Planning-Based Reasoning for Automated Large-Scale Data Analysis

Anton Riabov, Shirin Sohrabi, Daby Sow, Deepak Turaga, Octavian Udrea, and Long Vu
IBM T.J. Watson Research Center
PO Box 704, Yorktown Heights, NY 10598, USA

Abstract

In this paper, we apply planning-based reasoning to orchestrate the data analysis process automatically, with a focus on two applications: early detection of health complications in critical care, and detection of anomalous behaviors of network hosts in enterprise networks. Our system uses expert knowledge and AI planning to reason about possibly incomplete, noisy, or inconsistent observations, derived from data by deploying an open set of analytics, to generate plausible and consistent hypotheses about the state of the world. From these hypotheses, relevant actions are triggered leading to the deployment of additional analytics, or adaptation of existing analytics, that produce new observations for further reasoning. Planning-based reasoning is enabled by knowledge models obtained from domain experts that describe entities in the world, their states, and relationship to observations. To address the associated knowledge engineering challenges, we propose a modeling language named LTS++ and build an Integrated Development Environment. We also develop a process that provides support and guidance to domain experts, with no planning expertise, in defining and constructing models. We use this modeling process to capture knowledge for the two applications and to collect user feedback. Furthermore, we conduct empirical evaluation to demonstrate the feasibility of our approach and the benefits of using planning-based reasoning in these applications, at large real-world scales. Specifically, in the network monitoring scenario, we show that the system can dynamically deploy and manage analytics for the effective detection of anomalies and malicious behaviors with lead times of over 15 minutes, in an enterprise network with over 2 million hosts (entities).

Introduction

While big data technologies (e.g., Hadoop, stream computing) are addressing the systems aspects of the data overload problem that is experienced in many domains, the interpretation of the big data analytic results for decision making remains a significant challenge. In this paper, we address this challenge while focusing on early detection problems in two problem domains: the early detection of complications in computer networks and the early detection of complications in intensive care units (ICUs). Our approach uses domain knowledge to establish the correspondence between observations about monitored entities (patients or hosts) and their plausible states, coupled with planning-based mechanisms to generate hypotheses about the entity state from noisy, incomplete, inconsistent, time-varying observation sequences, and then use these hypotheses to drive analysis decisions. We develop tools and methodologies for knowledge elicitation from experts (physicians or network analysts) such that these models can be authored without coding effort or knowledge of AI planning.

A patient in typical ICU settings is connected to several monitoring devices that measure different physiological attributes such as the patient's blood pressure, heart rate, and temperature. The analysis of these raw streams of data results in semantically meaningful observations about the patient. For example, given the patient's heart rate, their respiration rate and their body temperature, which are measured continuously, and also their white blood cell count obtained from blood analysis, the Systemic Inflammatory Response Syndrome (SIRS) score (integer that takes values between 0 and 4) can be computed as a meaningful observation about the patient's health. Observations can also include other measurements provided by physicians such as their assessment of patient health or results of lab tests. Similarly, in network monitoring applications, several raw streams of data are available about individual hosts. These include Domain Name Service (DNS) queries, Netflow records, Firewall alerts, Dynamic Host Control Protocol (DHCP) requests, etc. These streams can be analyzed to produce a variety of meaningful observations about host behavior (normal, anomalous, infected) such that hosts behaving suspiciously can be identified in order to protect the network.

Current systems for patient or network host monitoring typically support the deployment of a predefined set of analytics on a pre-selected set of raw data sources and provide little support in terms of adaptive analysis. Recently developed big data platforms (Blount et al. 2010) provide increasing support for scalability, and dynamic adaptation of analysis using extensible sources and analytics. However, in the absence of additional automation, it becomes the responsibility of end-users (analysts and physicians) to decide which analytics to apply when and to which data sources. To address this problem, we propose an automated system that reasons about observations produced by such analytics, and generates hypotheses about monitored entities based on these observations. Our system then groups similar hypotheses and recommends or initiates actions that can include deploying additional analytics (or adapting current analytics) automatically in order to collect additional observations that augment previously incomplete knowledge.

Currently, the primary method to generate hypotheses and evaluate their plausibility is the judgment of the domain expert (physicians or analysts) responsible for monitoring the entity. While model-based diagnosis methods can determine whether observations are explainable by a model (e.g., (Cassandras and Lafortune 1999; Sampath et al. 1995)), we take a different approach, based on automated AI planning, following the work of Sohrabi et al. 2013. In particular, we address two important problems to make the use of AI planning for our application possible: data transformation problem, and knowledge engineering problem. To address the data transformation problem, we tailored for each application several data transformation approaches. To address the knowledge engineering problem we propose a modeling language, LTS++, derived from LTS (Labeled Transition System) (Magee and Kramer 2006), for defining models for hypothesis generation, and associating observation types. We also propose a process to help provide guidance to the domain expert, which often have no planning expertise, to specify the model. In addition, we developed a web-based tool that enables the specification of the LTS++ model and associated observations. Given the LTS++ model and the provided observation trace translated to a planning language, hypotheses are generated by running a planner capable of producing a plan set. Actions are then chosen directly based on the set of generated hypotheses using rules defined in LTS++, without an additional planning stage following the hypothesis generation stage.

We have tested this reasoning approach in our two applications. In both cases, the LTS++ modeling provides a natural and concise way to represent domain knowledge describing the different states in which an entity (host or patient) might be. Our experimental results, while preliminary, show the impact of the use of AI planning technology and in particular demonstrate the ability of our approach to accurately infer the state of entities from real-world observations.

