DISSERTATION

LEARNING AND PREDICTION OF RELATIONAL TIME SERIES

by

Kian-Moh Terence Tan

March 2013

Dissertation Supervisor: Christian Darken

This thesis was performed at the MOVES Institute
Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** March 2013	**3. REPORT TYPE AND DATES COVERED** Dissertation	
4. TITLE AND SUBTITLE LEARNING AND PREDICTION OF RELATIONAL TIME SERIES			**5. FUNDING NUMBERS**
6. AUTHOR(S) Kian-Moh Terence Tan			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ____N/A ____.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

Prediction of events is fundamental to both human and artificial agents. The main problem with previous prediction techniques is that they cannot predict events that have never been experienced before. This dissertation addresses the problem of predicting such novelty by developing algorithms and computational models inspired from recent cognitive science theories: conceptual blending theory and event segmentation theory. We were able to show that prediction accuracy for event or state prediction can be significantly improved using these methods.

The main contribution of this dissertation is a new class of prediction techniques inspired by conceptual blending that improves prediction accuracy overall and has the ability to predict even events that have never been experienced before. We also show that event segmentation theory, when integrated with these techniques, results in greater computational efficiency. We implemented the new prediction techniques, and more traditional alternatives such as Markov and Bayesian techniques, and compared their prediction accuracy quantitatively for three domains: a role-playing game, intrusion-system alerts, and event prediction of maritime paths in a discrete-event simulator. Other contributions include two new unification algorithms that improve over a naïve one, and an exploration of ways to maintain a minimum-size knowledge base without affecting prediction accuracy.

14. SUBJECT TERMS Relational time series, learning, prediction, conceptual blending, Cyber intrusion alert			**15. NUMBER OF PAGES** 235
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

LEARNING AND PREDICTION OF RELATIONAL TIME SERIES

Kian-Moh Terence Tan
Civilian, Singapore
B.S., University of London, 2000
M.S., Naval Postgraduate School, 2007
M.S. National University of Singapore, 2008

Submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY IN MODELING, VIRTUAL ENVIRONMENTS AND SIMULATION (MOVES)

from the

NAVAL POSTGRADUATE SCHOOL
March 2013

Author: Kian-Moh Terence Tan

Approved by:

Dr. Christian Darken
Assoc. Prof. of Computer Science
Dissertation Supervisor

Dr. Neil Rowe Prof. of Computer Science	Dr. Arnold Buss Assoc. Prof. of MOVES
Dr. Ralucca Gera Assoc. Prof. of Applied Math.	Mr. John Hiles Retired Prof. of MOVES

Approved by:
Christian Darken, Chair, MOVES Academic Committee
Peter J. Denning, Chairman, Department of Computer Science

Approved by:
Douglas Moses, Vice Provost, Academic Affairs

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Prediction of events is fundamental to both human and artificial agents. The main problem with previous prediction techniques is that they cannot predict events that have never been experienced before. This dissertation addresses the problem of predicting such novelty by developing algorithms and computational models inspired from recent cognitive science theories: conceptual blending theory and event segmentation theory. We were able to show that prediction accuracy for event or state prediction can be significantly improved using these methods.

The main contribution of this dissertation is a new class of prediction techniques inspired by conceptual blending that improves prediction accuracy overall and has the ability to predict even events that have never been experienced before. We also show that event segmentation theory, when integrated with these techniques, results in greater computational efficiency. We implemented the new prediction techniques, and more traditional alternatives such as Markov and Bayesian techniques, and compared their prediction accuracy quantitatively for three domains: a role-playing game, intrusion-system alerts, and event prediction of maritime paths in a discrete-event simulator. Other contributions include two new unification algorithms that improve over a naïve one, and an exploration of ways to maintain a minimum-size knowledge base without affecting prediction accuracy.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

RTS	Relational Time Series
CBT	Conceptual Blending Theory
EST	Event Segmentation Theory
SLT	Statistical Lookup Table
VM	Variable Matching
MSB	Multiple Simple Bayesian
SBM	Simple Bayesian Mixture
VOMM	Variable-Order Markov Model

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

This dissertation would not be possible without the professional and academic guidance, help and support from my dissertation supervisor and PhD committee members. I am most grateful to Dr. Christian Darken for his patience and guidance from the identification of the research problem to the actual research work, to the dissertation defense, and finally, the writing of this dissertation. Without his profound knowledge in the field of artificial intelligence, I would have taken a much longer time to complete the research objectives.

I would like to thank my PhD committee members for their proactive support in the course of my research work. Without Dr. Neil Rowe, I would not be able to demonstrate the utility of my research in the cyber security domain. I am grateful for cyber intrusion alert data provided by Dr. Rowe's honeypot, and also for his meticulous readings and comments on my dissertation. I would like to thank Dr. Arnold Buss for his proactive support and guidance for helping me to apply my research work on Simkit, the most widely used discrete event simulation in the Naval Postgraduate School. I also would like to thank Dr. Ralucca Gera for her friendly guidance on the most important mathematical theory that underpins my research work. Her advice on the most advanced concept on graph theory has led to the reduction in time complexity in my algorithm development. Additionally, I would like to thank retired Professor John Hiles for introducing the conceptual blending theory and providing his continual guidance despite his retirement.

Above all, I would like to thank my wife, Hwee-Choo Serene Tan, for her love and support. She not only has supported me, but also assumed the primary role of taking care of our three children: Sheryl, Sherry and Timothy.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

Prediction of events is fundamental to both human and artificial agents. Many prediction techniques exist. The main problem with many prediction techniques is that they cannot predict events that have never been experienced before. Our approach to solving the problem is inspired by theories of cognitive science. Before we go into the details, we will start by defining some of the terms that we use throughout the dissertation, followed by a brief discussion on the motivations of prediction and backgrounds of this dissertation. We then describe the dissertation problems and objectives, and the methodology for solving the problem.

