

Combining Qualitative and Quantitative Temporal Reasoning for Criminal Forensics*

Abbas K. Zaidi, Mashhood Ishaque, and Alexander H. Levis

Abstract The paper presents an application of temporal knowledge representation and reasoning techniques to forensic analysis, especially in answering certain investigative questions relating to time-sensitive information about a criminal or terrorist activity. A brief introduction to a temporal formalism called Point-Interval Logic is presented. A set of qualitative and quantitative temporal facts is taken from the London bombing incident that took place on July 7, 2005, to illustrate the use of temporal reasoning for criminal forensics. The information used in the illustration is gathered through the online news sites. A hypothetical investigation on the information is carried out to identify certain time intervals of potential interest to counter-terrorist investigators. A software tool called Temper that implements Point-Interval Logic is used to run the analysis and reasoning presented in the paper.

1 Introduction

While a sequence of events may unfold linearly in time, information about it comes in segments from different locations at different times, often overlapping and often with small contradictions. This phenomenon occurs at a centralized information gathering node where information can be analyzed and fused both because of the

Abbas K. Zaidi

System Architectures Lab, George Mason University, Fairfax, VA 22030, e-mail: szaidi2@gmu.edu

Mashhood Ishaque

System Architectures Lab, George Mason University, Fairfax, VA 22030, e-mail: ADD-EMAIL

Alexander H. Levis

System Architectures Lab, George Mason University, Fairfax, VA 22030, e-mail: alevis@gmu.edu

* An earlier version of the paper has appeared in the proceedings of Descartes Conference on Mathematical Models in Counterterrorism, Center for Advanced Defense Studies, Washington DC, 2006.

different times sensors make their information available and because of the paths that the data or information takes to reach the centralized node. As the US Department of Defense and the Intelligence Community move towards an information sharing paradigm in which data and information are published by all entities and subscribers can access them, the need to develop algorithms that acknowledge this paradigm and even exploit it gains in importance. In colloquial terms, the question is how quickly can we connect the dots when different dots arrive in random order and at random times. We need to know when we have enough dots (given the information that they carry) to declare that we can make a useful inference – that they have been connected.

This problem has particular significance when one tries to reconstruct the sequence of events that led to an observable effect and, especially, to identify the time interval during which some critical activity has taken place. This can be thought of as forensic analysis of a set of given data. The problem becomes more challenging, if the process is undertaken while pieces of information are arriving and there is a time sensitive aspect to it, i. e., useful inferences need to be made as quickly as possible.

Consider, for example, information regarding events surrounding some criminal activity or an act of terrorism to be unfolding in no specific order. The information gathered at some instant in time, in turn, may be incomplete, partially specified, and possibly inaccurate, or inconsistent, making it difficult for investigators and counter-terrorism experts to piece together the events that can help resolve some of the investigative questions. In situations where the gathered information contains qualitative temporal references (e. g., Event A occurs during Event B) to some events of interest, the investigators (and tools supporting them) are not even equipped with standard representation for a temporal language that can capture these qualitative temporal relations. The time-sensitive information, the information about the timing of events surrounding a criminal/terrorist act, even if provided with qualitative temporal references, may contain hidden patterns or temporal relations that can help identify missing links in an investigation. This calls for a formal, computer-aided approach to such an analysis.

The growing need for a formal logic of time for modeling and analyzing temporal information has led to the emergence of various types of representations and reasoning schemes, extensively reported in the research literature. This paper demonstrates the use of one such formalism, called Point-Interval Logic (PIL), in addressing the problems listed above; how it can be used to create temporal models of situations arising in forensics and help investigators operating in real time answer interesting questions in a timely manner. An earlier student paper by Ishaque et al.[1] first proposed and demonstrated the application of PIL to criminal forensics.

This paper is organized as follows: Section 2 gives a very brief overview of some of the influential work on temporal reasoning and knowledge representation. Section 3 presents an informal description of Point-Interval Logic (PIL). This section gives a description of the logic's syntactic and semantic structure, presents a graphical representation for the temporal statements, and illustrates inference mechanism and algorithm for deciding consistency. The software implementation of the ap-

proach, called Temper, is also discussed in this section. A hypothetical investigation on the information taken from London Bombing of July 07, 2005, is carried out with the help of Temper to identify certain time intervals of potential interest to counter-terrorism experts. It also shows that the approach not only operates on the arriving sequence of data but, as a result of the diagnostics produced by the inference engine, it also identifies in a visual manner the type of information that is needed to disambiguate the inferences.

2 Temporal Knowledge Representation and Reasoning

The earliest attempts at formalizing a time calculus date back to 1941 by [2], and 1955 by [3]. Since then, there have been a number of attempts on issues related to this subject matter, like topology of time, first-order and modal approaches to time, treatments of time for simulating action and language, etc. The development of some of these formalisms has matured enough to attract comparative analysis for the computational aspects of these calculi and their subclasses. A number of researchers have attempted to use temporal reasoning formalisms for planning, plan merging, conditional planning, and planning with uncertainty. Other applications of temporal logics include specification and verification of real-time reactive planners, and specification of temporally-extended goals and search control rules.

As noted by Kautz [4], the work in temporal reasoning can be classified in three general categories: algebraic systems, temporal logics, and logics of action. The work on the temporal reasoning within the framework of constraint satisfaction problems was initiated with the influential Interval Algebra (IA) of [5] (formalized as an algebra by [6]) and was followed by Point Algebra (PA) of [7]. They also proved that the problem of determining consistency in IA is NP-Complete. The nature of temporal variables, i. e., point and/or interval, and different classes of constraints, namely qualitative and/or quantitative constraints, have led to a characterization of different types of temporal formalisms: qualitative point, quantitative point, qualitative interval, quantitative interval, and their combinations. A temporal relation (or constraint) between two variables X_i and X_j is represented by the expression $X_i C_{ij} X_j$, where $C_{ij} = \{r_1, r_2, \dots, r_n\}$; r_i 's are basic relations and the expression translates to $(X_i r_1 X_j) \vee (X_i r_2 X_j) \vee \dots \vee (X_i r_n X_j)$. A relation $r_i \in C_{ij}$ is *feasible* for the time variables if and only if there exists a *solution* holding the relation between the two variables. A *solution* to such a temporal problem consists of finding *consistent* relations among all the variables involved. The notion of consistency is defined with respect to specific semantics for each type of temporal problem and with the help of *transitivity/composition tables*. A computationally less expensive notion of *local-consistency* is employed in most implementations to incrementally achieve global consistency. In most cases, enforcing local consistency can be done in polynomial time. Almost all the temporal formalisms use some form of graph representation, e. g., constraint networks, distance graphs, timegraphs, point graphs, etc., for representing/formulating the temporal problem under consideration. Some

of these graphs are merely used for representing the variables and temporal relations between them, whereas others exploit the graph-theoretic properties to enforce consistency and process queries.

