between CBASlackPEX and CBASlackPropPEX.

Turning to mean CPU time, SumHeightPropPEX uses significantly less mean CPU time than CBASlackPropPEX, which in turn uses significantly less than LJRandPropPEX. When PEX-edge-finding is not used, there is no difference between CBASlackPEX and SumHeightPEX while both are significantly better than LJRandPEX. Holding the heuristic component constant, we see that SumHeightPropPEX and LJRandPropPEX both incur a lower mean CPU time than their corresponding non-PEX-edge-finding algorithm, while there is no such difference for CBASlackPEX.

Other search statistics indicate that SumHeightPropPEX makes significantly fewer backtracks, commitments, and heuristic commitments than CBASlackPropPEX which makes significantly fewer backtracks, commitments, and heuristic commitments than LJRandPropPEX. Both SumHeightPEX and CBASlackPEX make significantly fewer backtracks, commitments, and heuristic commitments than LJRandPEX. The only significant difference between SumHeightPEX and CBASlackPEX is that the former makes fewer commitments ( $p \leq 0.005$ ). With each heuristic, the use of PEX-edge-finding results in significantly fewer backtracks and heuristic commitments ( $p \leq 0.005$  when CBASlackPEX is the heuristic). In terms of total commitments, LJRandPropPEX is not significantly different from LJRandPEX, while the difference is significant for the other two heuristic commitment techniques ( $p \leq 0.005$  for CBASlackPropPEX versus CBASlackPEX).

## Experiment 2: Adding Alternative Resources

The problems in this experiment are transformations of those in Experiment 1. Alternative resources are added to each activity by randomly generating the total number of resource alternatives following the distribution shown in Table 4. The original resource requirement and duration are preserved in the new problem. In addition, the new resource alternatives (if any) are randomly chosen with uniform probability from among the other resources in the problem. The duration of the activity on each new alternative

Alternative Resources per Activity	Probability
1	0.03125
2	0.5
3	0.25
4	0.125
5	0.0625
6	0.03125

**Table 4: The Distribution of Alternative Resources for the Problems in Experiment 2.**

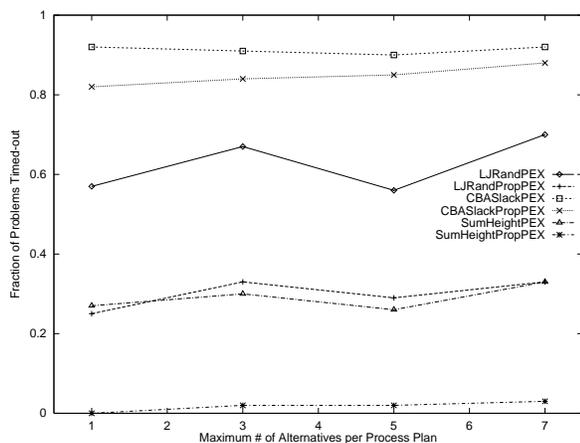


Figure 5. The Fraction of Problems in Each Problem Set for which Each Algorithm Timed-out.

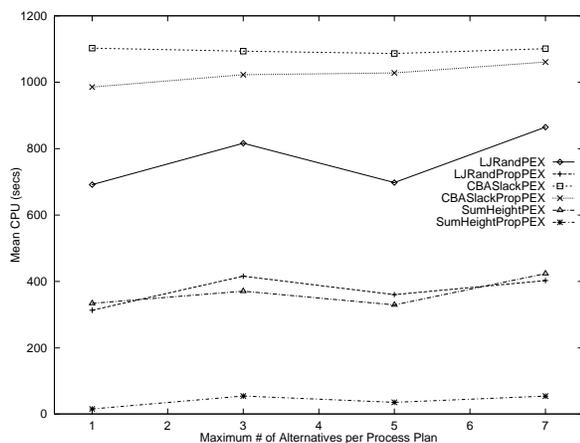


Figure 6. The Mean CPU Time in Seconds for Each Problem Set.

resource is generated by multiplying the activity's original duration by a randomly chosen factor in the domain [1.0, 1.5] and then rounding to the nearest integer value. These transformations result in problems that have widely varying PEX values: theoretically, from less than  $2^{-8}$  to 1. Such a range should favor heuristics that reason explicitly about the PEX value (*i.e.*, SumHeightPEX).

