Using Temporal Probabilistic Rules to Learn Group Behavior

John P. Dickerson, Gerardo I. Simari, and V.S. Subrahmanian

1 Introduction

The ability to reason about the past, present, or future state of the world is widely applicable to many fields. Additionally, considering uncertainty over the precise time at which events occurred or will occur increases realism, but also increases theoretical and computational intractability. This sort of probabilistic temporal reasoning is important in domains like those listed below.[1]

1. The advent of the Internet has clearly affected traders' reasoning about past and future movements in the **stock market**. For example, Fujiwara et al. [7] and De Choudhury et al. [5] discuss how stories in newspapers, blogs, and miscellaneous websites move prices in the stock market. A variety of data mining and machine learning techniques are used by investment banks and hedge fund managers to predict future stock movements based on past patterns in the values of various indicators. Formally, an investor could learn rules like, "the probability that the

[1] These examples and others are discussed in depth in work by Dekhtyar et al. [6] and Shakarian et al. [18, 19]. We omit some discussion due to space; for more information, see these articles.

J.P. Dickerson (✉)
Gates-Hillman Center, School of Computer Science, Carnegie Mellon University,
Pittsburgh, PA 15213, USA
e-mail: dickerson@cs.cmu.edu

G.I. Simari
Department of Computer Science, University of Oxford, Oxford OX1 3QD, UK
e-mail: gerardo.simari@cs.ox.ac.uk

V.S. Subrahmanian
Department of Computer Science, University of Maryland College Park,
College Park, MD 20742, USA
e-mail: vs@cs.umd.edu

V.S. Subrahmanian (ed.), *Handbook of Computational Approaches to Counterterrorism*,
DOI 10.1007/978-1-4614-5311-6_12,

stock of IBM will rise by at least 3 % at time $(t + \Delta)$ is 90 % given that, at time t, early coverage from blogs is positive about an upcoming earnings call and, also at time t, IBM does not announce fresh layoffs." The processes required to collect this data are outside the scope of this chapter, but a large financial institution has the resources to scrape web and print sources from which such rules could be learned.

2. Advances in both electronic record keeping and large-scale data analysis have introduced the "big data" mentality into **medicine**. For example, the Dartmouth Atlas of Health Care [22] aggregates US health care data across multiple dimensions (time, location, socioeconomic status, gender, severity, etc.); however, like other such large projects, it is largely sourced from billing data. Billing data alone is both incomplete and inaccurate, so reasoning over such uncertain temporal data is difficult. For instance, medical practitioners or policy analysts may wish to write rules of the sort, "the probability that a patient will return to the hospital before time $(t + \Delta)$ is 10 % if the patient was not in the ICU at time t and the patient's visit lasted less than 1 hour."
3. Large-scale data collection regarding **environmental phenomena** has resulted in a deluge of noisy, temporal data available to the public. For example, a government warning agency may wish to announce that, "if a forest fire occurs at time t and the amount of rain at time t is less than 0.1cm, then the probability that the fire will continue at time $(t + \Delta)$ is at least 85 %."
4. The Minorities at Risk research project [23] monitors the conflicts and activities of minority ethnicities, religious sects, and **terrorist groups** around the world. Our group at the University of Maryland has worked extensively with this data, and published analyses of some of these groups' behaviors (e.g., Hezbollah [10] and Hamas [11]). We built the SOMA Terror Organization Portal [14], which has registered users from over 12 US government agencies and contains thousands of (automatically) extracted rules about various groups' behaviors. Analysts engaged in counter-terrorism efforts need to be able to reason with such rules and make appropriate forecasts; in separate work, we have also done extensive work on making such forecasts [13, 15]. In this chapter, we formulate a running example in the context of the terrorist group Lashkar-e-Taiba.

In this chapter, we discuss two related types of logic programs that allow for logical reasoning in situations that involve temporal uncertainty. In Sect. 2, we first discuss *temporal probabilistic logic programs* (TPLPs), originally formulated by Dekhtyar et al. [6] as an extension to the generalized annotated programs (GAPs) of Kifer and Subrahmanian [9]. TPLPs allow for reasoning about point probabilities over time intervals using temporal probabilistic rules (tp-rules). In Sect. 3, we present an algorithm for automatically learning tp-rules from data, as detailed in [20]. We are also present a method for making policy recommendations by employing standard integer programming techniques to the automatically learned rules. Then, in Sect. 4, we describe a large-scale system we recently built to analyze terror groups using tp-rules. Using this system, we automatically learn rules about the south Asian terrorist group Lashkar-e-Taiba. In Sect. 5, we conclude with a

discussion of future research directions, including an adaptation of our architecture to the recently introduced *annotated temporal probabilistic* (APT) logic programs, an extension to TPLPs that does not make independence assumptions about the underlying features and allow for reasoning over probability intervals over time periods, rather than just point probabilities.

2 Modeling Group Behavior with Temporal Probabilistic Logic Programs

Temporal probabilistic logic programs (TPLPs) were first introduced by Dekhtyar et al. in [6]. The system provides a framework within which a logic programmer can express *tp-rules* of the form "If some condition is true, then some atom is also true at some time/time interval with some probability distribution over the points in the time interval." Dekhtyar et al. [6] also provided a syntax and semantics for temporal probabilistic logic programs, as well as initial complexity results. In this section, we overview TPLPs and tp-rules in the context of modeling group behavior.

2.1 Database Schema for a Group's Past Behavior

Before defining the general temporal probabilistic logic, we introduce a running example that focuses on Lashkar-e-Taiba (LeT), a well-known, active South Asian terrorist group. The example uses real data collected by the *Computational Modeling of Terrorism* (CMOT) codebook [17], a research project that records past and current activities of multiple terrorist groups including LeT.

We view the data as a single relation consisting of tuples with two types of attributes: *environmental* and *action*. Environmental attributes correspond to aspects of the environment in which the group operated, while action attributes correspond to the various types of actions taken by a group, along with their intensities. Each tuple corresponds to the set of these attributes' values for a given a month. Example 1 gives a very small subset of the raw data collected on LeT.

Example 1. The table below shows four attributes of CMOT data collected for Lashkar-e-Taiba across 12 months in 2004.

The first column is a date labeling each tuple. The next column corresponds to the action attribute `attackCiv`, a binary variable that is activated if LeT both attacked civilians during a given month and that attack resulted in casualties.[2]

[2]The CMOT codebook tracks fine-grained aspects of violent group behavior. Other civilian attack-related attributes include attacks on civilian transportation, attacks on civilians without civilian casualties, and attacks specifically targeting civilian minorities.

