

An Inference Mechanism for Point-Interval Logic

Mashhood Ishaque

Computer Science Department
Tufts University, Medford, MA
mishaq01@cs.tufts.edu

Faisal Mansoor and Abbas K. Zaidi

System Architectures Lab
George Mason University, Fairfax, VA
{fmansoor, szaidi2}@gmu.edu

Abstract

We present a new inference algorithm for Point-Interval Logic. The mechanism removes the incompleteness of previously reported inference mechanism for Point-Interval Logic. We also show how this inference mechanism can be used to prune the search space for an instance in Generalized Point-Interval Logic.

Introduction

Point-Interval Logic (PIL) is a tractable subclass of Allen's interval algebra [1] which is used for modeling temporal information. The temporal information given in the form of a set statements in PIL (conjunction of statements), is converted into graph representation called Point Graph (PG), and checked for consistency (there is a mapping on timeline that satisfies all constraints). Once we have a consistent PG, we can answer temporal queries by executing various graph search algorithms on the PG representation.

The language of PIL has been shown to capture the temporal aspects of time-sensitive mission planning [5, 9, 10], project management [3], and criminal forensics [4]. It is important to have a complete and efficient inference mechanism to effectively solve problems of interest in the mentioned application domains. In this paper we describe a new inference mechanism for PIL, which removes the incompleteness of the previous mechanism reported in [8, 10].

The paper has been organized as follows: in Section 2 we briefly describe PIL and its PG representation, in Section 3 we present the new inference mechanism; in Section 4 we show how to use the inference mechanism to prune the search space for instances in Generalized Point-Interval Logic; and finally in Section 5 we identify future research directions.

Point-Interval Logic and Point Graphs

We begin with a brief description of Point-Interval Logic and Point Graphs for making this presentation self-contained; same description can also be found in [3, 8, 9, 10]. PIL is a formal logic for reasoning with temporal events. It has two types of variables: points (events) and intervals (activities with duration). An interval X implicitly defines two points

sX and eX that represent the start and end of the interval, respectively. The PIL is a pointisable logic [6], i.e. every relation between the temporal variables can be represented in terms of relationships between their start/end points. In Figure 1 we show examples of some temporal relationships between two intervals; for all possible relationships, see [3,9]. PIL also provides constructs to represent quantitative temporal information. In PIL a point variable can be assigned a stamp that represents its occurrence on the timeline. Similarly, an interval can be assigned a length that represents its duration on the timeline.

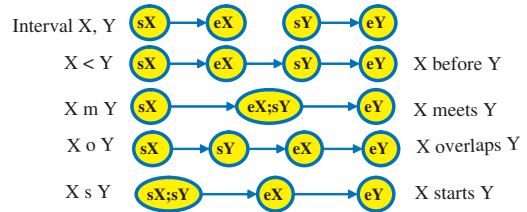


Figure 1: Interval-Interval Relationships in PIL

Point Graph is the knowledge representation scheme for Point-Interval Logic. In a PG, each node represents a point on the timeline, and edges represent the relationship between points. There can be two kinds of edges in a PG: 'less than' (LT) edge and 'less than or equal to' (LE) edge. An LT edge between two nodes p and q depicts that the point represented by the node p occurs on the timeline strictly before the point represented by q . An LE edge, on the other hand, depicts that the point corresponding to node p can either occur before or at the same moment as the point corresponding to q . In Figure 2 we show a set of PIL statements and the corresponding PG representation. In Figure 3 we show the steps involved in constructing a PG from a set of PIL statements.

Inference Mechanism

In this section, we describe the new inference mechanism for PIL. This inference mechanism is implemented in the form of various graph algorithms that operate on the PG representation of a given temporal input. These inference algorithms assume that the temporal information has already been checked for consistency. The resulting inference mechanism improves upon the previous inference mechanism for