Architecture

The architecture of the system we developed for data analysis automation via planning-based reasoning in both intensive care and network monitoring applications is shown in Figure 1. The system monitors the world consisting of *Entities*. The system receives raw data streams from *Sensors*, and executes actions via *Actuators*. The entities can represent individual patients, or hosts, or other objects. The set of available sensors and actuators may vary by application, and may change after initial deployment. Actuators may recommend actions to be performed by people or other systems. For example, the system can recommend a lab test that, if performed, would return results to the system via sensors.

Analytic Platforms provide distributed execution runtimes for big data analytics. *Online analytics* are deployed on a stream computing platform (IBM InfoSphere Streams,

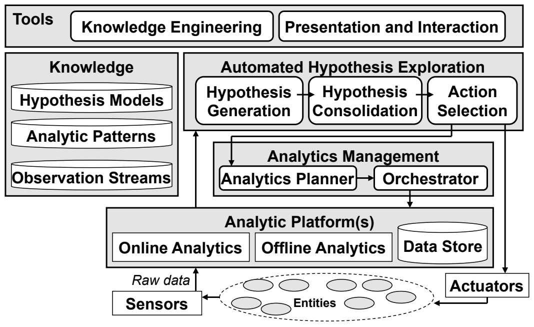


Figure 1: System architecture

in our implementation). *Offline analytics* are deployed on a Map-Reduce platform, for example, Apache Hadoop, and composed into workflows using Apache Pig or Oozie. Analytics can access *Data Stores*, to store or read historical data and results of processing. Selected analytic results are forwarded to Automated Hypothesis Exploration via *Observation Streams*, where each message carries timestamp, observation type and entity reference.

The **Analytics Management** subsystem is responsible for composing and executing analytics in order to achieve analysis goals. The *Analytics Planner* selects and composes analytics necessary to achieve the goal, using semantic descriptions of analytics, their input requirements and composition patterns described in the *Analytic Patterns* repository. The planner may compose analytic workflows that span across multiple analytic platforms, and the *Orchestrator* provides necessary coordination between platforms, and monitors the execution of analytics. The Analytics Management subsystem can be implemented, for example, using a classical AI planner to compose analytics (Bouillet et al. 2009).

The **Automated Hypothesis Exploration** subsystem interprets observation streams received from analytic platforms, generates and consolidates hypotheses, and selects actions. The first step in this process, *Hypothesis Generation*, generates hypotheses about the current state of entities by reasoning about new observations, in view of prior observations, and using *Hypothesis Models* defined by domain experts. Next, *Hypothesis Consolidation* groups similar hypotheses. Finally, *Action Selection* makes decision about actions based on hypothesized state of entities. The selected actions may be forwarded to actuators or used to initiate new data analysis via Analytics Management. For example, analytics can be used to find supporting observations for a hypothesis using historical data captured in Data Store.

These architectural components together form a cycle that continuously processes sensor data, generates hypotheses, investigates hypotheses via additional analysis, and recommends new actions, with support of the **Knowledge** management and representation infrastructure. *Presentation and Interaction* and *Knowledge Engineering* **Tools**, including the LTS++ IDE, allow the end-users to interact with the system.

Hypothesis Generation via Planning

In this section, we formally define the hypothesis generation problem and describe its relationship to planning.

Following (Sohrabi, Udrea, and Riabov 2013), we define the *dynamical system* as $\Sigma = (F, A, I)$, where F is a finite set of fluent symbols, A is a set of planning actions with preconditions and effects that describes actions that account for the possible transitions of the states as well as the *discard* actions that address unreliable observations, and I is a clause over F that defines the initial state. The instances of the *discard* action add transitions to the system that account for leaving an observation unexplained. This ensures that all observations are taken into account, but an instance of the *discard* action for a particular observation indicates that it is not explained. An observation formula φ is a sequence of fluents in F we refer to as *trace*. Note, φ can in general be expressed as an Linear Temporal Logic (LTL) formula (Emerson 1990), however, for our application we consider observations to be totally ordered.

Definition 1 (Hypothesis) *Given a trace φ, and the system description $\Sigma = (F, A, I)$, where F is a finite set of fluent symbols, A is a set of planning actions, and I is a clause over F that defines the initial state, a hypothesis α is a sequence of actions in A such that the hypothesis α satisfies the trace φ in the system Σ.*

We also define a notion of plausibility of a hypothesis. A hypothesis α is at least as plausible as hypothesis α', stated as $\alpha \preceq \alpha'$, where $\preceq$ is reflexive and transitive plausibility relation, if one or more of the following statements hold: α can explain more observations than α', α is a shorter hypothesis, α has minimum number of designated "unlikely" or "bad" actions. The third criteria is similar to the notion of minimum number of "faulty" actions in a diagnostic setting, based on having an optimistic view on what can go wrong.

Definition 2 (Relationship to Planning) *Given a plausibility relation $\preceq$, α is a hypothesis for a system $\Sigma = (F, A, I)$, and a trace φ if and only if α is a plan for the planning problem $P = (F, A', I, \varphi)$ where A' is the set A with the addition of positive action costs that accounts for the plausibility relation $\preceq$. Furthermore, given two hypotheses α and α', $\alpha \preceq \alpha'$ (is at least as plausible as) if and only if both α and α' are plans for the planning problem P, and $cost(\alpha) \leq cost(\alpha')$.*

Hence, we can use AI planning to compute hypotheses, moreover, the most plausible hypothesis is the minimum cost plan. However, it is often not enough to just find one single optimal plan and instead we must find a set of minimum cost plans, or a set of most plausible hypotheses. To compute this set, we use the techniques proposed in (Riabov, Sohrabi, and Udrea 2014) to find a set of high-quality plans in order to drive the hypothesis exploration process.