Point-Interval Logic (PIL) is a specialization of Pointisable Algebra [8], which is the first and the simplest tractable subclass identified, containing all the basic temporal relations, [9]. This class is characterized by the fact that the temporal relations in it can be represented by specifying relations between the *start/end* points of intervals. A time interval X is defined with the help of a pair of its starting point sX and end point eX . All allowable temporal relations can be represented as constraints between these start and end points associated with time intervals. Polynomial time algorithms for processing Pointisable algebra have been developed [10], [11]. The work on PIL originated from an earlier attempt on temporal knowledge representation and reasoning by Zaidi [12]. A graph model, called Point Graph (PG), is shown to represent the temporal statements in this approach. An inference engine based on this Point Graph representation infers new temporal relations among system intervals, identifies temporal ambiguities and errors (if present) in the system's specifications, and finally identifies the intervals of interest defined by the user. Zaidi and Levis [13] further extended the point-interval approach by adding provisions for *dates/clock* times and time distances for points and intervals. This extension allowed the assignment of actual lengths to intervals, time *distances* between points, and time stamps to points representing actual time of occurrences, whenever such information is available. A temporal model may change during and/or after the system specification phase. Support for an on-the-fly revision (add, delete, modify) was added to Point Graph formalism in [14]. Zaidi and Wagenhals [15] consolidated the results of the previous work on the logic and its application to the modeling and planning time-sensitive aspects of a mission and extended the approach further. The extension allows for a larger class of temporal systems to be handled by incorporating an enhanced input lexicon, allowing increased flexibility in temporal specifications, providing an improved verification and inference mechanism, and adding a suite of analysis tools.

3 Point-Interval Logic

This section presents a brief introduction to the PIL formalism with the help of illustrative examples. A more technical and detailed description can be found in [15], [16], and/or [13].

3.1 Language and Point Graph Representation

The lexicon of the Point-Interval Logic (PIL) consists of the following primitive symbols:

Points: A point X is represented as $[pX, pX]$ or simply $[pX]$.

Intervals: An interval X is represented as $[sX, eX]$, where sX and eX are the two end points of the interval, denoting the ‘start’ and ‘end’ of the interval, such that $sX < eX$.

Point Relations: These are the relations that can exist between two points. The set of relations R_P is given as:

$$R_P = \{<, =, \leq\} \text{ or } R_P = \{before, equals, precedes\}$$

Interval Relations: These are the atomic relations that can exist between two intervals. The set of relations R_I is given as:

$$R_I = \{<, m, o, s, d, f, =\} \text{ or } R_I = \{before, meets, overlaps, starts, during, finishes, equals\}$$

Point-Interval Relations: These are the atomic relations that can exist between a point and an interval. The set of relations R_{PI} is given as:

$$R_{PI} = \{<, s, d, f\} \text{ or } R_{PI} = \{before, starts, during, finishes\}$$

The symbol $?$ is used to represent an unknown relationship.

Functions: The following two functions are used to represent quantitative information associated with intervals:

The *Interval length function* assigns a non-zero positive real number to a system interval.

$$\text{Length } X = d, \text{ where } X = [sX, eX], d \in \mathfrak{R}^+$$

This function is also used to assign lower and upper bounds to an interval length. The two bounds can also be seen as representing *at least* and *at most* temporal relations.

$$\text{Length } X \geq d, \text{ where } X = [sX, eX], d \in \mathfrak{R}^+ \text{ (} d \text{ is a lower bound on length)}$$

$$\text{Length } X \leq d, \text{ where } X = [sX, eX], d \in \mathfrak{R}^+ \text{ (} d \text{ is an upper bound on length)}$$

The *stamp function* similarly assigns a non-negative real number to a point, or lower and upper bounds to it. The two bounds can also be seen as representing *no later than*, and *no earlier than* temporal relations.

$$\text{Stamp } p = t, \text{ where } t \in \mathfrak{R} \text{ (} \mathfrak{R} = \mathfrak{R}^+ \cup \{0\} \text{)}$$

$$\text{Stamp } p \leq t, t \in \mathfrak{R}$$

$$\text{Stamp } p \geq t, t \in \mathfrak{R}$$

Figure 1 shows the syntactic and semantic structure of PIL expressions. Note that each relationship between intervals or an interval and a point can be constructed with the help of inequalities between their start and end points, and by assigning values to expressions involving these points.

A graph construct called Point Graphs (PG) is used as an underlying structure to represent statements in PIL. In a PG, a node represents a point (or a *composite* point) and an edge between two points represents one of the two temporal relations, *before* and *precedes*, between the two. Two or more points p_i, p_j, \dots, p_n are represented as a composite point $[p_i; p_j; \dots; p_n]$, or a single node in a PG, if all are mapped to a

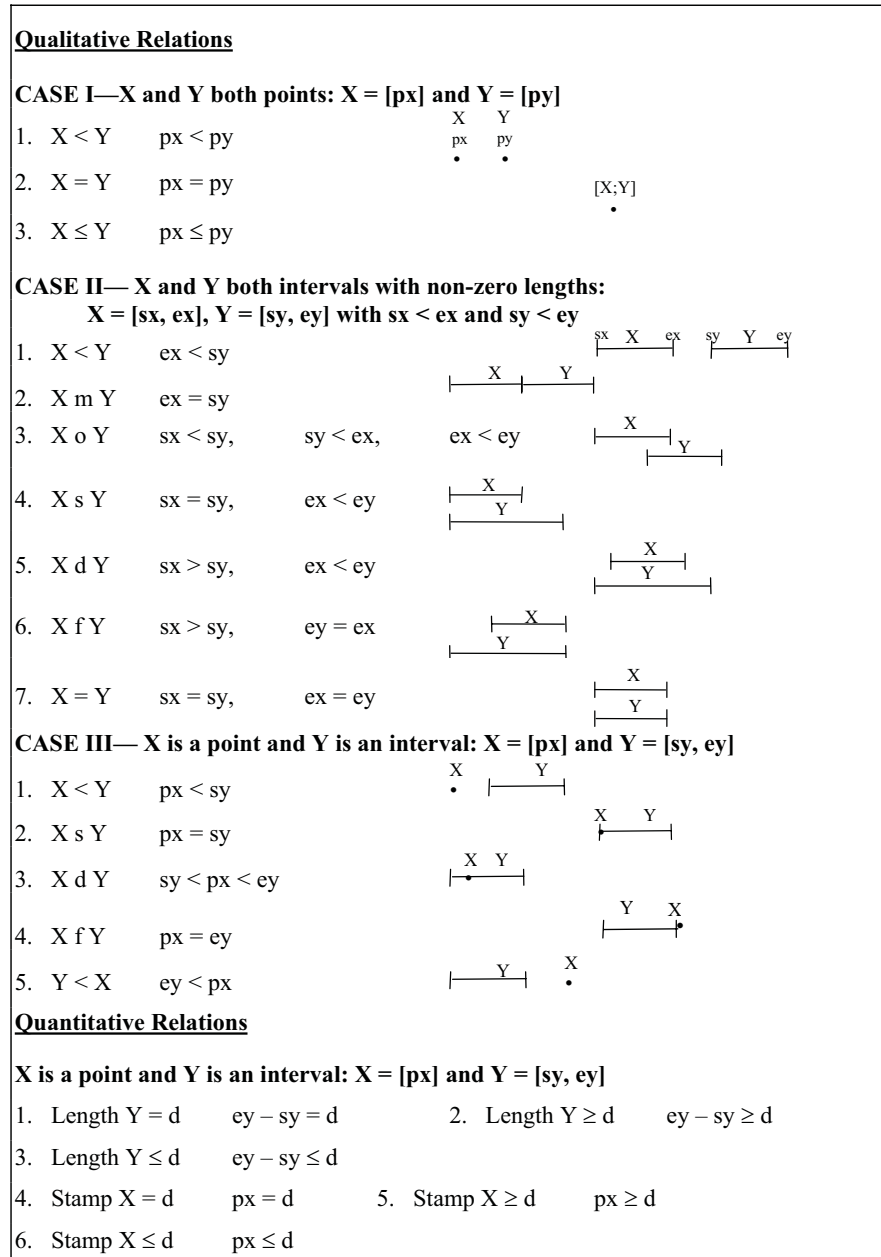


Fig. 1 PIL Expressions and Their Semantics

single point on the timeline. The statements in PIL can be converted to an equivalent PG representation with the help of the corresponding analytic inequalities shown in Fig. 1. In addition, the quantitative temporal information, modeled using the length and the stamp functions, is represented as node and arc inscriptions on the PG. All the verification, revision, and inference algorithms work by manipulating this Point Graph representation of the set of PIL statements. Figure 2 shows a set of PIL statements and the corresponding Point Graph representation.