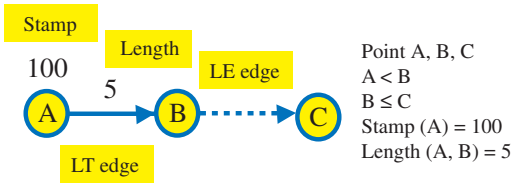


Figure 2: Point Graph for a Set of PIL Statements

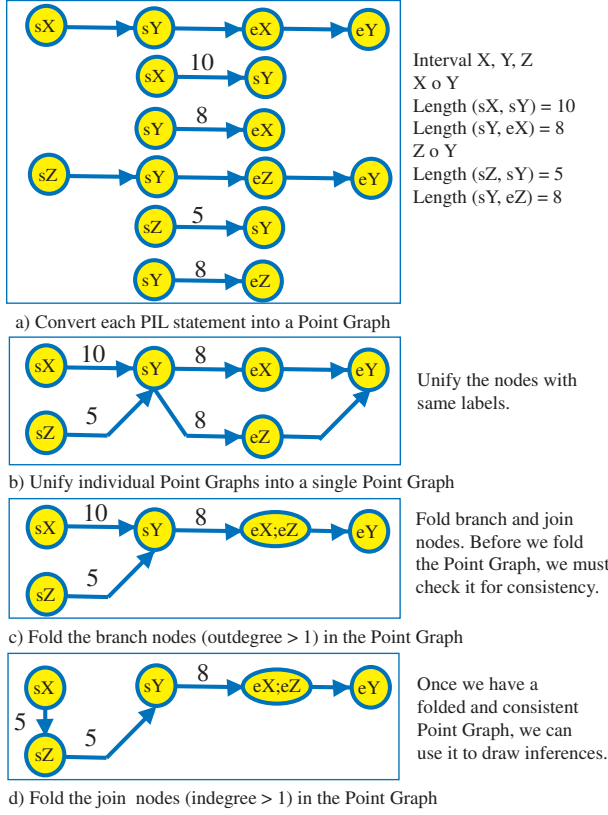


Figure 3: Steps for Constructing a Point Graph

PIL [8,10], which was not complete. The old inference mechanism was based on the idea of finding directed paths between two points in a PG, causing it to miss some temporal relations that can be inferred even in the absence of directed paths. We illustrate the incompleteness in Fig. 4.

The new inference mechanism implements the temporal queries by the following functions; all of which run in $O(m + n)$ time, where n is the number of nodes and m is the number of edges in the PG.

- **queryRelation**(p, q) - returns the relationship between the two temporal variables.
- **queryLength**(p, q) - returns the length of the interval defined by the two points.
- **queryStamp**(p) - returns the time stamp for the point.

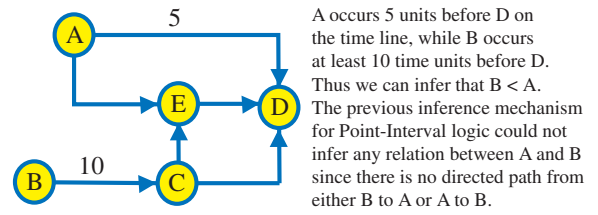


Figure 4: Illustration of Incompleteness

Inferring Relationship Between Two Variables

A relationship query asks for the temporal relationship between two variables (of type point or interval). There are three kinds of relationship queries based on type of the temporal variables.

- Inferring the relationship between two points.
- Inferring the relationship between a point and an interval.
- Inferring the relationship between two intervals.

Since PIL is a pointisable logic[6], all of the above queries can be answered using a constant number of point-point queries. For example, we can infer the relationship between two intervals $X[sX, eX]$ and $Y[sY, eY]$ by looking at the relationships among their end points. Once we have identified the relationships (sX, sY) , (sX, eY) , (eX, sY) and (eX, eY) , we can translate this information into a set of possible relationships between the two intervals using a lookup table. Since there are six possible point-point relationships $\{\leq, <, =, \geq, >, ?\}$, the lookup table for interval-interval relationships has about $6^4 = 1296$ entries. We have computed these lookup tables manually. In Table 1 we show a snippet of the lookup table for interval-interval relationship. The subscript i denotes an inverse relationship; for example, $X f_i Y$ also means $Y f X$.