From Theory to Practice

In the previous section, we established a correspondence between the generation of hypotheses and the generation of plans. This correspondences allows us to use AI planning to generate hypotheses. However, using planning presents at least four challenges we address in this section: 1) how to describe the planning problem from non-experts in planning; 2) how to obtain the observations from raw data; 3) how to make use of the generated hypotheses; 4) how to interact with the domain experts and present the results.

Model Description

In order to allow the domain experts define the domain knowledge we propose a language called LTS++, derived from LTS (Labeled Transition System) (Magee and Kramer 2006). We then translate the knowledge expressed in the LTS++ language and a given set of observations into a planning problem. In our experiments, we used one fixed encoding of the planning domain, (i.e., description of planning actions), but varied the planning problem (i.e., initial state, goal state, and variables) based on the given LTS++ model and observations. Note, in our fixed planning domain, we have actions for explaining or discarding observations as well as transitioning from one state to another. We also encode the notion of plausibility as actions costs. In particular, we assign a high cost to the *discard* action in order to encourage explaining more observations.

We also propose a process that further helps the domain experts in creating a model. Figure 2 shows our 9-step creation process for an LTS++ model. The arrows are intended to indicate the most typical transitions between steps. This process is meant to help provide guidance to the new users in developing an LTS++ model. While this process is geared towards our application, we believe that it also provides insight and inspiration into creation of a practical planning problem. Next, we will first describe the basic elements in the description of a model in LTS++.

Entity: the domain expert needs to identify the entity which is what the system monitors. This depends on the objective of the hypotheses generator, the available data, and the available actions. The entity could be a patient or a host or other objects in the application.

States: the domain expert needs to identify the possible states of the entity (different from a planning state). States are not directly observable but can be hypothesized. The states of patient for example could be Delayed Cerebral Ischemia (*DCI*), *SuspectedDCI*, *Infection*, *Precomplication* or *Highrisk*. States could also have types such as for example, unlikely, or "bad" states. This maps to the notion of faulty or unlikely planning actions as mentioned earlier.

Observations: the domain experts need to identify a set of observation types that the system needs to reason about. Since observations are received from analytics as a result of analyzing raw data, the available data and analytics may limit the space of observations. Heart Rate Variability Low (*OHRVL*, Figure 3), is an example of an observation.

State Transitions: the domain expert has to describe possible transitions between states. An example transition is going from state *Infection* to *Highrisk*. This transition reflects an improvement in patient state, without describing the cause of this transition. An example of a transition system in given in Figure 7.

Association between States and Observations: the domain expert has to associate observations to states meaning that this observation *can* be explained by this state. Note, this association can be many-to-many as observations could be ambiguous or indicative of more than one state, and each state can be associated with multiple observations. The observation *OHRVL* is an example of an ambiguous observation because it can be associated with multiple states.

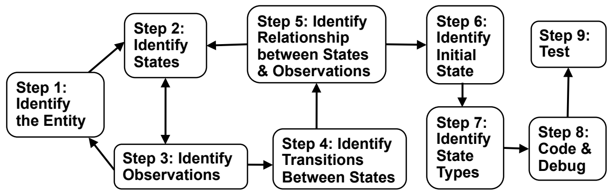

Step 1: Identify the Entity
Step 2: Identify States
Step 3: Identify Observations
Step 4: Identify Transitions Between States
Step 5: Identify Relationship between States & Observations
Step 6: Identify Initial State
Step 7: Identify State Types
Step 8: Code & Debug
Step 9: Test

Figure 2: Process for LTS++ model creation

Association between States and Actions: LTS++ models consisting of elements above are used to automatically generate hypotheses, i.e., plausible sequences of state transitions explaining a received sequence of observations. To specify how the system should act upon generated hypotheses, the domain expert needs to identify actionable states, and specify actions (different from planning actions) triggered by those states. Since observations can be ambiguous, missing, noisy or otherwise unreliable, more than one hypothesis may need to be generated and investigated, possibly triggering multiple actions simultaneously. To resolve potential conflicts, the expert may also define mutual exclusion rules for actions. More details are given in the next subsection.

Figure 3 top part shows part of a LTS++ model description for our healthcare application. The states are shown in blue. The observations are specified within the curly brackets and are shown in green. Multiple observations can be separated by whitespace or a comma. The state types are specified within angle brackets. The transitions between states are specified using arrows. Multiple transitions between states can be specified using a vertical bar. The starting state is specified in the last line. Actions are associated with states in a separate configuration file and are not shown.

Consider the following trace for the model partly shown in Figure 3: *HH1*, *SIRS0*, *SIRS2*. This denotes a patient with a Hunt and Hess (a grading system used to classify the severity of subarachnoid hemorrhage) score of 1, followed by *SIRS* (measure of the inflammatory response of the body) score of 0 and 2. The following are the top four most plausible hypotheses. Figure 4 also shows these hypotheses in our tool. Note, we are showing here only the sequence of state transitions in each hypothesis. *admitted* $\rightarrow$ *lowrisk* $\rightarrow$ *highrisk* or *admitted* $\rightarrow$ *lowrisk* $\rightarrow$ *highrisk* $\rightarrow$ *precomplication* or *admitted* $\rightarrow$ *lowrisk* $\rightarrow$ *highrisk* $\rightarrow$ *lowrisk* $\rightarrow$ *highrisk* or *admitted* $\rightarrow$ *lowrisk*. Although the current state of the patient is unknown, the generated hypotheses indicate that it is one of *highrisk*, *precomplication* or *lowrisk* with *highrisk* being most plausible.

Selecting Actions Based on Hypotheses

We use planning to find a set $\mathcal{H}$ of up to k distinct most plausible hypotheses given a trace of observations φ for an entity. The resulting set $\mathcal{H}$ is used to select actions. Unlike planning actions, which only serve as means of using a planning algorithm for hypothesis generation by establishing a correspondence between plans and hypotheses, the actions we discuss here on have direct impact on the system, and are carried out by Analytics Management or Actuators.