B. TERMINOLOGY

We first define some terminology that will be used in this dissertation.

1) Timed percept: A timed percept is defined as

$p = r(c_1,c_2,\ldots,c_{m,}$ t, type) where

- r is the predicate.
- c_i for $i \in [0..m]$ are constants that represent actors, location, or environment objects, related by r.
- t is the time when p is received.
- type $\in \{e, a, +, -\}$ describes the type of timed percept where 'e' means p is a point timed percept that describes an event, 'a' means p is a point timed percept that describes an action, '+' means p marks the beginning of an interval timed percept, '-' means p is a cancelling point timed percept (cancelling percept in short) that remove a corresponding interval timed percept.

A point timed percept is a timed percept that happens at one point in time and its assertion ceases to be guaranteed. A timed percept that describes an occurrence of an event or an action is a point timed percept. For example, a timed percept that describes “a ball hits the wall” becomes false immediately after it occurs.

An interval timed percept occurs at the ‘+’ percept, persist for an interval amount of time and is removed by a corresponding ‘-‘ percept. An interval timed percept occurs and remains true until something happens that changes its state to false. An interval timed percept contains a state that has a piecewise constant trajectory. For example, a percept that describes “a ball is in the box” is true until the ball is removed.

A timed percept indicating the beginning of an interval state has a ‘+’ indicator in the predicate such as $p = r(c_1,c_2,\ldots,c_m, t, +)$ where r is the predicate, c_i for $i\in[0..m]$ are constants. The interval timed percept becomes false when a special type of point timed percept arrives, indicated by ‘-‘ in the predicate such as $p = r(c_1,c_2,\ldots,c_m, t, -)$ where r is the predicate, c_i for $i\in[0..m]$ are constants. When type is empty, the timed percept is a point timed percept.

A timed percept is therefore a set of constants that are related by a predicate and occurs at a particular time. A timed percept is the smallest unit of data perceived by an artificial agent. Timed percepts may contain updates of states, actions taken, or events occurring in the world. Timed percepts may contain real number or categorical constants.

2) Simplified percept: A simplified percept p_s is derived from a timed percept p_t through the homomorphism function f: $(p_t = r(c_1,c_2,\ldots,c_m, t, type)) \rightarrow (p_s = r(c_1,c_2,\ldots,c_m, type))$

A simplified percept is a homomorphism of a timed percept such that the time of occurrence of the timed percept is discarded. Multiple timed percepts that are only different on time will be mapped into the same simplified precept. Each simplified percept has a corresponding timed percept. Other approaches to prediction could discard space or other arguments if they wanted to make less specific predictions (but hence predictions more often true).

In this dissertation, when a percept mentioned without ‘timed’ or ‘simplified’, it refers to a simplified percept.

3) Relational time series: A relational time-series is a sequence of timed percepts: $p_1p_2\ldots p_n$. If t_i is the time of timed percept p_i, the following holds: $t_{i-1} \leq t_i \leq t_{i+1}$. An example of relational time-series is given in Figure 1. .

P_i	Percepts	Descriptions
P_1	`loc(Ed,   road, 1, +)`	Ed is at location road
P_2	`loc(Fox1, road, 2, +)`	Fox1 is at location road
P_3	`go (Fox1, east, 3, e)`	Fox1 go east
P_4	`loc(Fox1, road, 5, -)`	Fox1 is NOT at location road
P_5	`loc(Fox2, road, 10, +)`	Fox2 is at location road
P_6	`go (Fox2, east, 11, e)`	Fox2 is going east
P_7	`loc(Fox2, road, 13, -)`	Fox2 is NOT at location road

Figure 1. An Example of Relational Time-series.

4) Current time and current percept: Given a relational time-series $p_1p_2\ldots p_n$ and a time sequence $t_1t_2\ldots t_n$ such that timed percept p_i occurs at time t_i, the current timed percept is p_n and the current time is defined as t_n.

5) Next percept: Given a relational time-series $p_1p_2\ldots p_n$ and a time sequence $t_1t_2\ldots t_n$ such that timed percept p_i occurs at time t_i, the next simplified percept is p_{n+1}

6) Next percept prediction: A prediction can be defined as an expected simplified percept that the agent will receive at a future time. Each prediction should have a time restriction and space restriction to be fair. In this dissertation, we use the same definition in Sun and Giles [1] defined in the paragraph that follows.

A next simplified percept prediction problem is $p_1p_2\ldots p_n \vdash p_f$ where $p_1p_2\ldots p_n$ is a sequence of timed percept, $\vdash$ is an operator that expresses that simplified percept p_f is the predicted next simplified percept given the current timed percept history.

We call the predicted next simplified percept as predicted percept.

A predicted percept is by default a simplified percept that has no corresponding timed percept because the predicted percept is a hypothetical percept.

A predicted percept may have a predicted time component that indicates restrictions on the predicted time of occurrence for the predicted-percept.