Date	AttackCiv	Religious	Raided	PersonnelKilledJK
Jan 2004	1	1	0	13
Feb 2004	0	1	1	23
Mar 2004	0	1	0	10
Apr 2004	0	1	0	8
May 2004	0	1	0	15
Jun 2004	0	1	0	7
Jul 2004	0	1	0	14
Aug 2004	0	1	0	13
Sep 2004	0	1	0	11
Oct 2004	0	1	0	25
Nov 2004	0	1	0	16
Dec 2004	0	1	1	9

The next three columns correspond to environmental attributes. The attribute `religious` is set to 1 if LeT operated as a religious organization during a specific month. We see that LeT operated as a religious organization during every month of 2004. The attribute `raided` is a binary variable that is set to 1 if the government of a host country raided LeT during a specific month. The data shows that this occurred in February and December of 2004. Finally, the last column, `personnelKilledJK`, is an integral variable that takes nonnegative values corresponding to how many members of Lashkar-e-Taiba were killed in the northernmost Indian state of Jammu and Kashmir.

Example 1 considers a subset of the database where each attribute has a value for each time period. This need not be the case; attribute values can be left unset if they are unknown. For instance, the CMOT database considers group behavior over many decades; data on some attributes may no longer be available (e.g., pertaining to the number of kidnappings that occurred, or whether or not LeT actively lobbied the government of Pakistan).

We will now define the formal syntax for temporal probabilistic logic, through which we will be able to learn tp-rules with which we can reason about a group's past and future behavior.

2.2 Syntax

We assume the existence of a first order logical language with finite set $\mathscr{L}_{cons}$ of constant symbols, finite set $\mathscr{L}_{pred}$ of predicate symbols, and infinite set $\mathscr{L}_{var}$ of variable symbols. Each predicate symbol $p \in \mathscr{L}_{pred}$ has an *arity* (denoted *arity(p)*). A (ground) *term* is any member of $\mathscr{L}_{cons} \cup \mathscr{L}_{var}$ (resp. $\mathscr{L}_{cons}$); if $t_1, \ldots, t_n$ are (ground) terms, and $p \in \mathscr{L}_{pred}$, then $p(t_1, \ldots, t_n)$ is a (ground) atom.

In the context of the behavioral data discussed in Sect. 2.1, every attribute corresponds to a predicate symbol. In fact, each attribute in the example (and, in

fact, the entire CMOT codebook) represents a *unary* predicate symbol. Although our formalization is easily generalized, we will thus concentrate only on predicates p such that $arity(p) = 1$. Let p be the predicate corresponding to an attribute, and t a term in the domain of p. Then $p(t)$ is an *action atom* when p corresponds to an action attribute, and an *environmental atom* when p corresponds to an environmental attribute. Finally, if $X \in \mathscr{L}_{var}$ and $Y \in \mathscr{L}_{cons}$, then $X = Y$, $X < Y$, $X > Y$, $X \leq Y$, and $X \geq Y$ are called *comparison atoms*.

Example 2. In this example, we use the table of data shown in Example 1. In this table, `attackCiv` is an action attribute with domain 0 and 1. In January 2004, Lashkar-e-Taiba's attacks on civilians resulted in civilian casualties; we represent this using the ground atom `attackCiv`(1). Similarly, `personnelKilledJK` is an environmental attribute whose domain is the non-negative integers. If $X \in \mathscr{L}_{var}$ ranges over the non-negative integers, then `personnelKilledJK`(X) can be instantiated to represent any number of LeT personnel killed in Jammu and Kashmir. For example, in January 2004, we instantiate $X = 13$ to return the ground atom `personnelKilledJK`(13).

We now formally introduce the concept of time. Let $T = \{1, \ldots, \tau_{max}\}$ denote the entire set of time points in which we are interested. We require a fixed time window size ranging over T, but allow τ_{max} to be arbitrarily large. The user may choose both the granularity of T and τ_{max} in an application-specific way. For instance, in the stock market example given in Sect. 1, a user may be interested in reasoning about 15-min segments (when the market is open) over the course of 10 years, and would set τ_{max} to around 78, 000 to represent 30 periods per day over roughly 2,600 trading days. On the other hand, our terrorism application does not require such a fine-grained temporal resolution. The CMOT codebook records data on the order of months, so we use a t_{max} of approximately 240 to reflect an interest in events over the past 20 years.

Given time period $\tau \in T$ and probability $\rho \in [0, 1]$, we call $[\tau, \rho]$ a *temporal-probabilistic annotation* (or *tp-annotation*). Intuitively, a tp-annotation $[\tau, \rho]$ refers to some unspecified event occuring exactly τ time periods after a given time, with a probability of ρ.

We now syntactically connect time to our fledgling logic. Given a tp-annotation $[\tau, \rho]$ and an action (environmental) atom $p(t)$, we call $p(t) : [\tau, \rho]$ an action (environmental) *tp-annotated atom*. If $p(t)$ is ground then $p(t) : [\tau, \rho]$ is called *ground* as well. Intuitively, $p(t) : [\tau, \rho]$ says that $p(t)$ will occur with probability ρ exactly τ time intervals after some fixed time. Example 3 gives sample tp-annotated atoms in the context of our running example.

Example 3. The action tp-annotated atom `attackCiv`(1) : [3, 0.9] states that there is a 90 % chance of Lashkar-e-Taiba carrying out deadly attacks against civilians in 3 time units after some fixed time. The environmental tp-annotated atom `personnelKilledJK`(4) : [1, 0.5] states that there is a 50 % chance that personnel belonging to Lashkar-e-Taiba will be killed in Jammu and Kashmir in 1 time unit after some fixed time.

We are now ready to introduce the main basic reasoning tool used in our analysis.

Definition 1 (Temporal probabilistic rule). If $p(t) : [\tau, \rho]$ is a tp-annotated atom and $A_1, A_2, \ldots, A_n$ are atoms (or comparison atoms), then

$$p(t) : [\tau, \rho] \leftarrow A_1 \wedge A_2 \wedge \ldots \wedge A_n$$

is a *temporal-probabilistic rule* (tp-rule). The *head* of the rule is $p(t)$, and the *body* of the rule is $A_1 \wedge A_2 \wedge \ldots \wedge A_n$.

Intuitively, such a tp-rule r states that if each atom in $body(r)$ is true at a fixed time, then the $head(r)$ atom will be true with probability ρ at a time τ units afterward.

Definition 2 (Temporal probabilistic logic program). A *temporal probabilistic logic program* (TPLP) is a finite set of tp-rules.

Example 4 presents a small TPLP consisting of a subset of tp-rules learned about LeT from real data.