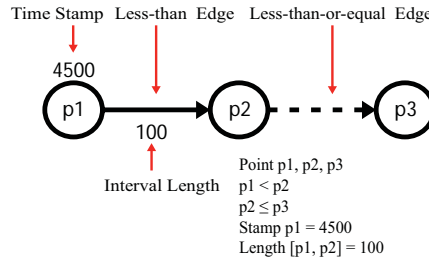


Fig. 2 Point Graph Representation of a Set of PIL Statements

The Point Graph (PG) in Fig. 2 illustrates how the inequalities shown in Fig. 1 for each qualitative temporal relation can be translated to a corresponding equivalent graph structure. The figure also shows graphical objects representing some of the quantitative temporal relations. The graphical representation of the remaining temporal relations requires introduction of virtual time point(s) or *virtual nodes* in a PG. A virtual node is like any other node in a PG except for the fact that there is no temporal variable (point, start of interval, or end of interval) associated with it. It, therefore, does not have a unique identifier or *name* associated with it. Figures 3 and 4 illustrate the PG representations of the quantitative temporal cases not covered by the structure in Fig. 2.

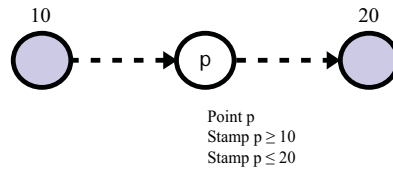


Fig. 3 PG Representation of Lower and Upper Bounds on Stamps

A formal definition of Point Graphs is given as follows:

Definition 1 (Point Graphs). A Point Graph, $PG(V, E_A, D, T)$ is a directed graph with:

- V : Set of vertices with each node or vertex $v \in V$ representing a point on the real number line. Points p_i, p_j, \dots, p_n are represented as a composite point $[p_i; p_j; \dots; p_n]$ if all are mapped to a single point on the line.
- E_A : Union of two sets of edges: $E_A = E \cup E_{\leq}$, where
 - E : Set of edges with each edge $e_{12} \in E$, between two vertices v_1 and v_2 , also denoted as (v_1, v_2) , representing a relation ' $<$ ' (*before*) between the two vertices- $(v_1 < v_2)$. The edges in this set are called LT edges;
 - E_{\leq} : Set of edges with each edge $e_{12} \in E_{\leq}$, between two vertices v_1 and v_2 , also denoted as (v_1, v_2) , representing a relation ' \leq ' (*precedes*) between the two vertices – $(v_1 \leq v_2)$. The edges in this set are called LE edges.
- D : Edge-length function (possibly partial): $E \rightarrow \mathfrak{R}^+$
- T : Vertex-stamp function (possibly partial): $V \rightarrow \mathfrak{R}$

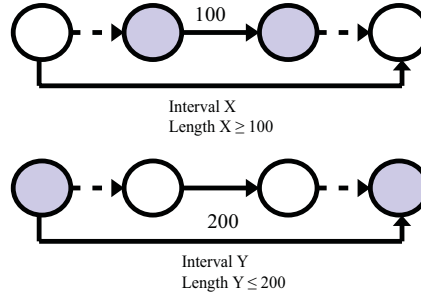


Fig. 4 PG Representation of Lower and Upper Bounds on Interval Lengths

3.2 Operations on Point Graphs

A relation R_i between two intervals X and Y can now be represented by an equivalent Point Graph representation by translating the algebraic inequalities/expressions shown in Fig. 1 to corresponding PGs, as illustrated in Figures 2, 3 and 4. The PG representing a set of PIL statement is then constructed by *unifying* individual PGs to a (possibly) single connected graph. The unifying process looks at the labels of the nodes (except for virtual nodes) and the values of the stamps associated with them to identify equalities. The nodes identified as being equal to one another are merged into a single node with a composite label. The *unified* PG is then *folded* with the help of lengths on edges. This folding process establishes new relations among system intervals, inferred through the quantitative analysis of the known relations specified by interval lengths and stamps. Figure 5 illustrates the two operations, unification and folding, on an example set of PIL statements. A more technical and detailed description of the two processes can be found in [15, 16] or [13].

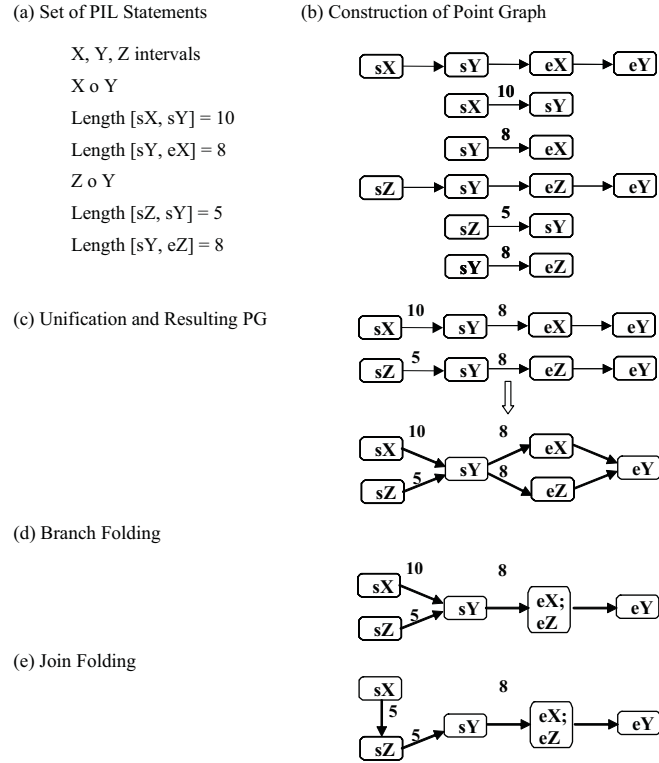


Fig. 5 Steps in PG Construction

3.3 Inference

Two points, $p1$ and $p2$, on a real number line are related to each other by one of the following three algebraic relations: ‘<’ (*before*), ‘=’ (*equals*), and ‘≤’ (*precedes*). A relation R_i between two intervals X and Y , denoted as XR_iY can, therefore, be represented as a 4-symbol string ‘ $abcd$ ’ made of elements from the alphabet $\{<, =, >, \leq, \geq, ?\}$, where the first (left-most) symbol ‘ a ’ represents the algebraic relation between sX and sY , second symbol ‘ b ’ between sX and eY , third symbol ‘ c ’ represents relation between eX and sY , and fourth ‘ d ’ between eX and eY . The ‘?’ is added to incorporate incomplete information. The inequalities in Fig. 1, the definition of an interval, i. e., $X = [sX, eX]$ implies ‘ $sX < eX$ ’, and a set of basic PIL axioms [15] are used to construct this 4-symbol string representation for all possible temporal relations, i. e., both basic and compound relations, in Pointisable logic. A complete list of temporal relations and the corresponding string representations is provided in [15].