Actions are initiated based on changes in current state of the entity. The current state is computed as follows. Each hypothesis $\alpha \in \mathcal{H}$ defines a non-empty sequence of valid state transitions according to the LTS++ model. The last state $s(\alpha)$ in α represents the last state of the entity inferred from the observation trace.

Observations are received continuously, and the system generates a new set of hypotheses when a new observation arrives. During each such round, actions are selected based on the current last state. For each state s, knowledge engineering tools allow defining the set of actions $u(s)$ and a set of exit actions $\bar{u}(s)$. Actions from $u(s)$ are selected when state s first becomes current in one of hypotheses generated for the entity. Actions from $\bar{u}(s)$ are selected when state s is no longer current for any of the hypotheses.

In practice, some actions may interfere with other actions, and hence are mutually exclusive. To address this, for each action u, a set of conflicting actions $uc(u)$ is defined, and during action selection current states are evaluated in order of plausibility of corresponding hypotheses. Actions that are in conflict with previously selected actions are not selected.

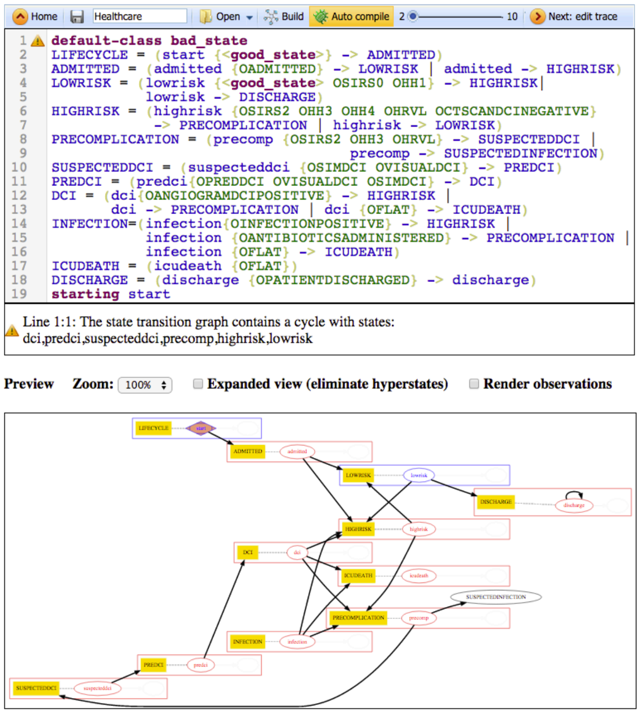
Data Transformation

As shown in Figure 1, the inputs to our system are raw data points sensed in the physical world. Within these data points are often buried nuggets of information that need to be extracted for further reasoning and decision making. In practice, the sensing process often introduces noise that complicates this information extraction process. For instance, in critical care, a badly placed electrocardiogram lead will often produce an electrical signal that masks critical aspects of the electrical activity of the heart that is meant to be measured. To address these issues, we have tailored for each application several data transformation approaches that we describe briefly in this section. Note that this is an important problem we need to solve before applying AI planning. Data transformation problem will only grow in importance as more AI planning techniques are in use in real-world applications that require data processing.

In network anomaly detection applications, the transformation of raw data into information is done by windowing raw data events and extracting statistics form these windows such as event counts, sums, minimal values, maximal values, averages, and medians. The stream of aggregated data values then become the input of network anomaly detection engine, which takes these input values and outputs observations.

Whether an analytic runs in offline mode or online mode, it receives raw data and produces observations. For example, a volumetric-based DNS anomaly detector can run offline on a dataset of DNS requests to detect abnormal hosts, which issue an unexpectedly high number of DNS requests for a certain time window. These hosts can be added into a candidate list of preinfected hosts and relevant observations are stored for further processing and reasoning. On the other hand, a volumetric-based Netflow anomaly detection can run online and receive a stream of aggregated values (e.g., number of Netflow requests per host per hour) as its input and output an observation whether hosts are malicious.

In the critical care application, the input data consists of both events and waveform data streams. Events or lab results such as clinical assessments are sometimes entered manu-



```
default-class bad_state
LIFECYCLE = (start {<good_state>} -> ADMITTED)
ADMITTED = (admitted {OADMITTED} -> LOWRISK | admitted -> HIGHRISK)
LOWRISK = (lowrisk {<good_state> OSIRS0 OHH1} -> HIGHRISK|
           lowrisk -> DISCHARGE)
HIGHRISK = (highrisk {OSIRS2 OHH3 OHH4 OHRVL OCTSCANDCINEGATIVE}
            -> PRECOMPLICATION | highrisk -> LOWRISK)
PRECOMPLICATION = (precomp {OSIRS2 OHH3 OHRVL} -> SUSPECTEDDCI |
                                       precomp -> SUSPECTEDINFECTION)
SUSPECTEDDCI = (suspecteddci {OSIMDCI OVISUALDCI} -> PREDCI)
PREDCI = (predci{OPREDDCI OVISUALDCI OSIMDCI} -> DCI)
DCI = (dci{OANGIOGRAMDCIPOSITIVE} -> HIGHRISK |
       dci -> PRECOMPLICATION | dci {OFLAT} -> ICUDEATH)
INFECTION=(infection{OINFECTIONPOSITIVE} -> HIGHRISK |
           infection {OANTIBIOTICSADMINISTERED} -> PRECOMPLICATION |
           infection {OFLAT} -> ICUDEATH)
ICUDEATH = (icudeath {OFLAT})
DISCHARGE = (discharge {OPATIENTDISCHARGED} -> discharge)
starting start
```

Line 1:1: The state transition graph contains a cycle with states: dci,predci,suspecteddci,precomp,highrisk,lowrisk

Preview **Zoom:** 100% ☐ **Expanded view (eliminate hyperstates)** ☐ **Render observations**

Figure 3: LTS++ IDE

ally by clinicians in the clinical information system. Consequently, they are quite error prone since the primary responsibility of these clinicians is to provide healthcare in these intense environments and not to enter data into these complex clinical systems. Waveform data sensing is inherently noisy despite all the advances in sensing technology that this area has seen in the last couple of decades. The transformation of such waveform data streams into observations requires sophisticated signal processing and feature extraction techniques. It results in observations that are also noisy, thus stressing the importance of being able to reason in the presence of unreliable observations.