Example 4. The following tp-rules, $\{r_1, \ldots, r_9\}$, form a small TPLP that focuses on the attack patterns of LeT. These rules were learned from the full set of CMOT data available for LeT (of which Example 1 displays a small subset).

$$
\begin{aligned}
&r_1.\texttt{attackCiv}(1) : [1, 1.0] \quad \leftarrow \texttt{religious}(1) \wedge \texttt{leadersDied}(X) \wedge X \leq 2.\\
&r_2.\texttt{attackHin}(1) : [3, 0.909] \leftarrow \texttt{terrClaims}(0) \wedge \texttt{leadersDied}(X) \wedge X \leq 2.\\
&r_3.\texttt{attackCiv}(1) : [1, 1.0] \quad \leftarrow \texttt{religious}(1) \wedge \texttt{raided}(X) \wedge X \leq 12.\\
&r_4.\texttt{attackSym}(0) : [3, 0.909] \leftarrow \texttt{locIndia}(1) \wedge \texttt{leadersDied}(X) \wedge X \leq 5.\\
&r_5.\texttt{attackSym}(0) : [2, 0.976] \leftarrow \texttt{locIndia}(1) \wedge \texttt{leadersDied}(X) \wedge X \leq 4.\\
&r_6.\texttt{attackSym}(0) : [3, 0.909] \leftarrow \texttt{locIndia}(1) \wedge \texttt{personnelRel}(X) \wedge X \leq 9.\\
&r_7.\texttt{attackHol}(1) : [2, 0.917] \leftarrow \texttt{remInfluenceJK}(1) \wedge \texttt{personnelKilled}(X)\\
&\qquad\qquad\qquad\qquad\qquad\qquad \wedge X \leq 8.\\
&r_8.\texttt{attackHol}(1) : [2, 0.909] \leftarrow \texttt{personnelArrested}(X) \wedge X \leq 8.\\
&r_9.\texttt{attackHol}(1) : [2, 0.917] \leftarrow \texttt{advChangeLife}(1) \wedge \texttt{personnelKilled}(X)\\
&\qquad\qquad\qquad\qquad\qquad\qquad \wedge X \leq 8.
\end{aligned}
$$

Temporal probabilistic rule r_1 states that, at time t, if Lashkar-e-Taiba is operating as a religious group and the number of group leaders who died during this time interval is at most 2, then with 100 % probability LeT will perform deadly attacks against civilians at time $t + 1$. The environmental atom $\texttt{religious}(1)$ ensures that, when $body(r_1)$ is true, LeT is operating as a religious group. Similarly, the environmental atom $\texttt{leadersDied}(X)$ and the comparison atom $X \leq 2$ combine to ensure that, when $body(r_1)$ is true, at most two leaders of LeT died during this time interval. Finally, $head(r_1)$ is an action atom stating that LeT performs deadly attacks against civilians when $body(r_1)$ is true.

As another example, rule r_6 states that with 90.9 % probability, LeT will *not* attack symbolic sites at time $t + 3$ if at time t LeT has active locations across the border of India and at most 10 LeT personnel were released by the government

during the time interval t. Unlike rule r_1, rule r_6 states that LeT will *not* perform an attack (with high probability); this is specified in $head(r_6)$, where the ground term serving as an argument for predicate `attackSym` has value 0 instead of 1.

While the examples above focus primarily on describing the attack patterns of a terrorist group, we emphasize that this temporal probabilistic logic can easily be used in other domains. Regardless of the domain, it is clear that *manually* determining tp-rules and TPLPs from historical data or expert opinions would quickly grow intractable. In the next section, we present a method to learn tp-rules automatically from historical data, as well as a general method for extracting *policy recommendations* (e.g., "reduce funding to LeT" or "sell stock in APPL but buy stock in GOOG") from these learned tp-rules.

3 Automatically Learning Rules from Historical Data

In Sect. 2, we formally introduced temporal probabilistic rules (tp-rules) and temporal probabilistic logic programs (TPLPs). In this section, we present a general method for automatically learning tp-rules from historical data. We then describe an integer programming-based method to derive "good" policy recommendations based on these learned tp-rules.

3.1 Automatic Extraction of TP-Rules

Temporal probabilistic reasoning is important in many domains (see Sect. 1), and tp-rules are one natural way for analysts and reasoning agents to formally write down their expert knowledge. However, *manually* constructing tp-rules from historical data is tedious in the small, infeasible in the large, and subject to human error and bias. For these reasons, it is necessary to remove the human from the tp-rule creation process in favor of automatically learning tp-rules from historical data.

3.1.1 SOMA Rules

Our method for learning tp-rules from historical data is heavily based on one by Subrahmanian and Ernst [20]. This algorithm was originally motivated by the need to mathematically model the behavior of terrorist groups, and operates on the first (to our knowledge) model used toward this end. The algorithm uses *Stochastic Opponent Modeling Agent rules* (SOMA-rules), which provide probabilistic but not temporal reasoning about a group. In fact, SOMA-rules are syntactically very similar (although they do not consider time) to the tp-rules discussed in this chapter, making statements of the form, "When conditions C are true in the environment

in which a terror group G operates, there is a probability of between l % and u % that G will take actions A at some intensity level L." We formalize this notion in Definition 3.

Definition 3 (SOMA-rule). If $A_1, A_2, \ldots, A_n$ are environmental or comparison atoms, $p(t)$ is an action atom, and $l, u \in [0, 1]$, then

$$p(t) : [l, u] \leftarrow A_1 \wedge A_2 \wedge \ldots \wedge A_n$$

is a *SOMA-rule*. As with tp-rules, the *head* of the rule is $p(t)$, and the *body* of the rule is $A_1 \wedge A_2 \wedge \ldots \wedge A_n$.

Recently, SOMA-rules have been used to formally present the behaviors of many terrorist groups. In the past 5 years, work by Mannes, Subrahmanian, and others has automatically learned expressed SOMA-rules about Hezbollah [10], Hamas [11], and Lashkar-e-Taiba [12]. These projects accessed historical data about their respective terrorist groups through the CMOT codebook, and have shown confirmed predictive power. For example, the work by Mannes et al. [10] covering Hezbollah made predictions about the group's behavior in early 2009 before the Lebanese elections. Hezbollah then made public comments in the Beirut Daily Star expressing skepticism about the predictions; however, the group proceeded to operate exactly as predicted in early 2009.

Formally, SOMA-rules use a constrained version of the syntax of probabilistic logic programs [16]. However, for the purposes of this section, we can think of SOMA-rules as tp-rules with no temporal offset and a point probability; that is, the tp-annotation $[\tau, \rho]$ will always have $\tau = 0$, and the corresponding SOMA-annotation $[l, u]$ will always have $l = u$. Intuitively, the non-trivial temporal offsets of tp-rules can be thought of as adding a notion of causality to SOMA-rules. This is accomplished by clearly separating the time interval during which the body of a tp-rule takes place (i.e., interval t) and the time interval during which the head of a tp-rule files (i.e., interval $t + \tau$).

We are now ready to present the algorithm by Subrahmanian and Ernst [20], as well as our straightforward augmentation to allow the algorithm to work with temporally-aware TPLPs.

3.1.2 Subrahmanian-Ernst Algorithm: Preliminaries

We now describe a method for automatically extracting SOMA-rules from a database, first proposed by Subrahmanian and Ernst [20]. We call this the Subrahmanian-Ernst (SE) algorithm. Afterward, we describe the small tweak required to adapt the method to extract tp-rules.