The PG representation of PIL statements helps the inference mechanism of PIL to construct the string representation for the pairs of intervals with unknown relations

by performing an undirected search [1, 17] in the PG constructed after unification and folding processes. The algorithms presented in [1, 17] searches for undirected paths between a pair of nodes (points) in a PG by applying a variant of a depth-first search algorithm. This algorithm, while traversing a PG, constructs an expression with the help of quantitative information available on the edges and the nature of the edges, i. e., LT and LE types, to determine the distance between the two nodes (points). An inference for a PIL relation between two intervals requires four such searches to be performed, one for each pair of start/end points. The resulting string representation is pattern-matched with the strings of all possible relations to identify the corresponding atomic/compound PIL relation. The quantitative information from the edges collected by the algorithm helps identify quantitative relations between the points involved.

As an illustration of the inference mechanism of PIL, an inference on the relationship between the two intervals Z and X , in Fig. 5, can be performed as follows: The four searches performed on the last PG in Fig. 5, to construct the 4-symbol string representation, return '>, <, >, =' as the output. This string when pattern-matched with the list of all relations identifies ' ZfX ' (or, Z finishes X) as the inferred relation.

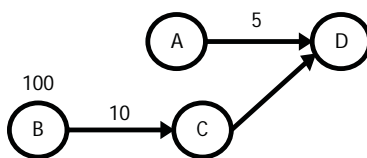


Fig. 6 Inference in a PG

As mentioned earlier, the search algorithm in PIL performs an undirected traversal of a PG to make inferences. The need for an undirected search arises from the presence of quantitative information in PGs, i. e., edge lengths. Figure 6 illustrates a situation where an algorithm that looks for directed paths between a pair of nodes will not be able to infer the temporal relationship between points ' A ' and ' B ', since there is no directed path either from ' A ' to ' B ' or from ' B ' to ' A '. A search from node ' A ' to ' B ', using the algorithm in [16] returns ' $5 - \delta - 10$ ' as the output expression, where δ represents the unknown, non-zero length on LT edge between nodes ' C ' and ' D ', and the negative signs represent the backward orientation of the edges traversed from ' A ' to ' B '. The expression represents the distance from point ' A ' to point ' B ' on a timeline. The expression when evaluated results in a negative value, i. e., ' $5 - \delta - 10 < 0$ ', thus establishing the temporal relationship ' $B < A$ ' (or, B before A).

In addition to the inference algorithm used to infer temporal relations between a pair of points/intervals, a couple of similar algorithms are presented in [17] to calculate time stamps on points and lengths for intervals. The two algorithms try to identify an exact value for the stamp or the length. In case where exact value cannot be ascertained, the algorithms try to calculate an upper and/or lower bounds for the

value. For example in Fig. 6, a query for Stamp [D] is returned by the following: ‘Stamp [D]> 110’.

The time complexity of the inference algorithms in PIL have a time complexity of $O(mn + n^2)$, where m is the number of edges and n is the number of nodes in a PG. The worst-case complexity, therefore, is $O(n^3)$ (in case of a complete graph when m is to the order of n^2) with a much better average performance observed during empirical studies.

3.4 Deciding Consistency

The inference mechanism described above may result in erroneous and inconsistent results provided the system of PIL statements, represented by the PG, contains *inconsistent* information. The inference, on the other hand, is guaranteed to yield valid assertions given a consistent PIL system and corresponding PG representation. The following theorem characterizes inconsistency in PIL.

Theorem 1 (Inconsistency in PIL [15]). *A system’s description in PIL contains inconsistent information iff*

1. *for some intervals X and Y , and atomic PIL relations R_i and R_j , both XR_iY and XR_jY , $i \neq j$, or XR_iY and YR_jX (with the exception of ‘=’ relation) hold true;*
or
2. *for some intervals and/or points, the system can determine two string representations such that at least one pair of the algebraic inequalities representing relationships between the corresponding points represents an inconsistency. Let the two string representations be ‘abcd’ and ‘uvw x ’, where a, b, c, d, u, v, w , and $x \in \{<, =, >, \leq, \geq, ?\}$. One of the (un-ordered) pairs of corresponding inequalities, i. e.,
(a, u), (b, v), (c, w), or (d, x) \in ($<, =$), ($<, >$), ($<, \geq$), ($=, >$), ($>, \leq$);*
or
3. *for a point $p1$, the system calculates two different stamps;*
or
4. *for some points $p1$ and $p2$, ‘ $p1 < p2$ ’, the system can determine two different lengths for the interval $[p1, p2]$.*

The verification mechanism of PG representation identifies these inconsistent cases by using a path-searching algorithm, [18]. The path-searching algorithm employs techniques by [19] and Warshall’s algorithm [20] to identify the erroneous cases. The inconsistencies identified in the theorem, above, manifest themselves in one of the two forms in the PG representation: (a) cycles, and (b) multiple paths between a pair of nodes in a PG with infeasible path lengths. An example of this second case is illustrated in Fig. 7. The details of actual algorithms used to decide consistency in a PIL system can be found in [16].

Verification is the most costly step of the Point Graph construction process. It uses Warshall’s algorithm [20] for searching paths and has time complexity of

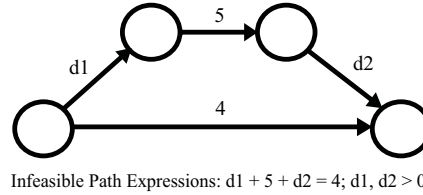


Fig. 7 An Inconsistency in a PG

$O(n^3)$, where n is the number of nodes in a Point Graph. It must be noted that the $O(n^3)$ time complexity is only for detecting cycles and inconsistent paths in the Point Graph. Reporting all cycles or inconsistent paths can have an exponential time complexity, since there can be an exponential number of paths or cycles in a Point Graph, which may not be the case for most real-world PIL systems under study.

The total time complexity of Point Graph construction is $O(n^3)$, where $O(n^3)$ is due to the verification mechanism. But if it is known a priori that the set of PIL statements is consistent, the verification step can be avoided and the Point Graph can be constructed fairly quickly. This and several other techniques have been employed in the software implementation of the approach, called *Temper*, to make representation and reasoning algorithms more efficient. A brief introduction of *Temper* follows in the next subsection.

3.5 *Temper*

A software tool called *Temper* (**Temporal Programmer**) implements the inference mechanism of Point-Interval Logic along with its verification and revision mechanisms. A screen shot of *Temper*'s user interface is shown in Fig. 8. *Temper* provides a language editor, shown in Fig. 9, to input PIL statements and a query editor, shown in Fig. 10, to run various queries on the constructed Point Graphs. It has a graphical interface to display the Point Graphs and also a text I/O interface to display information and results of the analysis (Fig. 8). In the PG shown in Fig. 8, each point is represented as a node, and each interval is represented by two nodes connected by a LT or LE edge. Each LT edge is represented by a solid arc and the length, if available, appears adjacent to the arc. Each less-than-or-equal (\leq) or LE edge is represented by a dotted arc. The stamp on each point appears inside the node representing the point. A special type of node, called virtual node, is used to represent *at least*, *at most*, *no later than*, or *no earlier than* temporal relations. The text callouts in Fig. 8 identify the various elements of a Point Graph as displayed in *Temper*.