The portion of problems in each set that each algorithm timed-out on is displayed in Figure 5. These results indicate, regardless of the use of PEX-edge-finding, that SumHeightPEX outperforms LJRandPEX which outperforms CBASlackPEX. Each heuristic times-out on significantly fewer problems when using PEX-edge-finding than without it. The mean CPU times are displayed in Figure 6. These results are consistent with the time-out results.

All the other search statistics which were evaluated (number of backtracks, number of commitments, and number of heuristic commitments) agree in the relative ranking of the performance of each heuristic: regardless of the PEX-edge-finding condition, SumHeightPEX outperforms LJRandPEX which outperforms CBASlackPEX. All heuristics exhibited significantly fewer backtracks and heuristic

commitments when used with PEX-edge-finding. Both CBASlackPEX and LJRandPEX made significantly more overall commitments when using PEX-edge-finding.

## Discussion

The experiments present interesting and conflicting results. While SumHeightPEX outperforms CBASlackPEX with PEX-edge-finding, their relative performance without PEX-edge-finding is inconsistent: little difference is observed in Experiment 1, while, in Experiment 2, SumHeightPEX is significantly better. Furthermore, LJRandPEX is outperformed by both of the other heuristics (regardless of the use of PEX-edge-finding) in Experiment 1, but outperforms CBASlackPEX (again regardless of the use of PEX-edge-finding) in Experiment 2.

One explanation for the dramatic difference in the quality of heuristics between the two experiments is the fact that the non-uniformity in PEX values is much greater in the second experiment. SumHeightPEX exploits the non-uniformities among PEX values and so is able to perform better. Another, compatible, explanation for the difference is that the PEX values may be particularly damaging to the ability of CBASlackPEX to identify critical activity pairs. Recall that in the original CBASlack heuristic, the most critical activity pair is one that is not already sequenced (by the CBA propagator or previous heuristic commitments) and that has the smallest biased-slack. Because the CBA propagator cannot be used when one or both members of an activity pair have a PEX value of less than 1 (Beck, 1999), the CBASlackPEX heuristic calculates the biased-slack on such activity pairs even if their time windows do not overlap. Although activities with non-overlapping time windows do not compete with each other for a resource, their biased-slack calculation will tend to be very low. CBASlackPEX, therefore, may focus on such an activity pair even though it is not in any way critical. We hypothesize that such behavior is occurring, and at least contributing to the poor performance of CBASlackPEX. Further research is needed to confirm this behavior and, perhaps, modify CBASlackPEX to avoid it.

Through almost all experiments, PEX-edge-finding was shown to be beneficial to the overall problem solving ability. The only exception is when the CBASlackPEX heuristic is used. In general, these results are as expected. Given the significant increase in the performance of scheduling algorithms with the use of edge-finding propagators (Nuijten, 1994), we expect that some gain is likely with the PEX-edge-finding variation. Our intuitions as to why PEX-edge-finding improves search performance rests on two impacts of propagation. First, propagation techniques reduce the search space by removing alternatives that would otherwise have to be searched through. Second, propagators improve the search information upon which heuristics are based. PEX-edge-finding improves both the information represented in the PEX values and the information represented in the time windows of activities. SumHeightPEX benefits from both improvements while CBASlackPEX does not make direct use of the PEX values in forming its

commitments and so does not benefit from this extra information. Another explanation for the CBASlackPEX result with PEX-edge-finding is that CBASlackPEX commitments tend to result in less propagation (Beck et al., 1997a; Beck, 1999). In Experiment 1, for example, SumHeight-PropPEX makes a significantly smaller percentage of heuristic commitments than CBASlackPropPEX. While a CBASlackPEX algorithm using PEX-edge-finding incurs the computational cost, the benefits are not apparent.

In this paper, we have not investigated the scaling behavior of our representation of alternative activities. While experiments elsewhere (Beck, 1999) indicate that the superiority of SumHeightPEX and PEX-edge-finding techniques continues with larger problems (e.g., alternative process plan problems up to 20X20), the requirement to represent multiple alternatives necessarily leads to poor scaling behavior. This is empirically observed in (Davenport et al., 1999). The trade-off, of course, is that it is the representation and exploitation of the PEX information that results in both the higher quality commitments and the poor scaling behavior. Characterization of this trade-off and effort to optimize it form a central theme of our future research plans.

## Conclusion

In this paper, we introduced the probability of existence (PEX) of an activity and used it to represent that an activity in the original problem definition does not necessarily have to execute in a solution. The use of PEX required extensions to the constraint representation of activities and of the temporal network, including algorithms for the propagation of PEX values among related activities and modifications to the temporal propagation. Heuristic commitment techniques and two edge-finding propagators were also extended to account for PEX values.