LTS++ Integrated Development Environment

LTS++ Integrated Development Environment (IDE) is a web-based tool that helps the domain experts to create model descriptions by describing LTS++ models and to generate hypotheses. LTS++ IDE consists of an LTS++ editor, graphical view of the transition system, specification of the trace, and generation of hypotheses. The tool automatically generates planning problems from the LTS++ specification and entered trace. The generated hypotheses are the result of running our planner and presenting the result from top-most plausible hypothesis to the least plausible hypothesis.

Figure 3 shows the LTS++ IDE. The top part is the language editor, which allows syntax highlighting and the bottom part is the automatically generated transition graph, which can be useful for debugging purposes. The IDE also features error detection with respect to the LTS++ syntax. The errors and warning signs are shown below the text editor and can also be used for debugging the model description.

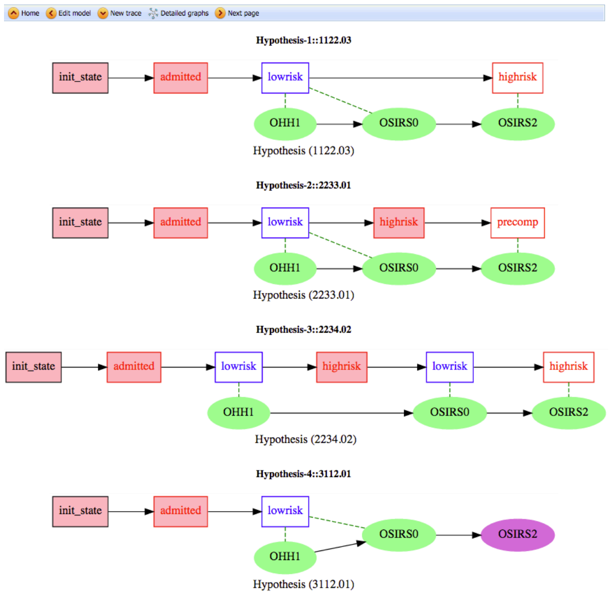


Figure 4: Top 4 hypotheses for the intensive care example

Observations can be entered by clicking on the "Next: edit trace" from the LTS++ IDE main page. Once the trace selection is complete, the hypotheses can be generated by clicking on "Generate hypotheses". The hypotheses are presented to the user 10 per page, and users can navigate through these pages. Note, the trace editor is intended mainly for testing purposes, and in operation the system will read observations automatically from an input queue.

Fig. 4 shows the top 4 automatically generated hypotheses for a sample trace [*OHH1 OSIRS0 OSIRS2*]. Each hypothesis is shown as a sequence of states (shown in rectangles) matched to observed event sequence. The explained observations are shown in green ovals and are connected to the state that explained them by dashed green lines. The unexplained observations are shown in purple ovals. The arrows between the observations show the sequence of observations in the trace. Each hypothesis is associated with a cost. The lower the cost value, the more plausible is the hypothesis.

Experimental Results

Our experimental evaluation consists of two parts. In the first part, we provide a quantitative evaluation of the proposed system for a network anomaly detection application. The second part presents a qualitative evaluation of the system for the early detection of complications in intensive care.

Network Anomaly Detection

We implement a network anomaly detection system running on 18 months of real-world network data collected from an enterprise network. Our system works with 2M hosts, where 40K hosts are internal hosts of our network and the rest are external hosts. The input network datasets include Netflow, DNS, and Firewall. The Netflow dataset includes the "time stamp" when the flow starts, source IP, and destination IP. The Firewall dataset includes the "time stamp" when the

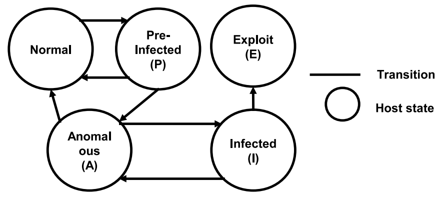


Figure 5: Host State Transition Model

firewall alert was issued, host IP that causes the alert, and the alert level. The DNS dataset includes the "time stamp" of the DNS request, DNS client IP, and the requested URL.

To represent different states of network hosts, we design a host state transition model as shown in Figure 5. Specifically, a host may be in one of five states: Normal, Pre-Infected (host that communicates with Infected or Exploit hosts), Anomalous (host with abnormal activity but with unclear evidence to conclude that it is in Infected or Exploit state), Infected (host with behavior indicative of infection), and Exploit (infected host that performs abnormal activities against other hosts). We then encode our host state transition model in the LTS++ language.