The implementation of PIL is in the form of a .NET class library called *PIL Engine* and provides an application programming interface (API) that can be used in any .NET compliant programming language. The tool *Temper* has been built using this API. It provides a graphical user interface to *PIL Engine*. It uses *Quick-Graph*, which is an open-source C# implementation of *Boost Graph Library* [21],

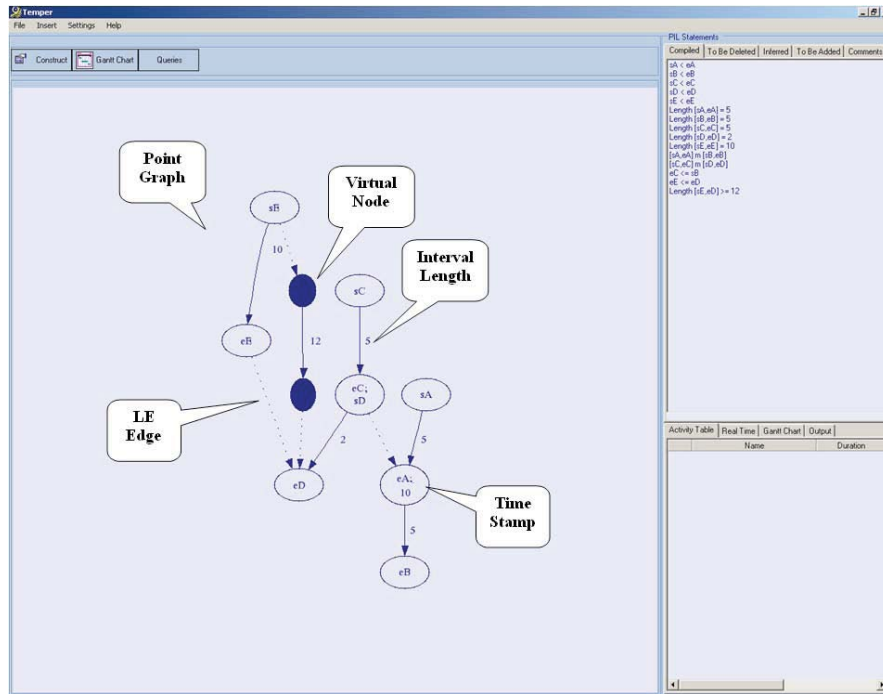


Fig. 8 Temper User Interface with a PG Visualization

The screenshot shows the 'Add/Delete PIL Statements' dialog box. It features several options and input fields:

- Radio buttons for 'Numeric Stamp' and 'Real Time'.
- Checkboxes for 'Day', 'Hour', 'Minute', and 'Second'.
- A 'Reference Date and Time' field set to 'Monday, November 24, 2008'.
- Buttons for 'Add Stamp', 'Delete Stamp', 'Add Length', 'Delete Length', 'Add Relation', 'Delete Relation', and 'Add Composite Relation'.
- Input fields for defining relations, including a dropdown for the relation type (e.g., '<') and a dropdown for the unit (e.g., 'm').
- An 'OK' button at the bottom right.

Fig. 9 Temper's Language Editor

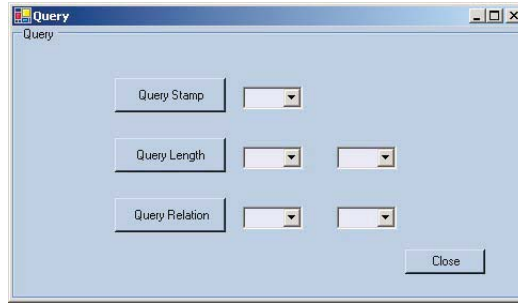


Fig. 10 Temper's Query Editor

and *Graphviz* library from AT&T [22], for internal graph representation and for implementation of PIL algorithms.

The following section presents an application of Point-Interval Logic (PIL) to criminal forensics, especially in answering certain investigative questions relating to time-sensitive information about a terrorist activity. The illustrations are presented with the help of Temper generated outputs to the analysis. A set of temporal facts is taken from the internet on the London bombing incident that took place on July 7, 2005, as input to Temper's reasoning module. A hypothetical investigation on the information is carried out by constructing temporal queries in Temper to identify certain time intervals of potential interest to counter-terrorism investigators.

4 Using Temper for Criminal Forensics – The London Bombing

On July 7, 2005, there were four explosions in London at Tavistock Square, Edgware Road, Aldgate, and Russell Square. Three of these explosions, Edgware Road, Aldgate, and Russell Square, took place in trains that departed from King's Cross station. Images from close-circuit cameras installed at London's various railway stations were an important source of information for investigators. There were hours of images available from these cameras and the task of investigators was to analyze these images to identify possible suspects. The large number of such images, although desirable, can make an investigation that requires searching through them in a timely manner very time consuming. Time pressure was also created because there was need to identify the perpetrators quickly enough to apprehend any ones that survived.

In this section, we demonstrate how temporal reasoning, in general, and especially Temper can be used to restrict the size of a potential interval for which to analyze images (by making sense of the available temporal information) and thus speeding up the investigation. Since Temper has the ability to handle both qualitative and quantitative constraints, both types of information regarding the incident and/or the surrounding events can be input to it. Temper also offers the additional

advantage of the verification mechanism that can be invoked to check the consistency of the available temporal information. This can be very useful when temporal information may originate from multiple (and possibly unreliable) sources. In this example, we demonstrate the capabilities of Temper by modeling a set of temporal information items related to the incident and by trying to identify the exact time or the shortest possible interval during which one of the ill-fated trains left from King's Cross station for Edgware. The graphical visualization of the temporal relations shows clearly where additional data are needed to establish temporal relations that will facilitate developing responses to specific queries.

The journey of the three trains from King's Cross station can be represented as PIL intervals. The journey of these trains ended in explosions. We also know the lower bounds on the travel times of these trains after their departures from King's Cross station, based on the distances of the sites of the explosions from King's Cross station. The train from King's Cross to Edgware must have traveled for at least 5 minutes. Similarly trains to Aldgate and Russell Square must have traveled for 4 and 5 minutes, respectively. Table 1 shows how this information can be represented as PIL statements. These PIL statements are typed or read into Temper using its language editor. Figure 11 shows the corresponding Point Graph in Temper.

Once the temporal information has been inserted, Temper can be used to draw inferences about the event(s) of interest, i. e., the instant when one of the trains left King's Cross station for Edgware. We run a query, using the query editor of Temper, for the time stamp of the point "sTrain_King_Cross_to_Edgware" which represents the departure of the train from King's Cross station to Edgware. Figure 12 shows the query in Temper. The inference algorithm in Temper returns a '?' or 'unknown' as the result of the query, since the temporal information available to the inference mechanism is not enough to draw any meaningful relationship for the temporal event under investigation.