Experimental results indicate that incorporating PEX values into the texture-based heuristics results in significantly higher quality commitments and better overall search performance. Performance differences are especially large when there is a wide range of PEX values in a problem. Experimental results also validate the use of PEX-edge-finding which, in most cases, leads to significantly better overall search performance.

A key contribution of this work is the applicability of our representation of alternative activities to a wide variety of real-world scheduling problems. More generally, we have introduced a mechanism to explicitly reason about choosing *not* to take certain actions in order to achieve an overall goal. Such reasoning has relevance in many areas of artificial intelligence. Future research will investigate such applications.

## Acknowledgements

This research was performed while the first author was a Ph.D. student at the Department of Computer Science, University of Toronto. It was funded in part by the Natural Sciences Engineering and Research Council, IRIS Research

Network, Manufacturing Research Corporation of Ontario, Baan Limited, and Digital Equipment of Canada.

Thanks to Andrew Davenport and Angela Glover for discussion of and comments on previous versions of this paper.

## References

- Baptiste, P. and Le Pape, C. (1995). Disjunctive constraints for manufacturing scheduling: Principles and extensions. In *Proceedings of the Third International Conference on Computer Integrated Manufacturing*.
- Baptiste, P. and Le Pape, C. (1996). Edge-finding constraint propagation algorithms for disjunctive and cumulative scheduling. In *Proceedings of the 15th Workshop of the UK Planning and Scheduling Special Interest Group*. Available from <http://www.hds.utc.fr/~baptiste/>.
- Beck, J. C. (1999). *Texture measurements as a basis for heuristic commitment techniques in constraint-directed scheduling*. PhD thesis, University of Toronto. Forthcoming.
- Beck, J. C., Davenport, A. J., Sitariski, E. M., and Fox, M. S. (1997a). Beyond contention: extending texture-based scheduling heuristics. In *Proceedings of AAAI-97*. AAAI Press, Menlo Park, California.
- Beck, J. C., Davenport, A. J., Sitariski, E. M., and Fox, M. S. (1997b). Texture-based heuristics for scheduling revisited. In *Proceedings of AAAI-97*. AAAI Press, Menlo Park, California.
- Carnegie Group Inc. (1994). Knowledge-based logistics planning system. Internal Documentation.
- Cheng, C. C. and Smith, S. F. (1997). Applying constraint satisfaction techniques to job shop scheduling. *Annals of Operations Research, Special Volume on Scheduling: Theory and Practice*, 70:327–378. Forthcoming.
- Cohen, P. R. (1995). *Empirical Methods for Artificial Intelligence*. The MIT Press, Cambridge, Mass.
- Davenport, A. J., Beck, J. C., and Fox, M. S. (1999). An investigation into two approaches for resource allocation and scheduling. Technical report, Enterprise Integration Laboratory, Department of Mechanical and Industrial Engineering, University of Toronto.
- Fox, M. (1986). Observations on the role of constraints in problem solving. In *Proceedings of the Sixth Canadian Conference on Artificial Intelligence*.
- Fox, M. S. (1983). *Constraint-Directed Search: A Case Study of Job-Shop Scheduling*. PhD thesis, Carnegie Mellon University, Intelligent Systems Laboratory, The Robotics Institute, Pittsburgh, PA. CMU-RI-TR-85-7.
- Kott, A. and Saks, V. (1998). A multi-decompositional approach to integration of planning and scheduling – an applied perspective. In *Proceedings of the Workshop on Integrating Planning, Scheduling and Execution in Dynamic and Uncertain Environments*, Pittsburgh, USA.
- Le Pape, C. (1994). Using a constraint-based scheduling library to solve a specific scheduling problem. In *Proceedings of the AAAI-SIGMAN Workshop on Artificial Intelligence Approaches to Modelling and Scheduling Manufacturing Processes*.
- Nuijten, W. P. M. (1994). *Time and resource constrained scheduling: a constraint satisfaction approach*. PhD thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology.
- Saks, V., Johnson, I., and Fox, M. (1993). Distribution planning: A constrained heuristic search approach. In *Proceedings of the Knowledge-based System and Robotics Workshop*, pages 13–19. Industry Canada.
- Smith, S. F. and Cheng, C. C. (1993). Slack-based heuristics for constraint satisfaction scheduling. In *Proceedings AAAI-93*, pages 139–144.