(sX, sY)	(sX, eY)	(eX, sY)	(eX, eY)	(X, Y)
<	<	<	<	$\{<\}$
<	<	=	<	$\{m\}$
<	<	>	<	$\{o\}$
<	<	>	=	$\{f_i\}$
<	<	>	>	$\{d_i\}$
<	<	>	?	$\{o, f_i\}$
<	<	>	?	$\{d_i, f_i\}$
<	<	>	?	$\{o, d_i, f_i\}$

Table 1: Lookup Table for Interval-Interval Relationships

Since we have already shown that all relationship queries can be reduced to point-point relationship queries, we will only talk about the point-point queries. The incompleteness in the previous inference mechanism resulted from the fact that the mechanism only looked for directed path between two points. But a relationship between two points might still exist even if there is no directed path. We can discover such relationships if we look for the directed path from the two query points to all nodes that are reachable from both points. We not only look for a directed path, but a directed path with the greatest length. Once we have calculated the longest

directed paths from both query points to all reachable nodes, we look at each of those nodes to see if there is sufficient information to infer a relationship. We also calculate the greatest lower bound on the path from p to q if $p < q$ or vice versa, in the process. We can find the longest path between two nodes in a directed acyclic graph using topological sort [2] in $O(m + n)$ time. Note that finding the longest path in an undirected graph is NP-Hard. For the sake of clarity and brevity we only give the version of $\text{queryRelation}(p, q)$ that decide whether $p < q$ or not.

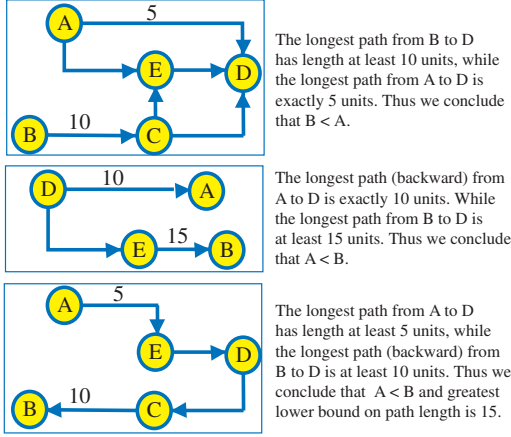


Figure 5: Inferring Relationship Between Points A and B

Since a path can contain edges with or without the length information, comparing two paths needs some explanation. We maintain for each path two parameters: the length of path so far, and a flag to indicate if the path has some edge(s) without the length information. If both the paths that are being compared, contain edges without length information, we cannot decide which path is longer. Otherwise we take the one with greater path length. If the lengths of two paths are equal and one of the paths contains a LT edge without length, then the path with LT edge is longer. While running the topological sort, the longest path is the one with the greatest length regardless of whether it contains any edge without length information. Figure 5 shows examples of how to infer the relationship between two points.

Theorem 1. *Inference mechanism correctly infers the relationship between two points, if it can be inferred at all.*

Proof. Let p and q be the two points between which we want to infer the relationship. We can have only the following four situations involving p and q :

- **p and q are not connected**
In this case we do not have sufficient information to infer any relationship between p and q , and inference mechanism answers that relationship between p and q is unknown.
- **p has directed path to q or vice versa**
We just consider the case when p has a directed path (with at least one LT edge without length information) to q , which implies $p < q$. Since p has directed path to q ,

Algorithm 1 $\text{queryRelation}(p, q)$

Using topological sort find the longest paths from p and q to every reachable node in the directed graph.

Again using topological sort find the longest paths from p and q to every reachable node in the directed graph obtained by reversing the direction of every edge.

Store at each node in the graph the information about the length and direction (forward/reverse) of the longest paths.

$\text{pathLength}_{glb}(p, q) \leftarrow 0$

for all nodes v in the graph **do**

if $\text{forwardPath}(p, v)$ and $\text{backwardPath}(q, v)$ **then**

$\text{relation}(p, q) \leftarrow '<'$

$\text{pathLength}_{lb}(p, q) \leftarrow \text{pathLength}_{glb}(p, v) + \text{pathLength}_{glb}(v, q)$

else if $\text{forwardPath}(p, v)$ and $\text{forwardPath}(q, v)$ **then**

if $\text{pathLength}_{glb}(p, v) > \text{pathLength}_{glb}(q, v)$

then

$\text{relation}(p, q) \leftarrow '<'$

$\text{pathLength}_{lb}(p, q) \leftarrow \text{pathLength}_{glb}(p, v) - \text{pathLength}_{glb}(q, v)$