For the illustrated case, Temper cannot infer anything about the stamp of the event based on the information provided so far. This creates the need for information pull. Suppose, further investigation reveals that the explosion near Edgware took place between times 8:40 and 8:52 (the explosion is considered to be an instantaneous event so the range 8:40 to 8:52 does not represent duration but the uncertainty in determining the actual occurrence time). Similarly the explosions near Aldgate and Russell Square occurred between 8:45 and 8:50, and between 8:40 and 8:50 respectively. Table 2 shows how this information can be represented as PIL statements. These PIL statements are added to the initial temporal model to get the Point Graph of Fig. 13. Once again, the query for the time stamp of the point "sTrain_King_Cross_to_Edgware" is executed (Fig. 12). This time, Temper is able to determine an upper bound for the stamp of the event: its inference algorithm returns 'Stamp [sTrain_King_Cross_to_Edgware] \leq 8:47', i. e., the train from King's Cross to Edgware must have left no later than 8:47.

As indicated earlier, the verification mechanism of Temper can detect the inconsistencies in the available temporal information. The ability to detect inconsistency can be very useful, especially when the information from different sources is combined into a single model of a situation under investigation. Suppose, we input to

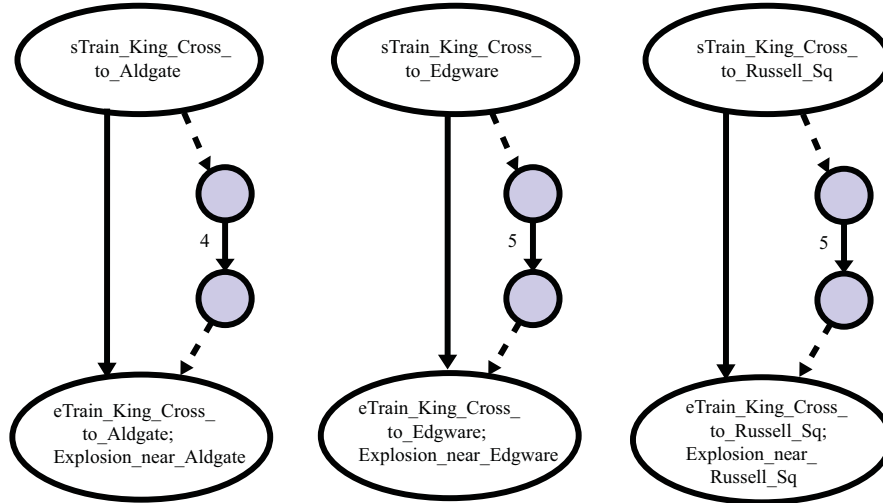


Fig. 11 Point Graph for London Bombing Scenario in Table 1

Table 1 PIL Statements for London Bombing Scenario

Temporal Information	PIL Statements
Train traveling from King's Cross to Edgware	<i>Interval</i> Train.KingX.Edgware
Train traveling from King's Cross to Aldgate	<i>Interval</i> Train.KingX.Aldgate
Train traveling from King's Cross to Russell Square	<i>Interval</i> Train.KingX.Russell.Sq
Explosion at Edgware	<i>Point</i> Explosion.Edgware
Explosion at Aldgate	<i>Point</i> Explosion.Aldgate
Explosion at Russell Square	<i>Point</i> Explosion.Russell.Sq
Explosion at Edgware ended the journey of train from King's Cross to Edgware	Explosion.Edgware <i>f</i> Train.KingX.Edgware
Explosion at Aldgate ended the journey of train from King's Cross to Aldgate	Explosion.Aldgate <i>f</i> Train.KingX.Aldgate
Explosion at Russell ended the journey of train from King's Cross to Russell Square	Explosion.Russell.Sq <i>f</i> Train.KingX.Russell.Sq
Train from King's Cross to Edgware traveled at least for 5 time units	<i>Length</i> [Train.KingX.Edgware] ≥ 5
Train from King's Cross to Aldgate traveled at least for 4 time units	<i>Length</i> [Train.KingX.Aldgate] ≥ 4
Train from King's Cross to Russell Square traveled at least for 5 time units	<i>Length</i> [Train.KingX.Russell.Sq] ≥ 5

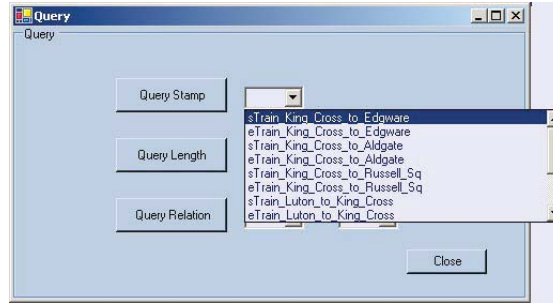


Fig. 12 Running a Query in Temper

Table 2 Additional PIL Statements for London Bombing Scenario

Temporal Information	PIL Statements
Explosion at Edgware happened no earlier than 8:40	$Stamp [Explosion_Edgware] \geq 8:40$
Explosion at Edgware happened no later than 8:52	$Stamp [Explosion_Edgware] \leq 8 : 52$
Explosion at Aldgate happened no earlier than 8 : 45	$Stamp [Explosion_Aldgate] \geq 8 : 45$
Explosion at Aldgate happened no later than 8 : 50	$Stamp [Explosion_Aldgate] \leq 8 : 50$
Explosion at Russel Sq. happened no earlier than 8 : 40	$Stamp [Explosion_Russel_Sq] \geq 8 : 40$
Explosion at Russel Sq. happened no later than 8 : 50	$Stamp [Explosion_Russel_Sq] \leq 8 : 50$

Temper the information that the train from King’s Cross to Edgware left at time 8:48 (represented by the PIL statement: $Stamp[sTrain_King_Cross_to_Edgware] = 8:48$). Clearly, this statement is in conflict with the previously added PIL statements. Temper detects this inconsistency, and identifies the portion of the Point Graph that contains the contradiction (inconsistent paths shown with an extra boundary around each node in the paths in Fig. 14). Note the two inconsistent paths (from node with stamp 8:40 to node with stamp 8:52): one path with length exactly equal to 12 minutes and the other path having a length of at least 13 minutes. We fix this inconsistency by deleting the last statement added, “ $Stamp [sTrain_King_Cross_to_Edgware] = 8:48$ ”. Temper’s revision algorithm employs an efficient use of internal data structures to identify the portion of the Point Graph that is affected by the delete operation and only redraws that affected part to reconstruct the new PG representation. This saves a lot of computational effort required to reconstruct the Point Graph for an entire set of PIL statements from scratch every time there is a need to make modifications in the temporal information. A detailed description of the revision algorithm can be found in [14].