Definition 4 (Bi-conjunct). If p is a predicate, $X \in \mathscr{L}_{var}$, and $l, u \in \mathscr{L}_{cons}$, then

$$p(X) \wedge l \leq X \leq u$$

is a *bi-conjunct*.

The SE algorithm generates a specific type of SOMA-rules whose bodies consist of bi-conjuncts. We formally define these *bi-SOMA-rules* in Definition 5.

Definition 5 (Bi-SOMA-rule). If $B_1, B_2, \ldots, B_n$ are bi-conjuncts, $p(t)$ is an action atom, and $l, u \in [0, 1]$, then

$$p(t) : [l, u] \leftarrow B_1 \wedge B_2 \wedge \ldots \wedge B_n$$

is a *bi-SOMA-rule*. As with standard SOMA-rules, the *head* of the rule is $p(t)$. The *bi-body* of the rule is $B_1 \wedge B_2 \wedge \ldots \wedge B_n$. The *dimension* of a bi-body is the number of bi-conjuncts in it.

Clearly, the set of all bi-SOMA-rules is a subset of the set of all SOMA-rules, as the definition is identical to that of the SOMA rule with the added constraint of a specific combination of environmental and comparison atoms in the body of the rule. We now induce equivalence classes on the set of all bi-bodies of bi-SOMA-rules.

Definition 6 (Equivalence of bi-bodies). If r_1 and r_2 are bi-SOMA-rules with bi-bodies b_1 and b_2, then b_1 and b_2 are *equivalent* if and only if:

- The bi-conjuncts in bi-bodies b_1 and b_2 always co-occurred (i.e., the set of time intervals in which b_1 is true is identical to the corresponding set of time intervals for b_2); and
- The *environmental* atoms in both bi-bodies are identical (but not necessarily their respective comparison atoms).

The SE algorithm requires a *tight canonical member* from each equivalence class. Informally, if $B^* = \{B_1, B_2, \ldots, B_n\}$ is an equivalence class such that B_i contains some bi-conjunct $p(X) \wedge l_i \leq X \leq u_i$, then the tight canonical member chosen must contain the bi-conjunct:

$$p(X) \wedge \min_{i=1,\ldots,n} (l_i) \leq X \leq \max_{i=1,\ldots,n} (u_i)$$

The tight canonical member must contain similar "tight" (with respect to the equivalence class B^*) bi-conjuncts for each unique environmental atom in the bi-bodies $B_1, \ldots, B_n$.

Example 5. The bi-bodies B_1, B_2, and B_3 each have two bi-conjuncts (and thus $dimension(B_i) = 2$). Each bi-body references two environmental attributes, the binary-valued `religious` and nonnegative integral-valued `leadersDied`.

B_1. $[\texttt{religious}(X_1) \wedge 0 \leq X_1 \leq 1] \wedge [\texttt{leadersDied}(X_2) \wedge 0 \leq X_2 \leq 2]$
B_2. $[\texttt{religious}(X_1) \wedge 1 \leq X_1 \leq 1] \wedge [\texttt{leadersDied}(X_2) \wedge 0 \leq X_2 \leq 6]$
B_3. $[\texttt{religious}(X_1) \wedge 0 \leq X_1 \leq 1] \wedge [\texttt{leadersDied}(X_2) \wedge 1 \leq X_2 \leq 12]$

Assume the bi-conjuncts in each bi-body always co-occurred. Since each bi-body contains the same environmental atoms (although their respective comparison atoms

are different), they are also in the same equivalence class B. Then a tight canonical member of B^* is B_t, as shown below.

$$B_t.\ [\texttt{religious}(X_1) \wedge 0 \leq X_1 \leq 1] \wedge [\texttt{leadersDied}(X_2) \wedge 0 \leq X_2 \leq 12]$$

To aid in reasoning over tight canonical members of equivalence classes, the SE algorithm also induces an ordering on bi-bodies, formalized in Definition 7 below.

Definition 7 (Simpler than). If B_1 and B_2 are bi-bodies and $p(t)$ is an action atom, then B_1 is *simpler than* B_2 (denoted $B_1 \gg B_2$) if:

- $dimension(B_1) \geq dimension(B_2)$,
- $conf(B_1) \geq conf(B_2)$; and
- $sup(B_1) \geq sup(B_2)$.

The confidence of bi-body B_i, $conf(B_i)$, with respect to the action atom of interest $p(t)$ is defined as follows:

$$conf(B_i) = \frac{\#\text{intervals when } B_i \text{ was true and } p(t) \text{was true}}{\#\text{intervals when } B_i \text{ was true}}$$

The support, $sup(B_i)$, is just the numerator of the $conf(B_i)$ fraction.

We now define the structure computed as the end goal of the SE algorithm.

Definition 8 (Up-set). If B is a bi-body and d is a positive integer, then the *up-set* of B (denoted $up(B)$) is:

$$up(B) = \left\{B' \mid B' \text{isatightbi} - \text{body} \wedge\ dimension(B') \leq d\ \wedge\ B' \gg B\right\}$$

Intuitively, given some bi-body B and a maximum dimension d, the up-set of B is the set of all bi-bodies of dimension at most d that are also simpler than B. The SE algorithm computes layers of sets of bi-bodies based on these up-sets as follows:

Definition 9 ($Tp \uparrow k$). If d is a positive integer, then $\forall k \in \mathbb{Z}^+$ we define $Tp \uparrow k$ iteratively as follows:

$$\begin{aligned} Tp \uparrow 1 &= \{B \mid B\text{isatightbi} - \text{body} \wedge\ dimension(B) \leq d\ \wedge\ up(B) = \emptyset\} \\ Tp \uparrow (k+1) &= \{B \mid B\text{isatightbi} - \text{body} \wedge\ dimension(B) \\ &\qquad \leq d\ \wedge\ up(B) \subseteq Tp \uparrow k\} \end{aligned}$$

The set $Tp \uparrow 1$ is then the set of all bi-bodies B with d or fewer bi-conjuncts in the body, such that no other bi-body B' with d or fewer bi-conjuncts in the body is strictly simpler than B. The subsequent $Tp \uparrow i$ for $i > 1$ are "looser" versions of each parent set. The computation of these sets is the main purpose of the SE algorithm; however, naïvely computing all such sets (up to some constant integer k) would be intractable. To this end, we define the workhorse of the SE algorithm, the *condition graph* (COG).

Definition 10 (Condition graph). A condition graph (COG) is a graph $G = (V, E)$ such that $\forall v \in V$:

- $v.bibody$ is a label referencing a single, tight bi-body
- $v.level$ is a label that is set to 0 if there is no vertex v' such that $\exists (v', v) \in E$; otherwise, it is defined as $\max_{v' in V, v \neq v} level(v') + 1 \mid (v', v) \in E$.

Let $K \in \mathbb{Z}^+$ represent the maximum desired level of a COG. Then, for each bi-body $B \in TP \uparrow K$, there is exactly one vertex $v \in V$ such that $v.bibody = B$. This completely defines the set V.