end if

else if $\text{backwardPath}(p, v)$ and $\text{backwardPath}(q, v)$

then

if $\text{pathLength}_{glb}(v, p) < \text{pathLength}_{glb}(v, q)$

then

$\text{relation}(p, q) \leftarrow '<'$

$\text{pathLength}_{lb}(p, q) \leftarrow \text{pathLength}_{glb}(v, q) - \text{pathLength}_{glb}(v, p)$

end if

end if

if $\text{pathLength}_{lb}(p, q) > \text{pathLength}_{glb}(p, q)$ **then**

$\text{pathLength}_{glb}(p, q) \leftarrow \text{pathLength}_{lb}(p, q)$

end if

end for

the inference mechanism can find a node, q in this case, for which length of path from p to q is greater than that of the (zero length) path from q to q . Thus the mechanism correctly concludes that $p < q$

- **p and q have directed paths to some common node v**
If for all such nodes v both the directed paths contain at least one LT edge without length information, we cannot conclude anything, and neither can the inference mechanism. Suppose w.l.o.g. the path from p has exact length, we can compare it with the greatest lower bound on the length of the path from q . If the length of the path from p is less than the greatest lower bound, then we can conclude that $p > q$ (farther on the timeline). Otherwise the node v does not have enough information for us to conclude anything and we look at other reachable nodes. Thus inference mechanism correctly identifies the relationship between p and q whenever sufficient temporal information is available.
- **some node v has directed paths to both p and q**
Same argument as above.

The claim that the inference mechanism determines the greatest lower bound on the length of the path or exact length if it exists, comes from the ability of topological sort in finding the longest path between two nodes in a directed acyclic graph. \square

Inferring Length of an Interval

A length query asks how far apart are the two query points on the timeline. If the exact length of the interval defined by the two points cannot be inferred, we try to find the tightest lower and upper bounds. The queryLength algorithm given here cannot be used to find the least upper bound on a path's length. For establishing the least upper bound the inference mechanism contains an exponential time algorithm, which is not discussed here. Figure 6 shows an example of how to find the length of a query interval.

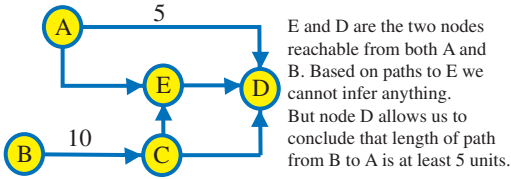


Figure 6: Inferring Length of the Interval [B, A]

Algorithm 2 queryLength (p, q)

```

queryRelation( $p, q$ )
 $Length(p, q) \leftarrow ?, Length_{glb}(p, q) \leftarrow ?$ 
if queryRelation returns  $p < q$  then
     $Length_{glb}(p, q) \leftarrow pathLength_{glb}(p, q)$ 
    if  $pathLength_{glb}(p, q)$  is exact then
         $Length(p, q) \leftarrow Length_{glb}(p, q)$ 
    end if
end if

```

Theorem 2. *Inference mechanism finds the exact length of interval if it can be inferred. Also the algorithm establishes the tightest lower bound for the interval length.*

Proof. The proof follows from the correctness of queryRelation algorithm. \square

Inferring Stamp of a Point

A stamp query asks where the query point (event) occurs on the timeline. If the exact stamp of the given point cannot be inferred, we answer the query is in the form of tightest lower and upper bounds. Figure 7 shows an example of how to find the stamp of a query point.

Theorem 3. *Inference mechanism is complete for stamp queries i.e. if the time stamp of a point can be inferred, the queryStamp algorithm find that time stamp. Also the algorithm establishes the tightest lower and upper bounds for time stamp of a point.*