Suppose that the investigators have also identified four suspects who were spotted entering the Luton railway station at time instant 7:20. The investigators believe that these suspects took a train from Luton to King’s Cross station, and at King’s

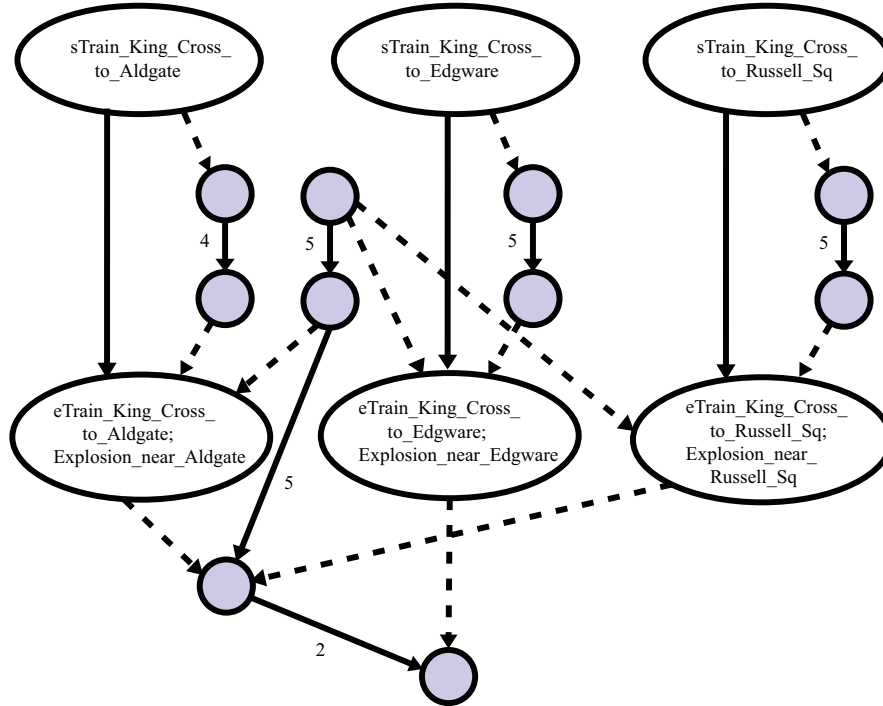


Fig. 13 Revised Point Graph for London Bombing Scenario

Cross station they boarded the trains in which the explosions took place. The next train from Luton to King’s Cross departed at 7:48 and reached King’s Cross at time instant 8:42. Obviously, if these suspects were in fact the bombers, the train from Luton should have reached King’s Cross before any train in which there was an explosion left King’s Cross station. This information can be represented by PIL statements as given in Tab. 3 and the resulting PG is shown in Fig. 15. Note that Tab. 3 contains both qualitative and quantitative PIL statements.

The query for the time stamp of the point “`sTrain_King_Cross_to_Edgware`” is executed once more (Fig. 12). This time, Temper is able to determine both an upper bound and a lower bound for the stamp of the event, its inference algorithm returns: $‘8:42 < \text{Stamp}[\text{Train_King_Cross_to_Edgware}] \leq 8:47’$, i. e., the train must have left King’s Cross station after time instant 8:42 and no later than 8:47. Note that the lower bound is strict. Thus by applying the inference mechanism of Point-Interval Logic to the analysis of available temporal information, the approach has identified the bounds of the interval that we were interested in. The images need to be analyzed for this interval only; this improves the timeliness of the labor intensive image analysis process.

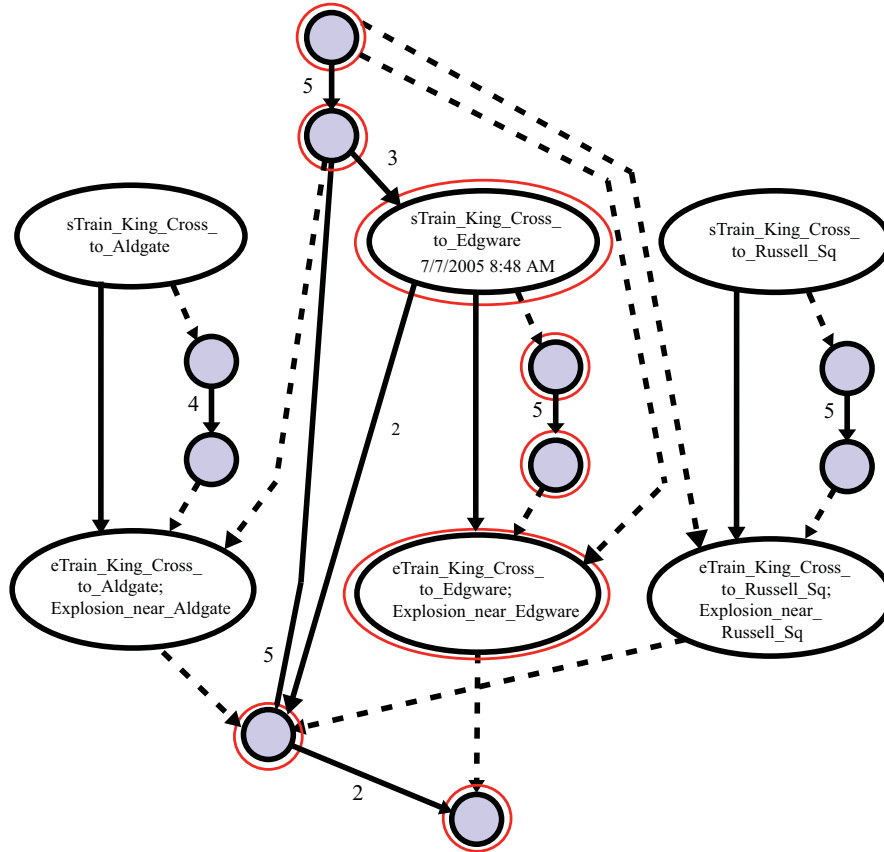


Fig. 14 Inconsistency in the Point Graph

Table 3 Additional PIL Statements for London Bombing Scenario

Temporal Information	PIL Statements
Train traveling from Luton to King's Cross station	<i>Interval</i> Train.Luton.KingX
Suspected spotted entering the Luton station	<i>Point</i> Suspects.Spotted.at.Luton
Suspected spotted at Luton at time instant 7 : 20	<i>Stamp</i> [Suspects.Spotted.at.Luton] = 7 : 20
Train from Luton to King's Cross left at 7 : 48	<i>Stamp</i> [sTrain.Luton.KingX] = 7 : 48
Train from Luton to King's Cross arrived at 8 : 42	<i>Stamp</i> [eTrain.Luton.KingX] = 8 : 42
Train to Edgware left after the train from Luton	eTrain.Luton.KingX < Train.KingX.Edgware
Train to Aldgate left after the train from Luton	eTrain.Luton.KingX < Train.KingX.Aldgate
Train to Russell Sq. left after the train from Luton	eTrain.Luton.KingX < Train.KingX.Russell.Sq

5 Conclusion

This paper presented an illustration of the use of Point-Interval Logic (PIL) for creating temporal models of situations arising in forensics and for helping investigators answer relevant questions in a timely manner. The approach was demonstrated using Temper, which is a software implementation of Point-Interval Logic (PIL), and the London bombing incident as the scenario. The example presented demonstrates how Temper can be used in identifying a small interval for which the images from close-circuit cameras should be analyzed. The reader may argue that the problem could have been solved manually as well; that is true in the case of a small example like the one presented in this paper; however, in general situations the set of temporal statement may be too large for a human to handle. (The example set of PIL statements used in this paper is intentionally kept small for the sake of presenting the ideas and types of analysis that can be performed, instead of presenting the actual solution to the problem posed.) This presence of large set(s) of incomplete, inaccurate, and often inconsistent data calls for a computer-aided approach to such an analysis. Temper can combine temporal information from multiple sources, detect inconsistencies and identify the specific source(s) with inconsistent information. Temper can, therefore, be used to compare witness accounts of several individuals on the same incident for overlaps and inconsistencies—another useful application for forensics.

Acknowledgements

The work was carried out with support provided by the Air Force Office of Scientific Research under contract numbers FA9550-05-1-0106.