7) Correct next percept prediction: A predicted simplified percept is said to be correct if the prediction occurs at an expected time or space. In this dissertation, we use a more restricted metric that disregards the expected time and space prediction. The predicted simplified percept $p_p = r_p(\mathrm{c_{p1}}, \mathrm{c_{p2}}, \dots, \mathrm{c_{pm}}, \mathrm{IP_p})$ is said to be correct if the next simplified percept $p_n = r_n(\mathrm{c_{n1}}, \mathrm{c_{n2}}, \dots, \mathrm{c_{nm}}, \mathrm{IP_n})$ is such that $p_p = p_n, r_p = r_n, \mathrm{c_{p1}} = \mathrm{c_{n1}}$ for all $\mathrm{i} = 1,2, \dots, \mathrm{m}$, $\mathrm{IP_p} = \mathrm{IP_n}$ where r_n and r_p are predicates, $\mathrm{c_{p1}}$ and $\mathrm{c_{n1}}$ are constants in the predicate, and $\mathrm{IP_p}$ and $\mathrm{IP_n}$ are the types of simplified percept. Time of timed percept occurrence is ignored.

8) Situation: A situation is a set of point and interval percepts such that, the point percepts, excluding the cancelling percept, occur in a small fixed time window and the interval percepts that has no corresponding cancelling percept. Given a relational time-series $p_1p_2 \dots p_n$ that occurs at time $t_1t_2 \dots t_n$, and a time window t_w, a situation is formed by the set of simplified percepts $\{H, p_r, p_{r+1}, \dots, p_{r+m}\}$ if $t_r \le t_{r+1} \le \dots \le t_{r+m}$, $(t_{r+m} - t_r) \le t_w$ where H is a set of interval simplified percepts from $p_1p_2 \dots p_{r-1}$ that has not encountered the corresponding '-' percept, and that $p_r, p_{r+1}, \dots, p_{r+m}$ cannot include contradictory percepts, and the most recent percept will remove earlier contradictory percepts. $p_r, p_{r+1}, \dots, p_{r+m}$ cannot contain percept of type '-' and the corresponding interval percept must be removed.

In real problem, an agent may receive timed percepts that are not of interest to the prediction problem. We assume that all timed percepts have been preprocessed by an external process so that all timed percepts received by the agent are relevant to the prediction task.

9) Situation-based prediction: A prediction problem is $s_i \vdash p_c$ where $\vdash$ is an operator that predict that p_c is the next predicted percept after the current situation s_i.

10) Target percept: A target percept is a next percept of a situation.

11) Situation-target tuple: A situation-target tuple is a (situation, next-simplified percept-target) tuple defined as $st_i = (s_i, t_i)$ where s_i is a situation and t_i is a set of simplified percepts such that the corresponding timed percepts of t_i are the next simplified percept of the corresponding timed percepts of s_i. The simplified percepts in t_i are known as target simplified percepts. We will call the (situation, next-simplified percept-target) tuple as situation-target in short.

12) Exact matching: Two situations $s_1 = \{p_1^1, p_2^1, \dots, p_n^1\}$ and $s_2 = \{p_1^2, p_2^2, \dots, p_n^2\}$ exactly match if $p_1^1 = p_1^2, p_2^1 = p_2^2, \dots, p_n^1 = p_n^2$, where the p terms are simplified percepts

13) Matching by unification: Two situations $s_1 = \{p_1^1, p_2^1, \dots, p_n^1\}$ and $s_2 = \{p_1^2, p_2^2, \dots, p_n^2\}$ are said to have matched by unification if $SUBST(\theta, p_1^1) = p_1^2, SUBST(\theta, p_2^1) = p_2^2, \dots, SUBST(\theta, p_n^1) = p_n^2$ where θ is a set of unification $p_1^1 : p_1^2, p_2^1 : p_2^2, \dots, p_n^1 : p_n^2$ where ':' is a binding operator.

14) Current situation: The current situation is the situation that contains the current simplified percept.

15) Novel percept: Given a set of previously encountered simplified percepts C = $\{p_1, p_2, \dots p_n\}$ and a current simplified percept p_c, we say that p_c is novel if $p_c \notin C$.

A simplified percept can be said to be novel if it has no exact match with any previous simplified percept, even if the object constants are of the same type. For example, suppose we have previously encountered simplified percept p_1: color-white(car1). Sometime in the future, we encountered another simplified percept p_2: color-white(car2). We say that p_2 is a novel percept because it is not the same as p_1.

C. MOTIVATIONS

1. Prediction Is Important

Decision making plays a major role for both human and artificial agents. Kurby and Zacks [2] discovered from their neuroscience studies that human agents make decisions based on current and predicted future states. Furthermore, the ability to predict future events and to act based on the predicted states can enhance the fidelity of agent behavior models. Kunde and Darken [3] showed that prediction capability can enhance the realism of an artificial military agent when the agent was able to delay a call for fire action at the moment when one adversary tank of a convoy was sighted by predicting that there are more enemy tanks in the convoy. Prediction capability enables an agent to manipulate its current environment to do better in the future rather than simply react to events.

In addition to an agent in artificial environments, relational time-series are also found in cyber intrusion-detection system [4] such as Snort [5]. In cyber intrusion-detection system, cyber intrusion activities are captured as intrusion alerts. When alerts occur, damages may have already occurred. Hence, it is imperative to prevent certain high priority alerts from occurring. These alerts are represented in a relational table form, which can be converted into a relational time series. The ability to predict the next alert may help the network system administrator to better prepare the network for future attack.

Ship movement can also be expressed as a relational time-series if we represent the continuous movement by discrete spaces such as a rectangular grid. We can then capture a sequence of ship related events as relational time series. We can therefore predict ship movements in order to differentiate between normal and suspicious ship behavior. Given a suspicious ship, we can also attempt to predict its future position in for interdiction plan.