The set E is defined as follows:

$$E = \{(v, v') \mid v, v' \in V \wedge (v.bibody \gg v'.bibody) \wedge \nexists w \in V \text{ s.t.} (v.bibody \gg w.bibody \gg v'.bibody)\}$$

Building the complete COG is a computationally difficult problem. To alleviate some of the computational complexity, the SE algorithm takes as a parameter a user-defined outcome (in our terror group example, an action atom) of interest, and computes only the portion of the COG relevant to that outcome. This is done by determining if a given bi-body references the outcome and, if it does not, ignoring it. Once this outcome-specific version of the COG is fully constructed, we need only extract the vertices from the COG that fall within the desired (user-specified) confidence and support intervals. We describe this process formally in the next section.

3.1.3 The Subrahmanian-Ernst Algorithm and an Adaptation to TPLPs

In this section, we formally describe the Subrahmanian-Ernst (SE) algorithm. We also adapt the algorithm to the temporal probabilistic logic presented in Sect. 2. This section builds on the formalizations of Sect. 3.1.2.

Algorithm 1 formally presents the Subrahmanian-Ernst algorithm. The algorithm takes as input:

- A database (DB) whose schema mirrors that discussed in Sect. 2.1. In the case of our running LeT example, this is a database whose rows correspond to months and columns correspond to action and environmental attributes.
- A list of environmental attributes (ENV). In the case of the LeT example, this is just the indices of the columns corresponding to environmental attributes.
- A positive integer d, the maximum dimension of a bi-body. For example, if $d = 3$, then all bi-bodies computed by the algorithm will have dimension at most 3.
- A positive integer k, determining the maximum level a vertex in the COG can attain.

Algorithm 1: Subrahmanian-Ernst algorithm

```
Data: Database DB, environmental attributes ENV, action atom Outcome, maximum
      dimension d ∈ ℤ+, maximum level k ∈ ℤ+
Result: Set of bi-bodies relevant to Outcome that satisfy pre-defined support and confidence
        levels
begin
    Set COG = (V, E) with V = E = ∅
    foreach combination φ of d or fewer attributes in ENV do
        SatTuples = BuildDataStructure(DB, ENV, φ, Outcome)
        NotSatTuples = BuildDataStructure(DB, ENV, φ, ¬Outcome)
        TightBibodies = GenerateTightBibodies(φ, SatTuples)
        foreach vertex v ∈ TightBibodies do
            numNotSat = CountQuery(v.bibody, NotSatTuples)
            v.confidence = v.support/(v.support + numNotSat)
            COG = InsertCOG(v, COG, k)
        end
    end
    return ExtractBibody(COG)
end
```

Algorithm 1 references five undefined procedures. We describe them here.

BuildDataStructure. Informally, this procedure splits the DB into two subsets of rows: those satisfying an outcome and those not satisfying an outcome. In the algorithm, after calling *BuildDataStructure* with "Outcome" as a parameter, the "SatTuples" variable contains the projection of DB on attributes in the combination ϕ for specific tuples that satisfy the user-defined outcome atom. The "NotSatTuples" variable then contains the projection that do not satisfy the outcome atom, since it is the product of calling *BuildDataStructure* with "¬Outcome".

GenerateTightBibodies. This procedure generates the support of all tight bi-bodies associated with the combination ϕ. A set of vertices corresponding to these tight bi-bodies is returned, such that for each vertex the *confidence*, *support*, and *bibody* fields are set properly.

CountQuery. This procedure counts the total number of tuples that satisfy the bi-body of a specific vertex, but do not satisfy the user-specified outcome atom.

InsertCOG. This procedure is called once per vertex returned by the *GenerateTightBibodies* procedure. The procedure first checks the level of the vertex and, if the level is at most k, inserts the vertex into the COG. The procedure also propagates the *level* value to neighbors of the vertex. If this cascade of updates forces any vertex's level to exceed k, the vertex is removed from the COG.

ExtractBibody. This procedure checks every vertex in the COG and, if the vertex satisfies some user-defined confidence and support criteria (e.g., "only report bi-bodies with support above 10 and confidence above 90 %"), reports the corresponding bi-body. The set of all such bi-bodies is then returned by the algorithm.

Algorithm 2: OffsetDB algorithm

Data: Database DB, temporal offset $\tau \in \mathbb{Z}$
Result: Temporally-augmented database DB$'$
begin
  Set DB$' =$ DB
  foreach *row* r_i *in DB*$'$ **do**
    if $i < \tau$ **then**
      Delete r_i from DB$'$
    else
      foreach *environmental attribute* E **do**
        Let $r'_{i-\tau}$ be row $i - \tau$ in the original database DB
        Replace $r_i(E)$ with $r'_{i-\tau}(E)$
      end
    end
  end
  return *DB*$'$
end

As presented, Algorithm 1 does not take time offsets into account. In other words, it will always return tp-rules that have a trivial temporal component. In Algorithm 2 (the OffsetDB algorithm), we provide a simple way to augment our database DB such that the SE algorithm returns general tp-rules.

Informally, Algorithm 2 takes as input the raw database of historical data DB, and outputs an augmented database DB$'$ such that each environmental attribute in DB$'$ has been "pushed up" τ rows. In this way, the temporal offset is built into the database DB$'$. The SE algorithm is then called with DB$'$ as the data source, and proceeds normally.

For a specific time offset τ, by invoking the OffsetDB algorithm followed by the SE algorithm once for every outcome of interest (e.g., for every action attribute corresponding to LeT performing violent attacks), an analyst can derive all possible tp-rules that satisfy specified support, confidence, and dimension levels for an offset of τ. Then, for all time offsets of interest (e.g., between 0 and 5 months), an analyst can derive all possible tp-rules for any time offset.

3.2 Toward Converting TP-Rules into Policy Recommendations

The SE algorithm presented in Sect. 3.1.3 automatically learns expressed tp-rules from historical data. These learned tp-rules can be analyzed manually by area experts and used to determine policies of actions; however, as with the creation of the tp-rules themselves, this is both intractable in the large and subject to both human bias and mental capacity constraints. For instance, in our running example focusing on attacks by Lashkar-e-Taiba, an immediately obvious policy for reducing

attacks in one dimension might have unforeseen repercussions at different points of time or with different types of attacks. In this section, we present a method for automatically extracting desirable policies from a database of tp-rules. The method makes a few assumptions that should be relaxed in future work; we discuss these as well.

3.2.1 Computational Policies

Informally, a *policy* is a specific setting of a (subset of the) world that, when present, triggers desirable properties elsewhere in the world. For example, in the context of effecting change in a terrorist group's behavior, a governing body or advisory committee may be interested in understanding what changes it could make to the environment in which a group operates (e.g., cutting down on foreign aid or increasing raids) so that the group behaves differently (e.g., no longer attacks civilians).