The raw incoming network data rate is 75K messages/second (i.e., incoming rate per detector is 5K messages/second) and messages are processed by network anomaly detectors in Table 1, which send the observations to the LTS++ servers. These observations are associated with different states of the host transition model as specified in Table 1. For instance, a source performing a port scan is likely in Exploit state, while a destination being port scanned is likely in Pre-Infected state. Our network anomaly detectors are written in Stream Processing Language (SPL) and run on IBM InfoSphere Streams, a platform for big data stream computing. Each detector focuses on specific features derived from one of three above datasets to detect network anomalies. The first six detectors in Table 1 are deployed at system startup to identify well-known patterns of Infected, Exploit or Pre-Infected behavior. Additional analytics - such as expensive anomaly detection on Netflow (e.g., analytic 7 in Table 1) are triggered by the LTS++ server for deeper analysis of suspicious hosts. We use three metrics to evaluate our system: (1) Early Detection: How early the system can put hosts in Pre-Infected state in advance before they are detected as Exploit, (2) Workload Dynamics: number of hosts analyzed per hour, (3) Analysis Adaptation Time: how long it takes for the system to deploy new analytics on suspicious hosts. We deploy the system on a cluster of 6 nodes and each run is for 24 hours of network data. We verify the results with multiple system runs for consistency.

For our experiments, we set the server's limit on number of hypotheses to 5. We find that the number of plans generated by our servers is from 100 to 2600 per minute and the LTS++ server receives from 100 to 10K observations per minute. Figure 6(a) shows that about 80% of hosts were labeled Pre-Infected with a lead time of one hour before they are detected as Exploit. For 5% of Exploit hosts, this lead time can be up to 10 hours. These results confirm that our system can provide early network anomaly even with a simple state transition model, and with a small number of

	Detector Name	Dataset	Src	Dst
1	Port scan vertical[1]	Netflow	E	P
2	Port scan horizontal[2]	Netflow	E	P
3	Distributed port scan[3]	Netflow	E	P
4	Malicious URL	DNS	P	-
5	Firewall high alert	Firewall	I	-
6	Firewall low alert	Firewall	P	-
7	High volume at src	Netflow	A	-

Table 1: Implemented Network Anomaly Detectors

anomaly detectors. Note that our current detectors can detect around 40% of Exploit hosts, and this can be improved with other detectors. Figure 6(b) shows that the number of analyzed hosts per hour by the system varies from 4,000 to 34,000. In practice, the number of active network hosts varies over time and that contributes to this fluctuation. This result confirms that our system works well with dynamic workloads at large scale. Figure 6(c) shows that about 80% of deployments of new analytics happen within 3 seconds and about 90% of deployments happen within 5 seconds. This means our system can produce hypotheses and trigger relevant actions in a timely fashion.

In summary, our system efficiently combines clues from multiple sources, quickly generates hypotheses, and dynamically deploys actions to detect network anomalies at large-scale. Moving forward, we plan to: extend our set of detector analytics, improve the host state transition model to capture other host behaviors, and deploy on the live network.

Qualitative Evaluation With Intensive Care Application

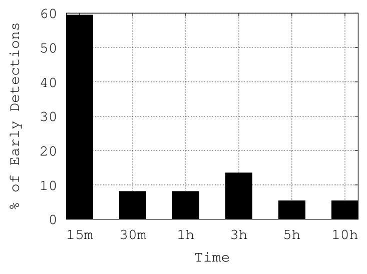
In 2003, critical care physicians have reported that they have to handle over 200 potentially temporal variables per patient to provide care (Imhoff et al. 2003). It is believed that this number has at least doubled if not tripled in the last decade. While big data analytic platforms are currently used to provide solutions to this data overload problem in critical care (Blount et al. 2010), the analysis complexity remains overwhelming. To provide situational awareness, domain experts need to work with IT analysts to decide on which data sources to tap into, what questions/analysis to ask/perform on the data, how to best use and tune existing analytics/systems/platforms to perform that analysis, when/how to abandon one investigation path and advance on another, how to best correlate information from multiple investigation paths, how to find evidence in support of hypotheses and how to rank them, how to act on the evidence. All of these operations need to be done in context with the current state of their patients.

Using the techniques described in this paper, we have developed an advanced clinical decision support system that addresses a subset of these challenges and helps bring situational awareness to the bedside. This system estimates the current state of patients from observations produced by the analysis of patient monitoring data such as electrocardiograms (ECG), discrete values of heart rate, and respiration

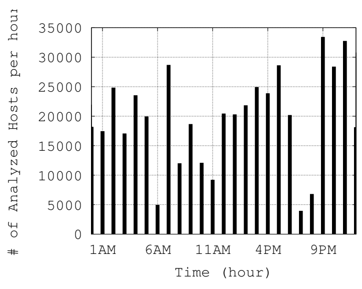
[1]a host being scanned by another host for multiple ports

[2]scan against a group of hosts for a single port

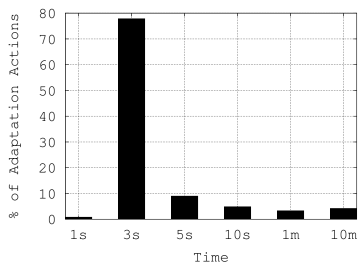
[3]a host being scanned by group of hosts for a single port



(a) Early Detection



(b) Workload Dynamics



(c) Analysis Adaptation Time

Figure 6: Our system provides network anomaly detection in a timely fashion and at large-scale.

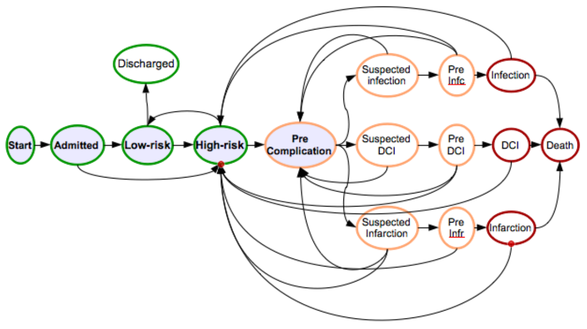


Figure 7: Patient State Transition Model

rates. The system ingests these observations in real-time and reasons on them to hypothesize patient states that are indicative of secondary complications. Accurate state estimations opens the door for strategic planning of analytic actions and recommendations in anticipation of complications. This approach promises to help transforming critical care monitoring from being reactive to patient problems to becoming more proactive by flagging abnormal patient states as early as possible to clinicians. Figure 7 represents a reasoning model for subarachnoid hemorrhage patients in neurological intensive care that we have implemented in LTS++. Part of the LTS++ encoding of this model is shown in Fig 3.