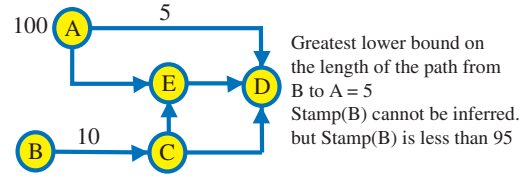


Figure 7: Inferring Stamp of the Point B

Algorithm 3 queryStamp (p)

```

Find the nodes  $s$  and  $t$  with smallest and largest time stamp respectively using breadth first search.
 $Stamp(p) \leftarrow ?, Stamp_{glb}(p) \leftarrow ?, Stamp_{lub}(p) \leftarrow ?$ 
queryRelation( $p, s$ )
if queryRelation returns  $p > s$  then
     $Stamp_{glb}(p) \leftarrow Stamp(s) + pathLength_{glb}(s, p)$ 
    if  $pathLength_{glb}(s, p)$  is exact then
         $Stamp(p) \leftarrow Stamp_{glb}(p)$ 
    return
end if
else
     $Stamp_{glb}(p) \leftarrow Stamp(s) + pathLength_{glb}(s, p)$ 
    return
end if
queryRelation( $p, t$ )
if queryRelation returns  $p < t$  then
     $Stamp_{lub}(p) \leftarrow Stamp(t) - pathLength_{glb}(p, t)$ 
end if

```

Proof. A node corresponding to a query can have an exact time stamp iff there is a directed path of exact length between itself and some node with a time stamp or it has a stamp itself. Since in a consistent and folded PG all the nodes with stamps form a directed path with exact length, a query node can have an exact stamp iff there is a directed path of exact length between itself and the node with the smallest time stamp. Since we have already established the correctness of queryRelation in finding an exact length path between two nodes whenever it exists, we can conclude that queryStamp always find the exact stamp of a node whenever possible.

We now prove that the algorithm finds the tightest lower and upper bounds. Let s and t be the nodes with the smallest and largest time stamps respectively. If the query point occurs before s , we cannot infer any lower bound on the time stamp (unless we can infer an exact stamp in which we are not interested in lower and upper bounds). Similarly no upper bound can be inferred for a point that occurs after t . Thus we find the lower bound on the time stamp of the point that occurs after s by adding the greatest lower bound on the length of the path from s to query point and the time stamp of s . The correctness follows from the fact that the queryRelation always correctly discovers the greatest lower bound on the length of a path. Similar argument applies to the calculation of least upper bound on the time stamp. \square

Generalized Point-Interval Logic

PIL is a tractable subclass of Allen's Interval Algebra. But the tractability comes at the cost of expressiveness. For example, the disjointness relation between two intervals cannot be expressed in PIL. To overcome this deficiency we can generalize PIL by allowing statements in logic to contain disjunctions between relations; we refer to this logic as Generalized Point-Interval Logic (GPIL). Table 2 shows an instance of GPIL. Deciding consistency in GPIL is NP-Complete, since GPIL subsumes Allen's Interval Algebra [1] (hence NP-Hard), and solution to a GPIL instance can be verified in polynomial time (hence NP).

Interval A, B, C
(A m B) or (A o B) or (A d B)
(C < B) or (C > B) (disjointness constraint)
(C s A) or (C f A)
(A < D) or (A > D)

Table 2: An Instance of Generalized Point-Interval Logic

CMI Algorithm

Assuming $P \neq NP$ we cannot expect to find a polynomial time algorithm to decide whether the given instance of GPIL is consistent. Instead we suggest a modified branch-and-bound algorithm called "Consistency Maintaining Inference-Based (CMI) Algorithm" to explore the search space (takes exponential time in the worst case). The CMI algorithm uses the inference mechanism of PIL, as the bounding function to prune the search space. Given a set of statements in GPIL, the CMI algorithm incrementally adds one PIL statement for each GPIL statement while always maintaining a consistent (hence the name) set of PIL statements.

Suppose we are given a set $S = \{s_1, s_2, \dots, s_n\}$ of GPIL statements, and the CMI algorithm is about to process the GPIL statement s_{i+1} . We have a consistent set of PIL statements C_i corresponding to statements from s_1 to s_i . Suppose the statement s_{i+1} is of the form (Xr_1Y) or (Xr_2Y) or ... (Xr_kY) where X and Y are temporal variables and $\{r_1, r_2, \dots, r_k\}$ are relations. The CMI algorithm queries the inference mechanism on C_i for the set of possible relationships between X and Y . The intersection of the set of possible relations from inference engine with the set $\{r_1, r_2, \dots, r_k\}$ gives the set of relations that can be added to C_i while still maintaining consistency. We can pick any relation say r_j from this intersection set and add the PIL statement Xr_jY to C_i to get C_{i+1} . If we can proceed in this manner all the way to C_n it implies the GPIL instance is consistent and C_n is a satisfying assignment.