Before formally defining a policy in the language of our temporal probabilistic logic, we discuss a fairly strong assumption: that the tp-rules over which we are reasoning can be represented in *propositional logic*. That is, terms in the body of each rule are all ground. The assumption that each body term is ground lets us view the body of each rule, consisting of atoms $A_1 \wedge A_2 \wedge \ldots \wedge A_n$, as a conjunction of *literals*. We can then reason about these literals and their negations in the standard way. For example, `religious`, which has domain $\{0, 1\}$, can be viewed as two complementary literals `religious(1)` and `religious(0)`. In our experience learning real tp-rules from data, this assumption is not too confining (in fact, as we will discuss in Sect. 4, our recent study focusing on preventing attacks by Lashkar-e-Taiba used only rules of this type). Future research will relax this requirement.

For the rest of this section, we will assume the existence of a set of tp-rules *RDB* (called a *rule database*). This set of tp-rules could have been learned automatically using techniques like those presented in Sect. 3.1 or constructed manually. Let *body*(*RDB*) be the set of all literals appearing in the body of any tp-rule in the rule database *RDB*. Furthermore, let $\neg$*body*(*RDB*) be the set of all literals $\{\neg\ell \mid \ell \in$ *body*(*RDB*)$\}$. We now formally define a policy.

Definition 11 (Policy). Given a set of tp-rules *RDB* (called a *rule database*) and a set of action atoms A, a *policy that potentially eliminates* A is a consistent subset of $\neg$*body*(*RDB*) that satisfies the following:

1. $\forall r \in RDB$ such that $head(r) \in A$, $\exists \ell \in P$ such that $\neg\ell \in body(r)$
2. $\nexists P' \subset P$ such that P' satisfies the preceding condition

Intuitively, given a database of tp-rules *RDB* and a set of action atoms that we would like to prevent, a policy is a way to set environmental variables such that no tp-rules pertaining to the specific set of action atoms fire. Furthermore, it is the "simplest" such set in that no strict subset of the policy would result in none of the desired tp-rules firing. Since, by definition, the policy is a consistent subset of

$\neg body(RDB)$, it cannot contain both literals ℓ and $\neg\ell$; if this were not the case, it would be impossible to implement the policy.

Example 6. The following set of tp-rules, $\{r_1, r_2, r_3\}$, forms a small rule database *RDB* that focuses on the attack patterns of LeT toward civilians.

r_1.attackCiv(1) : [1, 0.99] ← terrClaims(0) ∧ religious(1)
r_2.attackCiv(1) : [3, 0.909] ← terrClaims(0)
r_3.attackCiv(1) : [1, 0.916] ← remInfluenceJK(1) ∧ advChangeLife(1)

Let $A = \{\texttt{attackCiv}(1)\}$, representing a desire to prevent LeT from attacking civilians. There are two possible policies that potentially eliminate A:

- $P_1 = \{\texttt{terrClaims}(1), \texttt{remInfluenceJK}(0)\}$
- $P_2 = \{\texttt{terrClaims}(1), \texttt{advChangeLife}(0)\}$.

Clearly, any policy must include terrClaims(1), since this is the only way to prevent rule r_2 from firing. This also prevents rule r_1 from firing. Finally, we can choose to negate either of the components in $body(r_3)$. Thus, both P_1 and P_2 prevent all rules $r \in RDB$ pertaining to the set of action atoms A from firing; furthermore, no strict subset of either P_1 or P_2 satisfies this statement, and both P_1 and P_2 are consistent, so both P_1 and P_2 are policies that potentially eliminate A.

3.2.2 Iteratively Computing All Policies

We now describe the computational method used to automatically generate policies from a set of tp-rules. The algorithm we will describe builds upon integer linear programming techniques for computing the set of all minimal models of logic programs, originally discussed in Bell et al. [1]. We now explain its straightforward adaptation to the case of temporal probabilistic logic.

First, we define a set of linear constraints (*LC*) that enforce the formal rules of a policy, as defined above. Assume we have a tp-rule database RDB' and a set of action atoms A; for convenience, denote $RDB = \{r \in RDB' \mid head(r) \in A\}$. For each literal $\ell \in body(RDB)$, let X_ℓ be a binary variable representing whether or not literal ℓ is included in a policy. Similarly, define binary variable X_a for each $a \in A$. Then we define the set of linear constraints *LC* as follows:

1. For each rule $a \leftarrow \ell_1 \wedge \ell_2 \wedge \ldots \wedge \ell_n$, add a constraint

$$X_a + \sum_{i=1}^{n} (1 - X_{\ell_i}) \geq 1$$

   Intuitively, this constraint forces either the head of the rule (represented by X_a) to be true, or at least one of the literals in the body to be false.
2. For each pair of complementary literals ℓ and $\neg\ell$, add a constraint

Algorithm 3: Policy computation algorithm

```
Data: Database of tp-rules RDB, set of action atoms A
Result: Set of policies P
begin
    P = ∅
    (RDB′, LC) = MakeConstraints(RDB, A)
    while true do
        S = CalculateHS(RDB′, LC)
        if S exists then
            P = P ∪ {¬ℓ | ℓ ∈ S}
            LC = LC ∪ {∑_{ℓ∈S} X_ℓ ≤ card(S)}
        else
            return P;
        end
    end
end
```

$$X_\ell + X_{\neg\ell} \leq 1$$

This ensures consistency; that is, at most one of the complementary literals is included in a policy.

3. For each rule r and $a \in A$, if $a \in head(r)$, add the constraint

$$X_a = 0$$

This ensures that no rule (of interest) fires.

4. Ensure that each X_a and X_ℓ variable is binary by adding a constraint

$$X_{\{a,\ell\}} \in \{0, 1\}$$

The savvy reader will notice that we can combine the constraints in item 1 with those in item 3, removing the need for the X_a variables ranging over the action atoms in the heads of tp-rules entirely. We choose to present LC in a more general way. In the event that LC is over-constrained (that is, there is no policy P such that *no* tp-rule in the rule database fires), a policy analyst could relax the constraint in item 3 and then try to minimize the number of tp-rules that fire (instead of requiring that none fire at all).

Second, using this initial set of linear constraints LC, we iteratively solve a series of integer programs (minimizing the number of activated X_ℓ variables), adding constraints to LC until the program becomes infeasible. The solution to each intermediary integer program represents a legal policy that potentially eliminates A, given some set of action atoms A. Algorithm 3 formalizes this process.

Algorithm 3 makes use of two previously undefined functions:

MakeConstraints. Given a rule database and a set of action atoms, this returns the initial set of linear constraints LC as defined earlier in the section, as well as a

filtered rule database RDB' containing only tp-rules pertinent to the set of action atoms.