While formal user studies are needed to assess the effectiveness of the language and the tool we have designed, we have received informal feedback from medical professional on the expressiveness of the language. Overall, the language has been found to be extremely simple and intuitive. Beyond the model, the concepts such as states or observations were found to be natural. They bridge well the gap between the complexity of the underlying planning-based reasoning and the knowledge of the domain experts. During working sessions with physicians, we have found this representation approach to mimic how physicians think about their patients. In some instances, we were asked to provide an explicit ability for the system to allow patients to be simultaneously in more than one state. For instance, a patient could very well be developing both an infection and be in an ischemic state. We expect the current model to evolve into a set of models that can be composed to address this challenge.

The directed graph shown in Figure 7 represents the different states that a patient may be in. As a patient navigates through this graph, there are essentially three types of states that he or she may visit: (1) Green states that represent good states not associated with any potential complications; (2) Orange states that represent warning states indicative of a deterioration of health; (3) Red states that represent either complication states or bad outcome states including death.

Upon arrival in critical care, patients are triaged either as low risk or high risk patients mainly based on the results of health assessment scales like the Hunt and Hess scale or the Glasgow Coma scale. These scales are the results of physical exams assessing severity of the injury, including the patient's level of consciousness and the strength of their headaches. From the low risk state, a patient may be discharged from the ICU. From the high risk state, a patient may enter a pre-complication state defined as a state where a general deterioration of health becomes noticeable. At this point, the patient may develop specific complications. The rest of this model focuses on three complications that are quite common in neurological ICUs. The first one is sepsis which is a blood infection typically acquired in hospitals and with a mortality rate up to 40%. The second one is delayed cerebral ischemia (DCI) which is a condition with an equally high mortality rate often the result of an abnormal blood flow in the brain that can lead to tissue death and irreversible damages. The last one is infarction which is often related to DCI.

For each of these complications, a suspected state follows the pre-complication state. These suspected states are entered when the patient is evaluated to be similar to historical patients that developed the same complication according to a similarity metric learning scheme proposed in (Sun et al. 2010). The suspected states are followed by the corresponding early complication states (pre-infection, pre-dci and pre-infarction). In these states, the patient has developed the complication but clinical symptoms for it are not visually apparent. The complications may have been confirmed with lab tests, angiograms or CT scans. Following these early complication states are the complication states where physical symptoms are visible and if left untreated, the patient may end up in the death state.

We have begun testing the efficacy of this model retrospectively on a real critical care patient data set consisting of 8 patients, 6 of which did develop complications. Of these 6 patients, 2 of them had DCI complications while the remaining 4 patients suffered from hospital-acquired infections. For these experiments, time series of body temperature and respiration rate sampled at 0.5 Hz were used together with Electrocardiogram waveforms sampled at 240 Hz. Observations were generated by replaying the data of these patients though a stream computing system equipped with analytics able to compute several Heart Rate Variability (HRV) metrics from Electrocardiograms and the Systemic Inflammatory Response Syndrome (SIRS) score from heart rate, respiration rate, body temperature and white blood cell count measurements. HRV metrics are known to evaluate the state of the autonomic nervous system which is highly correlated to early onset of complications in neuro-ICUs (Schmidt et al. 2014). The SIRS score is a measure of the inflammatory response of the body. It is known to be associated with infections like sepsis. In addition, the results of several clinical tests typically performed upon admission were also used to generate observations that were fed to the reasoning engine.

It is important to note that the validity of these early experiments are hindered by both our inability to actuate and change the physiological trajectory of our retrospective patients and also by the interventions that were taken by clinicians on these patients while they were in the ICU. These interventions clearly affect the physiological readings and the observations that we are producing retrospectively. Nevertheless, we are obtaining interesting results demonstrating the effectiveness of this simple model. For instance, while all patients did visit precomplication and sometimes suspected complication states, they did so at very different rates. Patients suffering from complications are more often found to be in the suspected states while patients who did not suffer from complications spent the majority of their time between the low/high grade states and precomplication states.

We are currently in the process of refining this model with domain experts to add more states, more observations from more analytics and performing more experiments on a much larger data set. We hope to be able to test this system prospectively in a live environment to be able to assess the efficacy of this approach to patient monitoring.

Related Work

Network anomaly detection is becoming central to effective network monitoring due to the advancements in network infrastructures and technologies, as well as the increase in volume and sophistication of network attacks. Although numerous network anomaly detection solutions have been proposed (Bhuyan, Bhattacharyya, and Kalita 2014; Chandola, Banerjee, and Kumara 2009), they have failed to fully resolve issues with dynamic adaptation of analysis, linking anomalies with domain relevant events, and operating at large-scale. First, many solutions use rule-based approaches where rules are either manually created or learned from previously labeled data, and these rules are both hard to adapt and insufficient to capture new anomalies. Second, current solutions often use a single network source (e.g., either DNS data or Netflow data, but not both) and remain unable to efficiently leverage evidence from multiple sources to discover anomalies. In this paper, we propose and implement a system that provides an automatic and adaptive solution towards large-scale network anomaly detection. Specifically, our system first quickly blends clues about behavior of hosts from different network sources. Then, the system effectively utilizes a reasoning tool that uses these clues to generate hypotheses about the state of individual hosts and the network, and then adaptively deploys further investigation analytics on suspicious hosts.