References

1. Ishaque, M.; Zaidi, A. K; and Levis, A. H.: On Applying Point-Interval Logic to Criminal Forensics, Proc. Of 2006 Command and Control Research and Technology Symposium (CCRTS). San Diego, CA. (2006)
2. Findlay, J. N.: Time: A treatment of some puzzles, Aust. J. Phil., vol. 19:216–235 (1941)
3. Prior, A. N.: Diodoran modalities, Phil. Quart., vol. 5. 205–213 (1955)
4. Kautz, H.: Temporal Reasoning, In The MIT Encyclopedia of Cognitive Science, MIT Press, Cambridge. (1999)
5. Allen, J. F.: Maintaining Knowledge About Temporal Intervals, Communications of ACM, 26. 832–843 (1983)
6. Ladkin, P. B. and Maddux, R. D.: On binary constraint problems. Journal of the Association for Computing Machinery. 41(3):435–469 (1994)
7. Vilain, M. and Kautz, H.: Constraint-propagation algorithms for temporal reasoning. In Proceedings of the Fifth National Conference on Artificial Intelligence. 377–382 (1986)
8. Ladkin, P. B., and Maddux, R. D.: Representation and reasoning with convex time intervals, Tech. Report KES.U.88.2, Kestrel Institute. (1988)

9. Vilain, M., Kautz, H., and Van Beek, P.: Constraint propagation algorithms for temporal reasoning: a revised report. In *Readings in Qualitative Reasoning about Physical Systems*, San Mateo, CA, Morgan Kaufman. 373–381 (1990)
10. Gerevini, A. and Schubert, L.: Efficient Temporal Reasoning through Timegraphs. In *Proceedings of IJCAI-93*.(1993)
11. Drakengren, T. and Jonsson, P.: Eight Maximal Tractable Subclasses of Allen’s Algebra with Metric Time, *Journal of Artificial Intelligence Research*, 7,25–45 (1997)
12. Zaidi, A. K.: On Temporal Logic Programming Using Petri Nets. *IEEE Transactions on Systems, Man and Cybernetics, Part A*. 29(3):245–254 (1999)
13. Zaidi, A. K., and Levis, A. H.: TEMPER: A Temporal Programmer for Time-sensitive Control of Discrete-event Systems. *IEEE Transaction on Systems, Man, and Cybernetic*. (2001) 31(6):485–496
14. Rauf, I. and Zaidi, A. K.: A Temporal Programmer for Revising Temporal Models of Discrete-Event Systems. *Proc. of 2002 IEEE International Conference on Systems, Man, and Cybernetics, Hammamat, Tunisia*.(2002)
15. Zaidi, A. K., and Wagenhals, L. W.: Planning Temporal Events Using Point-Interval Logic. *Special Issue of Mathematical and Computer Modeling* (43)1229–1. (2006)
16. Ishaque, S. M. M.: On Temporal Planning and Reasoning with Point Interval Logic, MS Thesis, CS, George Mason University, VA. (2006)
17. Ishaque M.; Mansoor F.; and Zaidi A. K. An Inference Mechanism for Point-Interval Logic, *The 21st International FLAIRS Conference*, Association for the Advancement of Artificial Intelligence, Coconut Grove, FL. (2008)
18. Ma, C.: On Planning Time Sensitive Operations, MS Thesis, SE, George Mason University, VA. (1999)
19. Busacker, R. G., and Saaty, T. L.: *Finite Graphs and Networks: an introduction with application*. New York.: McGraw-Hill. (1965)
20. Warshal, I S.: A theorem on boolean matrices, *Journal of the ACM*, 9, 1,11–12. (1962)
21. Boost Graph Library. Information Available at: <http://www.boost.org/>
22. Grahviz. Information Available at: <http://www.graphviz.org/>

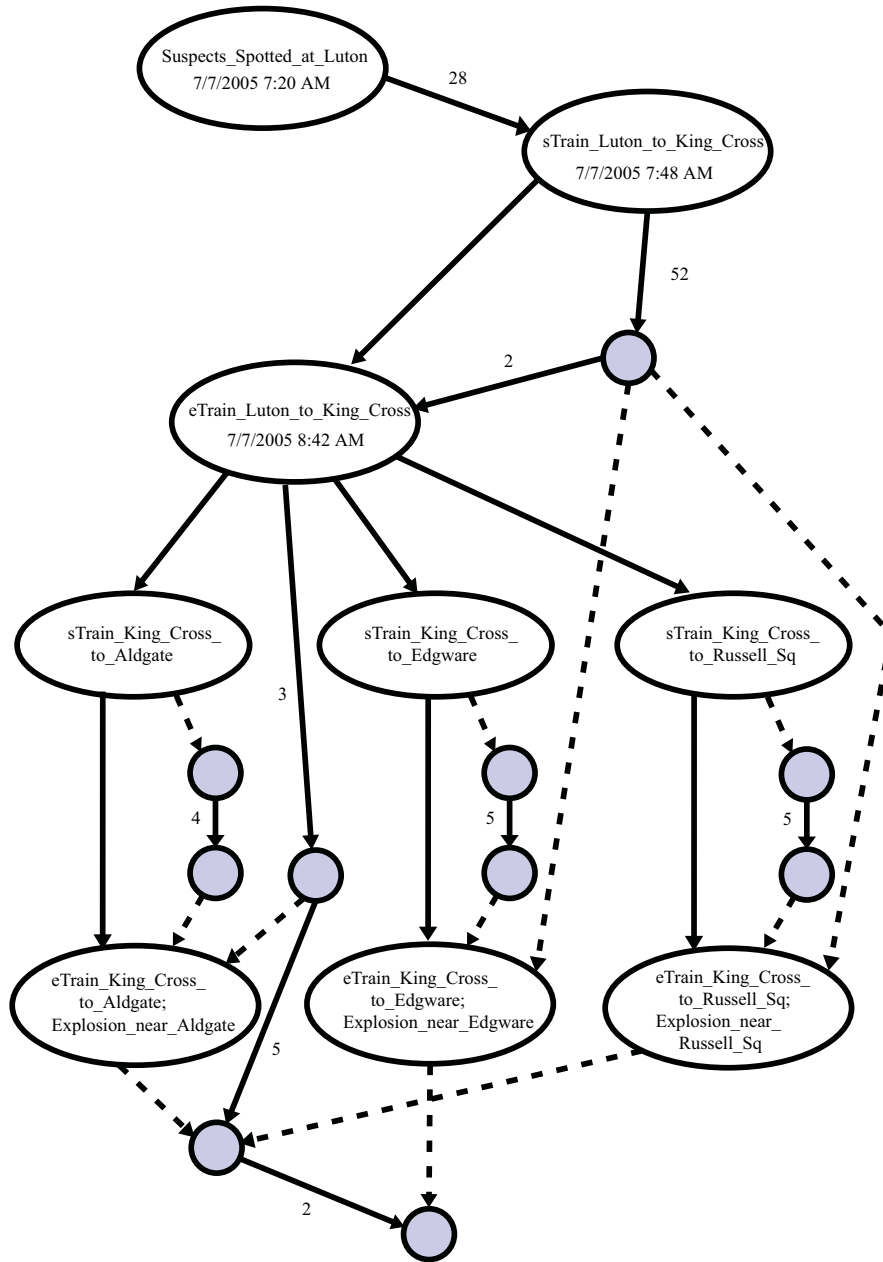


Fig. 15 Revised Point Graph for London Bombing Scenario