CalculateHS. This function calculates a minimum hitting set for the bodies of the pertinent tp-rules in the filtered rule database RDB', subject to the constraint defined by LC. The minimum hitting set can be calculated using the linear integer program:

$$\begin{aligned} &\min \textstyle\sum_{\ell \in body(RDB')} X_\ell \\ &\text{s.t. } LC \end{aligned}$$

Intuitively, Algorithm 3 iteratively produces minimum hitting sets consisting of a literals in rule bodies such that, were those literals to be negated, no tp-rules (in the filtered rule database RDB') would fire. After each successful solve of the integer program, a new constraint is added to LC preventing any strict superset of the most recently determined policy from being found in the future. In this way, we ensure that only legal policies are found. Finally, once all policies are found, the integer program becomes infeasible and the algorithm returns the set of all policies that potentially eliminate the user-specified set of action atoms.

In the next section, we provide an extensive application of Algorithms 1–3 to a large, real-world database representing the actions and operating environment of Lashkar-e-Taiba, an active terror group in southern Asia.

4 Policy Recommendations and Lashkar-e-Taiba

In this section, we apply the techniques discussed in Sect. 3 to study environments that provoke attacks by Lashkar-e-Taiba (LeT), a terror group in South Asia. Over the last two decades, LeT has been responsible for many terrorist attacks in India, Kashmir, Pakistan, and Afghanistan. In 2006, LeT operative Faheem Lodhi was arrested and convicted of planning sophisticated attacks on Australia's power grid [4], demonstrating the potential global threat of this organization.

We learn a set of tp-rules from real-world data collected by the *Computational Modeling of Terrorism* (CMOT) codebook [17], a research project that tracks past and current activities (recording data at a granularity level of months) of multiple terrorist groups including LeT. We then determine a set of policies that could help prevent further attacks by LeT. A far more in-depth discussion of these results in can be found in [21].

4.1 Experimental Methodology and Learned Rules

The CMOT codebook tracks hundreds of environmental and action variables for Lashkar-e-Taiba, recording intensity levels on a month-by-month basis. A few examples of environmental variables include those relating to:

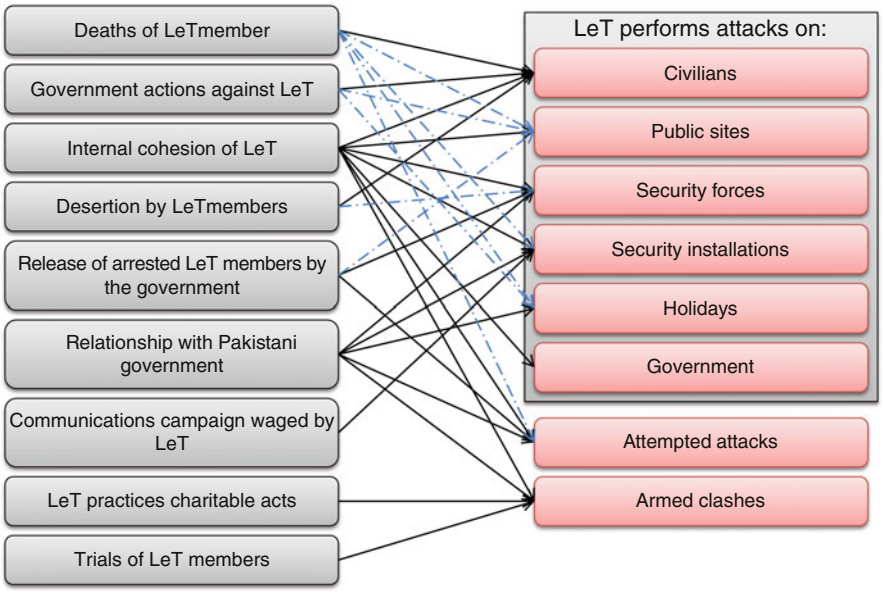

Fig. 1 A graphical summary of high support, high probability tp-rules learned about Lashkar-e-Taiba. *Solid black lines* from an environmental attribute to an action attribute represent a positive correlation, while *dashed blue lines* represent negative correlation

- The internal politics and activities of LeT (e.g., "What level of intra-organizational conflict exists in LeT?");
- The level of local and international monetary, military, and political support for LeT (e.g., "At what level is Pakistan's military supporting LeT?"); and
- Information about the group's operating facilities and staffing.

Examples of action variables tracked by the CMOT codebook include those relating to:

- Armed and suicide attacks against military forces, security forces, or civilians;
- Hijackings and abductions/kidnappings; and
- Attacks on military targets, government facilities, tourist sites, or symbolic sites.

We learned tp-rules using all of the action and environmental variables tracked by the CMOT codebook. These rules were learned automatically using Algorithms 1 and 2. We then filtered these rules to include only those with high *support* in the data and *probability* of occurring. Figure 1 shows a summary of the learned rules.

For example, Fig. 1 states that increases in the environmental variable tracking the deaths of LeT members is positively correlated with increased attacks on civilians, while increases in the same environmental attribute is negatively correlated with increased attacks on public sites. A much more in-depth discussion of the data, experimental methodology, and set of learned tp-rules can be found in upcoming

work by Subrahmanian et al. [21]. These results clearly show the expressive power of tp-rules and the promise of the methods for automatically learning them from real-world data.

4.2 *Policies That Potentially Eliminate or Reduce Violent Attacks by Lashkar-e-Taiba*

Using the filtered tp-rules described above and the policy computation methodology described in Algorithm 3, we computed a set of policies that potentially eliminate or reduce violent attacks by Lashkar-e-Taiba. Critically, we are *not* claiming that instituting these policies in reality will stop all attacks by LeT; rather, they may be effective at changing LeT's behavior. These policies are based on tp-rules forming a behavioral model of LeT based on past behavior; in reality, terrorist groups frequently change their behavior in response to counter-terror strategies [8]. In this light, it is imperative that policies adapt to the changing actions and strategies of groups like LeT.

The set of tp-rules produced eight policies. The policies were overall quite similar to one another, varying individually in a subtle ways. We very briefly describe them now. Overall, each of the policies suggest:

- Targeting LeT's internal cohesion;
- Targeting the Pakistani military's support of LeT;
- Targeting LeT's training facilities;
- Targeting any communication campaigns launched or run by LeT;
- Pushing for the resignation of senior LeT leaders;
- Keeping LeT prisoners (i.e., preventing the release of LeT prisoners by the governments that hold them);
- Reconsidering targeted efforts and long-term campaigns to kill or arrest LeT[3]; and
- Not explicitly encouraging low-level personnel to defect from LeT.

Individual variability amongst the policies was low. Individually, the policies suggested taking one or some of the following actions (in addition to those listed above):

- Targeting social and medical services run on the local level by LeT;
- reducing media coverage and publicity of trials of LeT members (especially in Australia);
- Maintaining or pushing for a government ban by Pakistan on LeT; and

[3]This is an interesting point. We emphasize that this is not discouraging governments or groups from working to arrest active LeT members. Rather, explicit campaigns to arrest members can lead to mixed and sometimes dangerous responses.

- Disrupting or targeting relationships between LeT and other Islamic organizations.