However, if the intersection set is empty we cannot proceed and we need to backtrack to statement s_i . We delete the last PIL statement added to C_i , and then add the PIL statement corresponding to next relation from the intersection set of the previous stage. If the intersection set at the previous have been completely explored we backtrack even further. If we cannot backtrack any further then the given GPIL instance is inconsistent. It should be pointed out that revising (adding/deleting) a set of PIL statements is very efficient [7].

Algorithm 4 CMI (S, i)

```

{The algorithm is invoked by calling CMI ( $S, 1$ ), and re-
turns true if the instance is satisfiable}
( $X, Y$ )  $\leftarrow$  Variables in  $s_i$ 
 $R_1 \leftarrow$  Relations in  $s_i$ 
 $R_2 \leftarrow$  queryRelation ( $X, Y$ )
sort ( $R_1 \cap R_2$ )
{sort according to desired heuristics}
for all relation  $r_j \in (R_1 \cap R_2)$  do
  addStatement( $X, r_j, Y$ )
  if  $i = n$  then
    return true
  else if CMI ( $S, i + 1$ ) then
    return true
  else
    deleteStatement ( $X, r_j, Y$ )
  end if
end for
return false

```

Figure 8 shows the search space for instance in Table 2. The green nodes are those for which the set of relations corresponding to the path from the root form a consistent set. A path from root to a green leaf node represent a satisfying assignment. The yellow (double-circles) nodes are the first nodes on any path from root that makes the path inconsistent. CMI algorithm is forced to backtrack upon reaching a yellow node. Notice the red (smaller-circles) nodes are never explored, and thus the inference mechanism prunes the search space.

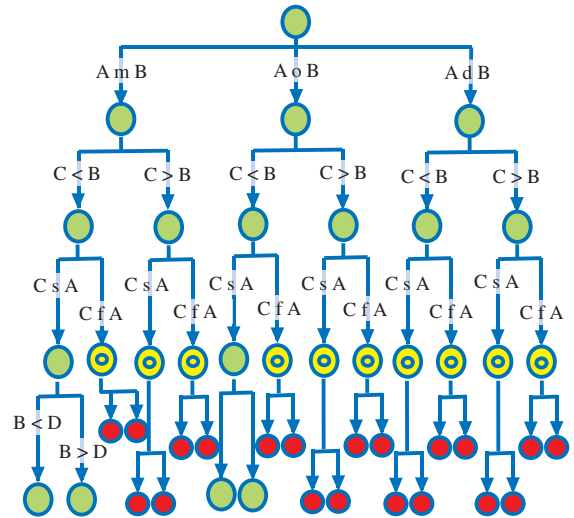


Figure 8: Search Space Exploration

Incorporating Heuristics in CMI Algorithm

CMI algorithm only explores the pruned search space (green and yellow nodes). The order in which these reachable nodes are explored can be controlled by plugging-in any

desired heuristics in CMI algorithm. A heuristics can either determine the order in which the statements of S are processed (inter-statement), or the order in which relations within an statement are added (intra-statement). To incorporate an intra-statement heuristics with CMI algorithm, sort the statements in S accordingly and then run the CMI algorithm. To plug-in an intra-statement heuristics in CMI algorithm, change the function used for sorting ($R_1 \cap R_2$). Here we describe two intra-statement heuristics that we used for the experimental evaluation of CMI algorithm.

Succeed-First heuristics selects the least-constrained relation first. A relation is less constrained when it contains less information. For example the relation ' $<$ ' is less constrained than the relation ' m '. The ordering of the relations from least-constrained to most-constrained is given as:

$$\{\leq, <, o, d, m, f, s, =\}$$

Fail-First heuristics selects the most-constrained relation first. The ordering of the relations is just the reverse of succeed-first ordering.