Relational time-series can also be found in discrete event simulation engine [6], [7] as sequence of events. Many software models are created on discrete event simulation engine for system engineering studies. Given prediction capability, the software model can use prediction to design anticipatory decision support system or use prediction to improve the fidelity of software human agent.

2. Prediction Is Hard

Sun and Giles [1] discussed several significant issues in sequence learning. The first is the many existing models (recurrent network, reinforcement learning and heuristic methods) cannot handle temporal dependency in which the next percept may depend on the current percept or a percept that occurred a way before the current percept. The second issue is hierarchical structuring of sequences, in which a sequence consists of subsequences. A third issue concerns noisy sequences.

In addition, the environmental behavior that an agent is tasked to learn can be unknown to the designer. Therefore, learning and prediction of relational time-series from environments that are characterized as unknown, high entropy, non-stationary and noisy is a hard problem. Since there is no knowledge of the environment, there can be no predefined statistical graphical model or structure for knowing what kinds of percept that will arrive next. There can be arbitrarily many constants and relations of arbitrary arity, which can result in a large state space and high entropy with a low rate of repetitive encounters. The environment can be changing frequently and noisy, with different percept subsequences occurring for different environment. For example, the percept sequence may look different when an agent is engaging in a fight with the same monster at different location. While each atom can be treated as a proposition, ignoring the relational structural properties can miss out opportunities to predict percepts that have not been seen before.

Therefore, prediction is hard, especially in unknown and noisy environments. In the case that the data is relational, the technologies available for this task are mainly based on production systems or statistical graphical model inference processes such as Bayesian networks. To apply these approaches, it is necessary that domain knowledge or a substantial amount of example data be available to a human engineer or computationally-expensive learning process [1]. In addition, percept sequence is characterized by high variability or a large number of unknown predicates and object constants become known only later, or an environment is changing over time.

The above characteristics of unknown and noisy relational time-series present many challenges and opportunities for sense-making. We have not seen any research effort that directly addresses the learning and prediction problem of relational time-series on unknown, high entropy, non-stationary and noisy environment. Research areas such as statistical relational learning or operator observable model are the most relevant. However, they are not designed for relational time-series prediction.

D. PROBLEMS OF PREDICTION

The first problem is that the current prediction techniques demonstrated on relational time-series have limited capability, if not none, to predict novel percepts. The Bayesian and Markov prediction techniques require percepts to be encountered first before they can predict them in the future. These techniques are suitable for domains when simplified percepts repeat frequently and new simplified percepts are few. While analogical reasoning can infer new knowledge on unknown domain, the first attempt of analogical reasoning on relational time-series prediction produces poor prediction accuracy when the simplified percepts encountered are mostly novel [8].

The second problem is the accumulation of new situations learned by the situation learning. In noisy environments with often new percepts, the rate of new situation encounters will be high, which will cause the number of situations to grow. The continual growth of situations will affect the overall complexity of learning and prediction when the time-series is long. In addition, the situations that are stored may have been obsolete but will continue to use memory and computational cycles.

E. OBJECTIVES AND METHODOLOGY

This dissertation aims to improve the current learning and prediction methods on relational time-series by exploiting cognitive science theories. We explored conceptual blending and event segmentation theories. It also aims to show that learning and prediction of relational time-series can be extended to a variety of real-world tasks.

We applied the prediction algorithms on three rather different applications: (1) Pymud, a role-playing gaming environment, (2) alerts reported by a network intrusion-detection system and (3) Simkit, a discrete event simulation engine.

F. DISSERTATION OVERVIEW

The introduction, problems, objectives and methodologies are given in chapter 1. The literature survey is given in Chapter 2. Chapter 3 describes the algorithms of the computational models used for making predictions. The experiments on the three application domains are given in Chapters 4, 5 and 6, respectively, followed by an algorithmic analysis in Chapter 7. In Chapter 8, we introduce another cognitive model (event segmentation theory) and show how it can help to improve the prediction performance of our conceptual blending predictor. In Chapter 9, we describe several ways in which we can eliminate some data to improve search efficiency while maintain the similar level of prediction accuracy. In Chapter 10, we describe a computation implementation of double-scope blending to see how it can help in extreme novel situations. We conduct some sensitivity analysis and describe the results in Chapter 11. Chapter 12 is the conclusion.

II. LITERATURE SURVEY

A. INTRODUCTION

Time-series predictions have been widely used in real world prediction such as weather forecast, economics data forecast, utility demand forecast, etc. Sapankevych and Sankar [9] have done a comprehensive survey of time-series prediction techniques using support vector machines. Most of the prediction techniques are based on machine learning that learn a nonlinear model from the data. The data usually contain real values, which can be modeled through regression analysis. The relational time-series that we are interested in is a time-series of predicated categorical data. Sun and Giles [8] provide a nice introduction and review of approaches for sequence learning and prediction for categorical data. Their review only addresses sequences of propositions and ignores the relational properties afforded by relational time series. Our survey will focus on possible prediction techniques that may work on categorical predicated data.

There are many existing techniques that are capable of making predictions of future percepts, depending on the characteristics. Some examples of these techniques include production system, Bayesian network, Markov model, etc. We will evaluate some of these techniques to qualitatively assess their possible utility on learning and prediction of relational time series. Recall that the characteristic of relational time-series are unknown, stochastic, noisy and high variability of predicate and object constants.