Clearly, no one policy offers a simple and deterministic route to preventing violent attacks by LeT. Furthermore, these policies would need to adjust to the constantly adapting strategies and actions of the active terror group. As statisticians George Box and Norman Draper wrote, "essentially, all models are wrong, but some are useful [3]." It is our hope that the policies presented here will be useful.

5 Conclusions and Directions for Future Research

Many applications require logical reasoning about situations that involve temporal uncertainty, including predicting movements in the stock market, assessing the potential future damage of environmental disasters, and reasoning about the behavior of terror groups. In this chapter, we overviewed temporal probabilistic logic programs (TPLPs), through which logic programmers can formally express rules that have both temporal and probabilistic aspects. We provided a general method to derive TP rules from databases of categorical and numerical variables based on work by Subrahmanian and Ernst [20]. We also presented a general method to provide "good" policy recommendations based on these automatically learned rules. Finally, we presented recent work that led to a successful, large-scale application of these techniques to model Lashkar-e-Taiba, an active militant terrorist group.

The framework we described in this chapter automatically finds expressed causal rules within historical data and presents the end user with a set of suggestions (e.g., policies in the case of terror groups) based on the rules found in the data. This framework could easily be adapted to handle different types of temporal reasoning systems. For instance, a recent extension to temporal probabilistic logic called *annotated probabilistic temporal* (APT) logic increases the expressiveness of tp-rules [18, 19]. Like TP logic, APT logic does not make independence assumptions; however, it provides bounds on probabilities as opposed to using only point probabilities. This generality could provide, for example, a more expressive system for policy recommendations. To our knowledge, systems based on APT logic have not yet been implemented in the large.

The integer programming-based method for finding desirable policies given a set of tp-rules can, as we found while doing experiments on the real-world LeT data, become overconstrained. This is due in part to the fact that *real-world groups are not rational*, leading to seemingly contradictory actions which leads to an infeasible hitting set problem. Expert knowledge could be used to cut out contradictory rules from a TPLP; however, manual interaction with large sets of tp-rules can be difficult, and this would be prone to human error and bias. Instead, a policy analyst could relax the objective function from preventing *all* rules from firing to discovering the *largest* subset of rules that could be prevented from firing. One technique to do this, again using integer programming, is suggested by Bell et al. [1, 2]. This problem

is equivalent to the maximum Boolean satisfiability problem (MAX-SAT), is NP-complete, and could still be solved using an industry-standard integer programming solver. We suspect a method like this will likely be necessary when dealing with large sets of tp-rules learned about imperfectly rational groups.

Acknowledgements Some of the authors were funded in part by AFOSR grant FA95500610405, ARO grant W911NF0910206 and ONR grant N000140910685.

References

1. Bell C, Nerode A, Ng R, Subrahmanian V (1994) Mixed integer programming methods for computing nonmonotonic deductive databases. J ACM 41(6):1178–1215
2. Bell C, Nerode A, Ng R, Subrahmanian V (1996) Implementing deductive databases by mixed integer programming. ACM Trans Database Syst 21(2):238–269
3. Box G, Draper N (1987) Empirical model-building and response surfaces. Wiley, New York
4. Brenner J, Frazzetto M (2011) America the vulnerable
5. De Choudhury M, Sundaram H, John A, Seligmann DD (2008) Can blog communication dynamics be correlated with stock market activity? In: Proceedings of the 19th ACM conference on hypertext and hypermedia (HC-08). ACM, New York, pp 55–60
6. Dekhtyar A, Dekhtyar MI, Subrahmanian VS (1999) Temporal probabilistic logic programs. In: ICLP 1999. MIT, Cambridge, MA, pp 109–123
7. Fujiwara I, Hirose Y, Shintani M (2008) Can news be a major source of fluctuation: a Bayesian DGSE approach, vol. Discussion Paper Nr. 2008-E-16. Institute for Monetary and Economic Studies, Bank of Japan
8. Ganor B (2005) The counter-terrorism puzzle: a guide for decision makers. Transaction Publishers, New Brunswick
9. Kifer M, Subrahmanian V (1992) Theory of generalized annotated logic programming and its applications. J Log Program 12:335–367
10. Mannes A, Michaell M, Pate A, Sliva A, Subrahmanian V, Wilkenfeld J (2008) Stochastic opponent modelling agents: a case study with hezbollah. In: Proceedings of the 2008 first international workshop on social computing, behavioral modeling and prediction. Springer, Berlin/New York
11. Mannes A, Sliva A, Subrahmanian V, Wilkenfeld J (2008) Stochastic opponent modeling agents: a case study with hamas. In: Proceedings of the 2008 international conference on computational cultural dynamics. AAAI, Menlo Park, pp 49–54
12. Mannes A, Shakarian J, Sliva A, Subrahmanian V (2011) A computationally-enabled analysis of Lashkar-e-Taiba attacks in Jammu & Kashmir. In: Proceedings of European intelligence and security informatics conference (EISIC-2011)
13. Martinez V, Simari G, Sliva A, Subrahmanian V (2008) CONVEX: similarity-based algorithms for forecasting group behavior. IEEE Intell Syst 23(4):51–57
14. Martinez V, Simari G, Sliva A, Subrahmanian VS (2008) The SOMA terror organization portal (STOP): social network and analytic tools for the real-time analysis of terror groups. In: Liu H, Salerno J (eds) Proceedings of the first international workshop on social computing, behavioral modeling and prediction. Springer, New York/Berlin
15. Martinez V, Simari G, Sliva A, Subrahmanian V (2009) CAPE: automatically predicting changes in terror group behavior. In: Memon N (ed) Mathematical methods in counterterrorism. Springer, Wien
16. Ng RT, Subrahmanian VS (1992) Probabilistic logic programming. Inf Comput 101(2): 150–201

17. Shakarian J, The CMOT codebook (2012). Available from the laboratory for computational cultural dynamics (LCCD), University of Maryland Institute for Advanced Computer Studies, University of Maryland, College Park
18. Shakarian P, Parker A, Simari GI, Subramanian V (2011) Annotated probabilistic temporal logic. ACM Trans Comput Log 12:14:1–14:44
19. Shakarian P, Simari GI, Subramanian V (2012) Annotated probabilistic temporal logic: approximate fixpoint implementation. ACM Trans Comput Log 13
20. Subrahmanian V, Ernst J (2009) Method and system for optimal data diagnosis
21. Subrahmanian V, Mannes A, Sliva A, Shakarian J, Dickerson JP (2012) computational analysis of terrorist groups: Lashkar-e-Taiba. Springer, New York
22. Wennberg J, Cooper M, Fisher E, Goodman D, Skinner J, Bronner K (1996) The Dartmouth atlas of health care. Dartmouth Institute for Health Policy, Hanover
23. Wilkenfeld J, Asal V, Johnson C, Pate A, Michael M (2007) The use of violence by ethnopolitical organizations in the middle east. Technical report, National Consortium for the Study of Terrorism and Responses to Terrorism