Experimental Evaluation of CMI Algorithm

We evaluated the performance of CMI algorithm on a set of randomly generated but satisfiable instances of GPIL. We compared the number of nodes in the search space explored by the CMI algorithm (with the two heuristics) until it found the first feasible solution, to the number of nodes in the search space. We present the summary of our results in Table 3, where each row is the average of about 20 random instances.

Var./Stat.	Stat.	Succeed-First	Fail-First	Search Space
50	30	1382	5894	$3.8 * 10^7$
50	40	144741	122022	$3.8 * 10^9$
75	30	66864	49277	$1.9 * 10^8$
75	40	254002	1374951	$5.1 * 10^{10}$
100	30	14382	16771	$5.1 * 10^8$
100	40	40	3226	$8.3 * 10^{10}$
125	30	30	30	$7.8 * 10^7$
125	40	40	40	$7.3 * 10^{10}$

Table 3: Number of Nodes Explored by CMI Algorithm

First column in the table represents the number of variables as a percentage of number of statements. Second column show the number of statements. The third and the fourth column represent the number of nodes explored by the Succeed-First and Fail-First heuristics, respectively. Notice that Succeed-First usually performs better than Fail-First. Also notice as we increase the ratio of number of variables to number of statements in a random instance, it becomes easier to find a feasible solution, which is intuitive since a less-constrained instance has more feasible solutions. The last column represent the total number of nodes in the search space.

Conclusions and Future Directions

In this paper we described a new inference algorithm for Point-Interval Logic. Since language of PIL has already

been shown to have applications in mission planning, project management, and criminal forensics, it is important to have a complete and efficient inference mechanism for it. The presented inference mechanism is complete, though the algorithm to establish the least upper bound on the length of an interval has exponential time complexity. So one obvious open problem is to find a polynomial time algorithm or show that finding the least upper bound is NP-Hard. The other direction of work would be to introduce new queries that are built on the top of the basic relationship, length and stamp queries, and to explore how various data structural techniques can be used to speed up the query algorithms at the cost of additional preprocessing. Also it would be interesting to explore the performance of CMI algorithm on instances resulting from real-life applications.

Acknowledgments

This work was supported by the Air Force Office of Scientific Research (AFOSR) under Grants FA9550-05-1-0106 and FA9550-05-1-0388.

References

- [1] Allen, J. F. 1983. Maintaining Knowledge About Temporal Intervals. *Communications of ACM* 26, 832-843.
- [2] Cormen, T. H., and C. E. Leiserson, and R. L. Rivest, and C. Stein, 2nd ed. 2001. Introduction to Algorithms. MIT Press and McGraw Hill.
- [3] Ishaque, M; A. K. Zaidi, and A. H. Levis 2007. Project Management Using Point Graphs. In Proceedings of 5th Conference on Systems Engineering Research.
- [4] Ishaque, M, and A. K. Zaidi, and A. H. Levis 2006. On Applying Point-Interval Logic to Criminal Forensics. In Proceedings of Command and Control Research and Technology Symposium, 2006.
- [5] Ishaque, M and A. K. Zaidi 2005. Time-Sensitive Planning Using Point-Interval Logic. In Proceedings of 10th International Command and Control Research and Technology Symposium.
- [6] Ladkin, P. B., and Maddux, R. 1988. On binary constraint networks, Technical Report, KES.U.88.8, Kestrel Institute, Palo Alto, Calif.
- [7] Rauf, I., and Zaidi, A. K. 2002. On Revising Temporal Models of Discrete-Event Systems. In Proceedings of 2002 IEEE International Conference on Systems, Man, and Cybernetics, Hammamat, Tunisia.
- [8] Zaidi, A. K. and A. H. Levis 2001. TEMPER: A Temporal Programmer for Time-sensitive Control of Discrete-event Systems. *IEEE Transaction on SMC*: 31, 6, 485-496.
- [9] Zaidi, A. K. and Lee W. Wagenhals 2006. Planning Temporal Events Using Point Interval Logic. *Special Issue of Mathematical and Computer Modeling*: 43, Elsevier, 1229-1253.
- [10] Zaidi, A. K. 2001. A Temporal Programmer for Time-Sensitive Modeling of Discrete-Event Systems. In Proceedings of IEEE SMC 2000 Meeting, Nashville, TN, 2186-2191.