B. POSSIBLE PREDICTION TECHNIQUES

1. Rule-based System

Rule-based systems are knowledge-based systems whose behaviors are governed by a set of precondition-action rules [10]. When the preconditions of a rule is satisfied or matches some states of the world, the action of the rule is triggered. Rule-based systems encapsulate domain knowledge in rules and have no or limited learning capabilities after they are trained and deployed. Referring to Chapter 1, the preconditions can refer to a set of simplified percepts that form a situation. If the current situation (a set of simplified percepts) matches the preconditions of a rule, the action of the rule is to return a simplified percept as prediction. If we know the application domain, we can write rules to make predictions based on preconditions encountered. Rule-based systems rely heavily on domain knowledge, which can only be created for known environments. Even if the developers have good anticipation capabilities or foresightedness, encoding a large state space or writing rules to address all possibilities are usually prohibitive.

2. Finite State Machine

Finite state machine [10] is a “finite, directed, connected graph, having a set of states, a set of input values and a state transition function.” The transition function defines the transition of states from one state to another. One state can only transition to one other state given a particular input. Finite state machines encapsulate domain knowledge in the finite state transition diagram and have no learning capabilities after they are designed and deployed. Referring to Chapter 1, state space corresponds to our simplified percept space while transition corresponds to a prediction. We can let the previous current simplified percept be the state in the finite state machine, and let the current simplified percept be the input. The transition is defined by the previous and current simplified percept. However, since finite state machine requires a predefined finite state transition diagram, which we assume is unavailable for relational time-series prediction.

3. Markov Model

A finite state machine is a special type of Markov model [10] in which the transitions in the machine are deterministic. A Markov model is a stochastic model such that there are possibly multiple next states in which a state can transit, whose probability of transition is conditioned on the current and historical states, depending on the order of the Markov mode. If the next state is defined based on the previous state, it is termed first order Markov model. If the next state is based on n previous percepts, it is termed n^{th}-order Markov Model. In a variable-order Markov model of order n, the next state is defined based on n previous states during learning. During prediction, if nth order is not achievable, the next lower order is used, and so on until the first order.

We can view a relational time-series as a Markov model in which each state corresponds to our simplified percept. Each simplified percept can be seen as a state in the percept space, made up of all possible combination of predicates and object constants. The transition with the highest conditional probabilities on the maximum achievable order is return as the prediction.

One limitation of the Markov model lies in its strict ordering. A new percept chain may simply have the order of two percepts swapped, or have extra trivial but relevant percepts in between two previously encountered percepts, but the Markov model will treat them as a new chain. This limitation may result in over-fitting. Furthermore, Markov model cannot predict a novel percept but can only predict percepts found in the Markov model. Nevertheless, a Markov model is suitable for online unsupervised learning in unknown and stochastic environment. An implementation of the variable-order Markov model is described in the next chapter.

Li et al. [11] proposed a sequential approach that is applied in the correlation of intrusion-detection alerts. During the offline training, the algorithm divides the entire list of processed alerts into multiple shorter sequences by using a sliding window. The sequences are then fused to form a minimal set of sequence that best represent the set of sequences. The sequence diagram generated is shown in Figure 2. . This approach is similar to n-order Markov chain approach. The variable-order Markov model described in the next chapter is a similar to Li et al. [11]'s sequential approach but with variable order and support online learning.

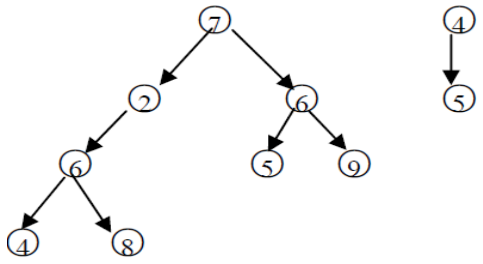


Figure 2. Sequence Diagram. Each Node Represents an Event. Each Edge Represents a Transition of Event. [11]

4. Observable Operator Model

The observable operator model [12] models a stochastic process in order to compute the probability distribution over all possible future sequences, given that a sequence of observation has been observed. The probability of observing a future sequence is

$$P\left(Y_0 = a_{i_0}, Y_1 = a_{i_1}, \dots, Y_k = a_{i_k}\right) = \mathbf{1} T_{a_{i_k}} T_{a_{i_{k-1}}} \dots T_{a_{i_0}} w_0$$

Where

- Y_0, Y_1, …, Y_k are random variables in the sequence
- a_{i_0}, a_{i_1}, … a_{i_k} are the observables corresponds to the random variables and i refers to different types of observable.
- 1 is an identity vector that attempts to sum the column vector to form the probability value
- $T_{a_{i_k}}$ is the operator corresponds to an observable at position k in the sequence where $T_a=M^TO_a$ where M^T is the transpose of the state transition matrix and O_a is a diagonal matrix that express the conditional distribution of each observation given each state.
- w_0 is the initial distribution of the hidden states.

The learning process requires prior manual estimation of the random variables and observables. In our setting, since we have no idea what to expect in the relational time series, there is no way that we can identify the random variables and observables. Jaeger et al. [12] describes a simple way to learn the random variables and observables iteratively. The estimated model in the previous iteration is used to construct an estimator with a better statistical efficiency for the next one. In the application of predicting future characters in a storybook, only 2 to 5 iterations are typically needed. Nevertheless, it is infeasible if we need to run the process each time when a new percept arrives. Spanczer [13] has also identified that learning in observable operator model, though Simple, but is a partially solved problem. He also highlighted the difficulty of choosing the heuristics required to have an efficient algorithm that can converge fast enough to the "real" observable operator model.