In medical informatics, a rich body of literature reports on the clinical decision support system (CDSS) that are defined by Robert Hayward from the Centre for Health Evidence as “systems [that] link health observations with health knowledge to influence health choices by clinicians for improved health care”. The healthcare application presented in this paper relates to CDSS systems applied to critical care. There are several CDSS developed for critical care. A notable example is BioStream (Bar-Or et al. 2004), a system for real-time analysis and interpretation of physiological signals. Also in (Blount et al. 2010), the authors developed an open platform for streaming analytics of critical care patient physiological data intended to help physicians detect in real-time patient complications. While these systems intend to provide knowledge to physicians to help them in their decision making process, they are all purely data driven. None of them make use of domain knowledge to reason on physiological data streams and hypothesize on patient states.

Our approach in using AI planning to generate hypotheses is related to several approaches in the diagnosis literature (e.g., (Sohrabi, Baier, and McIlraith 2010; Bauer et al. 2011)), but is most closely related to (Sohrabi, Udrea, and Riabov 2013) as we also need to deal with incomplete and inconsistent knowledge. However, we also tackle two important problems, not previously addressed, knowledge engineering, and data transformation, making it possible to apply AI planning to our data-intensive applications.

Conclusions

In this paper, we apply planning-based reasoning for data analysis focusing on the early detection problem. To this end, we introduced a novel system architecture for large-scale analysis automation. The distinguishing characteristic of this architecture is its ability to analyze observations, generate hypotheses and launch new analytics or initiate actions based on the hypotheses, while making use of domain knowledge expressed in an easy to specify language, LTS++. The main contributions of this paper are: (1) building a system that automates data analysis by reasoning about unreliable observations using AI planning and expert knowledge (2) applying data transformation techniques to obtain observations, and input to our system by analyzing the raw data; (3) developing a model description language that captures the expert knowledge and a web-based tool that implements our approach and allows interaction with the proposed system, and (4) evaluating the system on real-world problems from different domains to showcase the resulting scale, latency, and improved monitoring accuracy achieved by this automation. We also performed an informal user study on the approach and use of LTS++. Initial feedback received from domain experts indicates that LTS++ provides a natural and concise way to capture domain knowledge.

References

Bar-Or, A.; Goddeau, D.; Healey, J.; Kontothanassis, L.; Logan, B.; Nelson, A.; and Van Thong, J. 2004. Biostream: A system architecture for real-time processing of physiological signals. In *IEEE Engineering in Medicine and Biology Conference*.

Bauer, A.; Botea, A.; Grastien, A.; Haslum, P.; and Rintanen, J. 2011. Alarm processing with model-based diagnosis of discrete event systems. In *Proceedings of the 22nd International Workshop on Principles of Diagnosis (DX)*, 52–59.

Bhuyan, M. H.; Bhattacharyya, D. K.; and Kalita, J. K. 2014. Network anomaly detection: Methods, systems and tools. *IEEE Communications Surveys and Tutorials* 16.

Blount, M.; Ebling, M. R.; Eklund, J. M.; James, A. G.; McGregor, C.; Percival, N.; Smith, K. P.; and Sow, D. 2010. Real-Time Analysis for Intensive Care: Development and Deployment of the Artemis Analytic System. *IEEE Engineering in Medicine and Biology Magazine* 29:110–118.

Bouillet, E.; Feblowitz, M.; Feng, H.; Ranganathan, A.; Riabov, A.; Udrea, O.; and Liu, Z. 2009. Mario: middleware for assembly and deployment of multi-platform flow-based applications. In *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware (Middleware)*, 26:1–26:7.

Cassandras, C., and Lafortune, S. 1999. *Introduction to discrete event systems*. Kluwer Academic Publishers.

Chandola, V.; Banerjee, A.; and Kumara, V. 2009. Anomaly detection: A survey. *ACM Computing Surveys (CSUR)* 41(15).

Emerson, E. A. 1990. Temporal and modal logic. *Handbook of theoretical computer science: formal models and semantics* B:995–1072.

Imhoff, M.; Fried, R.; Gather, U.; and Lanius, V. 2003. Dimension reduction for physiological variables using graphical modeling. *AMIA Annu Symp Proc.* 313–7.

Magee, J., and Kramer, J. 2006. *Concurrency - state models and Java programs (2. ed.)*. Wiley.

Riabov, A.; Sohrabi, S.; and Udrea, O. 2014. New algorithms for the top-k planning problem. In *Proceedings of the Scheduling and Planning Applications woRKshop (SPARK) at the 24th International Conference on Automated Planning and Scheduling (ICAPS)*, 10–16.

Sampath, M.; Sengupta, R.; Lafortune, S.; Sinnamohideen, K.; and Teneketzis, D. 1995. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control* 40(9):1555–1575.

Schmidt, J. M.; Sow, D.; Crimmins, M.; Albers, D.; Agarwal, S.; Claassen, J.; Connolly, E. S.; Elkind, M. S. V.; Hripcsak, G.; and Mayer, S. A. 2014. Heart rate variability for preclinical detection of secondary complications after subarachnoid hemorrhage. *Neurocritical Care* 20:3:382–9.

Sohrabi, S.; Baier, J.; and McIlraith, S. 2010. Diagnosis as planning revisited. In *Proceedings of the 12th International Conference on the Principles of Knowledge Representation and Reasoning (KR)*, 26–36.

Sohrabi, S.; Udrea, O.; and Riabov, A. 2013. Hypothesis exploration for malware detection using planning. In *Proceedings of the 27th National Conference on Artificial Intelligence (AAAI)*, 883–889.

Sun, J.; Sow, D. M.; Hu, J.; and Ebadollahi, S. 2010. A system for mining temporal physiological data streams for advanced prognostic decision support. In *Proceedings of IEEE 10th International Conference on Data Mining series (ICDM)*, 1061–1066.