5. Bayesian Network

A Bayesian network is a directed acyclic graph where the nodes are random variables and edges represent conditional dependencies between the random variables [14]. Bayesian networks encapsulate domain knowledge in the form of conditional probabilities. The direction of inference is usually predefined because learning the direction of inference is too slow for many online learning tasks due to that conditional probabilities are usually unidirectional while the direction of inference is not obvious from the data. The inference structure learning can be seen as choosing a model from all possible inference networks that represents the data, which has been shown to be NP-Hard [15]. Bayesian networks with predefined inference structure have been used to interpret percept sequences, to derive possible adversarial goals and actions by computing the posterior probabilities of goals, states, and plans given the percept sequences [16]. When no training example is available, the conditional probability tables are based on human subjective judgments. With the large state space and changing environment, Bayesian networks structural learning is infeasible.

With reference to relational time-series prediction, we can interpret each situation-target tuple as a Bayesian network. From a set of situation-target tuples, we can have one naive Bayesian network for each target percept, where the target percept is the parent node while the percepts in the situation are the child nodes, effectively forming multiple simple Bayesian networks. To generate a prediction, we compute the posterior probability of all target percepts given the current situation and return the target percepts with the highest condition probability as the prediction. Bayesian mixture is another Bayesian network that improves upon naïve Bayesian to allow it to learn certain functions such as Exclusive-OR. Bayesian mixture contains probability mixture densities, constructed by normalizing a linear combination of two or more simple Bayesian networks probability densities having the same parent and child nodes. We will describe the Bayesian network techniques in the next chapter.

6. Genetic Algorithms

A genetic algorithm is a kind of evolutionary algorithm that can be described as a variant of stochastic beam search in which, successors states are generated by combining two parent states [17]. A genetic algorithm begins with a set of k randomly generated states. Each state is represented as a string of finite alphabets that represent some parameters in the real world. A fitness function, which can be a heuristic function or a simulator, is used to evaluate the value of the states. The child states are generated from the parent states through a process of selection, pairs, cross-over and mutation. Genetic algorithms have been used to predict adversary's future action by selecting sequences of events/states/actions based on perceived goals and situations [16]. For example, given a current situation and assumed goals of the adversary, genetic algorithms can generate a sequence of events to maximize some functions or the likelihood to achieve the assumed goals.

With reference to relational time-series prediction problem, the sequence of events can serve as percept predictions. The characters of the state could come from the percept space. There are many limitations of the genetic algorithms for use in the relational time-series prediction problem. Given an unknown domain, the percepts are unknown. Even if we could generate a permutation of all possible percept, we would not be able to know in advance the length of the string. In addition, evaluation functions can limit the nature of scenarios to be evaluated. Furthermore, these functions are usually developed for known domains. The other limitation is the assumed adversarial goals, which is also unavailable when the domain is unknown. Hence, genetic algorithms can only be used if domain knowledge is available.

7. Inductive Logic Programming

In inductive learning, also known as concept learning, an agent learns a general function or a set of rules from specific input-output pairs [17], [18]. The input usually contains a set of attributes and values. The rules or the concepts are learned if a combination of values in a certain feature set is a member of the learned concept. In relational time-series learning and prediction, we only have percepts as the value of the features set. Since each percept can contain anything from an unknown domain, we have no way to identify a set of features and training examples for concept learning.

Inductive Logic Programming is a type of inductive learning that induces first order logic theories from examples in relational form. For example, if we have the following percepts: FATHER(JOHN, CALEB), FATHER(CALEB, TIMOTHY), GRANDFATHER(JOHN, SHERYL), we can induce a rule: $\forall x \forall y \forall z$, FATHER(x,y), FATHER(y,z) ⇔ GRANDFATHER(x, z). The main limitation is on the strict logic constraints. A rule will not be learned if there is even just one counterexample. For example, the grandfather rule is generally true. However, if there is just one case of abnormal relation in the family that contradicts the rule, that rule will be violated, and will not be induced, even though it may be true statistically. Such contradictory phenomena are common in noisy environments.

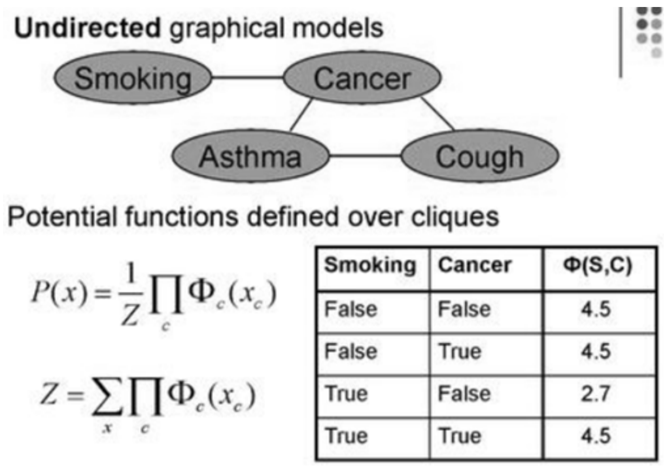
While probabilistic inductive logic programming may seem to better address the stochastic domain, the entire inductive logic programming algorithm must be rerun for each arriving percepts. This poses a great problem because inductive logic programming is exponential in the number of predicates and constants. Hence, inductive logic programming is unsuitable for online learning in relational time series.

8. Reinforcement Learning

In reinforcement learning [10], an agent learns a set of policy for action selection. The policy contains a set of state-action pairs with a value that describes the historical goodness of applying that action in that state. The goodness value is accumulated based on a reward or penalty function known as "reinforcement". Reinforcement learning is not the same as relational time-series learning mainly because its main focus is to learn a set of policies to maximize the cumulative reward, while relational time-series learning and prediction problem needs to predict environmental states even though they are irrelevant to the reinforcement calculation. Furthermore, in unknown domain, the reinforcement may not arrive or may be unknown.

9. Statistical Relational Learning

Statistical relational learning combines first-order logic with statistical learning [19]. The relational learning addresses the relational representation (first order logic) that better represents the world, while the statistical learning addresses the uncertainty of the data by relaxing the hard constraint in the relational domain. Statistical relational learning is usually modeled using a graphical model such as Markov network (MN) or Bayesian Network (BN). While Bayesian Network models causality, Markov network models association between two random variables, in the form of an undirected graph. The nodes in the Markov network are organized into cliques. A potential function Φ() is defined for each clique, which assigns non-negative real values to each state in each clique. The equation and an example for calculating a joint distribution is given in Figure 3. .


Undirected graphical models
Smoking
Cancer
Asthma
Cough
Potential functions defined over cliques

$$P(x) = \frac{1}{Z}\prod_{c}\Phi_c(x_c)$$

$$Z = \sum_{x}\prod_{c}\Phi_c(x_c)$$

Smoking	Cancer	Φ(S,C)
False	False	4.5
False	True	4.5
True	False	2.7
True	True	4.5

Figure 3. An Example of Markov Network from [20].

The example shows four random variables. In statistical relational learning, each random variable is a percept. In reference to our prediction problem, we are trying to predict which of the random variables is true given the state of the other random variables. Smoking and cancer nodes form one clique while cancer, asthma and cough nodes form another clique. Suppose that we have Φ(Cancer=true, Asthma=true, Cough=true)=5.0, Φ(Smoking=true, Cancer=true, Asthma=true, Cough=true) = (4.5 * 5.0) / Z where Z is a normalizing factor that sum over all possible states. Statistical relational learning has seen many applications such as relational classification [21], link based clustering of web search [22], and link prediction in relational data [23].

Khosravi and Bina [24] identified several limitations of statistical relational learning. The biggest limitation is the complexity of inference because the size of the graph grows exponentially with the number of attributes and objects. Most inference methods are based on the standard Bayesian or Markov network inference approaches. Markov network's inference approach requires the computation of the partition function Z, which makes the inference process NP-Complete. Most of the current research is focused on making the inference process more efficient. Statistical relational learning appears to better suit a domain with low variability that has many instances of repeated data, which are usually arranged in a relational database. This is due to the great challenge of structural learning in statistical relational learning. In our relational time-series problem where we expect mostly unknown, large and changing state spaces, Statistical relational learning is unsuitable for relational time-series learning and prediction.

10. Recognition Primed Decision Making

Recognition Prime Decision Making (Ross et al., 2004) is a human decision model that says that human make decision based on their experience. Sokolowski (2007) has developed the Recognition Prime Decision Making agent based on the recognition-primed decision making to model a military decision-maker at the operational level of warfare. Sokolowski uses expert-system approach, which comprises of handcrafted frame data structure that corresponds to a single experience that holds the cues, goals, and actions that describe that experience. In each decision situation, the RPDAgent searches its table of frames to look for a match. If a match is found, the matching frame, together with its associated cues, goals, and actions will be retrieved. Otherwise, the model will ignore the situation. The frame can be seen as the preconditions of a rule-based production system. With reference to our work, if the domain is known, we can create frame to match against the situations. Nevertheless, our set of percepts that made up a situation may not contain cues, goals, and actions. Kunde and Darken [3] show that prediction ability enhanced the realism of the behavior of a military commander by predicting the near future events before making a decision on the calling of fire on incoming enemy tanks. They use a decision tree approach to model the process of mental simulation in order to predict a future event. While these models demonstrated higher fidelity for modeling a human agent, these models require domain knowledge and cannot be used for novel situation prediction.

11. Case-based Reasoning

Case-based reasoning [25] is a problem-solving method that uses a knowledge based system of past known problem-solution cases to provide a solution for new problems. Case-based reasoning attempts to solve new problems by adapting solutions of similar past problems [26]. Case-based reasoning has been widely applied and includes questioning and answering [27], product classification [28], medical diagnosis [29], etc. Case-based reasoning can predict future events by retrieving a similar case previously encountered. In our context, we can match the current set of percept against the problems in the case repository. Like recognition prime decision-making agent (RPDA), a problem description may contain a set of attributes such as goal, action to describe a case. In our case, we only have the percepts, which come from an unknown percept space. Unlike the recognition prime decision making, case-based reasoning can represent messy concepts using examples [30].

Case-based reasoning usually assumes some kind of canonical labeling for similarity matching purpose and k-Nearest Neighbor is a common metric used for similarity measure. These approaches require the attributes-values and their weight to be predetermined in order to determine the contribution to the similarity measure when one attribute is similar or different. In an unknown domain when the attributes are largely unknown, traditional approach to case-based reasoning is not suitable for relational time-series learning and prediction. A statistical lookup table can be said to be a kind of case-based reasoning when each case to case matching is either exactly the same or not.

12. Analogical Reasoning

Analogy is the mind's ability to perceive associations between dissimilar things, and to make analogies based on these associations. Analogical reasoning attempts to find associations between the current problem and the known problem, by looking for unification among the attributes in the two problems. Analogical reasoning has been widely applied in areas where new knowledge is created from existing one. French [31] provides a comprehensive on analogical reasoning.

Reichman [32], from a spontaneous dialogues experiment, proposed that analogy focus on relationships between objects for dialogues. Analogy, through a context space model of discourse, specifies a frame of reference for discussion, and uses structure mapping approach for reasoning. Winston's work on analogical reasoning [33] is probably one of the earliest one. He demonstrated the analogical reasoning ability to infer knowledge about a new domain (electrical current) from a known domain (water current) and in comparison of story plots. He advocated identifying important predicates such as a predicate that has some form of "cause-effect" relationship. Although the problem being addressed is not on relational time-series prediction, Winston pointed out the problem of exponential time complexity and suggest using constant properties to filter away unwanted unification. Marshall's Metacat [33], an analogy making computer model allows comparison of problems in an insightful way and able to recall patterns that occur in its own "train of thought".

Analogical reasoning has seen several good applications. Mirayala and Harandi [35] used analogical reasoning to derive software specification by constructing an analogy of an informal specification to a formal one found in their knowledgebase. Objects are mapped by types without consideration relationship between objects. MacKellar and Maryanski [36] used analogical reasoning to retrieve knowledge from a database through the use of an example. Breitman et. al. [37] used analogy mappings to map entities between handcrafted database schema and a new weak database schema, to generate new entity-relationship in order to improve the weak schema of the new database. Eremeev and Varshavsky [38] show how analogical reasoning can be used on intelligent decision-support systems to look for solution of problems for diagnostic and forecasting. McSherry [39] used analogical reasoning to estimate the value of home property, by mapping the attributes of home property to those of known value properties. Baydin et. al. [40] recently demonstrated one possible use of analogical reasoning to retrieve distant but relevant cases to solve mediation problem. While their works demonstrate novel solution generation, a human is required for guided matching and case retrieval before the solution can be meaningful. Baydin et. al. use the structural mapping approach by Falkenhainer et. al. [41]. Falkenhainer's structural mapping demonstrates interesting analogical reasoning capability that uses guided binding of elements from one situation to another situation.

While analogical reasoning can generate novel knowledge, the case searching and matching process has exponential complexity because the unification processes in structural mapping is equivalent to a subgraph isomorphism problem. To speed up case search, MacKellar and Maryanski [36] used case indexing, which associate cases with type metadata. Cases that are of the same type as the problem on hand are identified for analogical reasoning. The searches for the optimal set of unification are usually based on backtracking search [41]. To speed up the process, heuristics are usually use for guided binding of elements and the backtracking method assume a connected graph. So far, we have not seen analogical reasoning applied on the problem of learning and prediction of relational time-series prediction. The reason could be due to complexity issues. Nevertheless, since analogical reasoning can generate novel knowledge, we look to a similar reasoning technique called conceptual blending.

C. CONCEPTUAL BLENDING

1. Theory of Cognition

Conceptual blending is a proposed general theory of cognition developed by Fauconnier and Turner [42]. The theory describes the way humans process and rationalize information through a set of mental operations. In their book, Fauconnier and Turner [42] present various examples that show how the theory of conceptual blending is one possible explanation of how humans think. The theory also explains the process by which humans assign meaning to incoming information from sensory input, then they integrate it, and then finally learn and gain new knowledge. Fauconnier and Turner [42] suggest that humans unconsciously and constantly blend when they talk, listen, and think in every aspect of human life. The blending process happens at a fast speed and generates many blends in parallel. The blended space can, in turn, serves as input space for subsequent blends.

To explain briefly, conceptual blending is a set of human cognition theories that explain how humans make sense of the world, through a process of imaginative blending of concepts to arrive at an understanding of a new environment. Conceptual blending blends two input spaces to form a blended space. Input spaces may refer to our previously encountered situation or the current situation. The blended space is a new situation, which can be hypothetical. The generic space in conceptual blending contains the common structure found in both input spaces. Extra counterfactual elements are then added from the blended space to the input spaces through the back-projection mechanism. There are four types of network in conceptual blending, which are differentiated based on how structures from the two input spaces are used in the blend. Structure is defined by the relations and object constant types in the situation.

Two of the network types, single-scope and double-scope networks, are similar to analogical reasoning. Other than single-scope network, we will also evaluate the other three types of network on relational time-series prediction.

a. Mental Spaces

Conceptual blending is a set of operations in which existing mental spaces are integrated to form new mental spaces. According to Fauconnier and Turner [42], mental spaces are small conceptual packets, constructed as we think and talk for local understanding and action. Mental spaces contain elements that are structured by frames and cognitive models. They are modified as thought and discourse unfold. They operate in working memory and are connected to long term schematic knowledge and specific memory. Mental Spaces can be given an abstracted neural interpretation by thinking of activated mental spaces as co-activated neuronal assemblies where the links between elements correspond to co-activation (neurobiological bindings).

There are three types of mental spaces: input, generic and blended spaces. While input spaces are activated spaces, generic spaces contain the common elements and links from the input spaces, and the blended space contains elementscaptured in the generic spaces and other more specific structures.

In one example given by Fauconnier and Turner [42], one input space represent an office environment, and one input space represent the computer science environment. The generic space show the common aspect of the office and computer science environment such as files and folder. The blended space is a computer desktop design that has folders and files, but has a recycle bin on the desktop.

b. Organizing Frame

Every mental space has two parts: an organizing frame and a set of elements. The organizing frame provides relations to organize the elements. An example is that "father-of" is a relation that relate two elements "John" and "Mary".

c. Integration Network

A simple integration network is described in Figure 4. . In the figure, two input spaces connected by solid lines representing the cross-mappings among related elements in the input spaces through "vital relationship" mapping. The vital relations identified are change, cause-effect, time, space, identity, change, distinctness, part-whole, representation, role, analogy, disanalogy, property, similarity, category, and Intentionality. The dotted lines are the projection of elements from the input spaces to the generic and blended spaces. The links between the input spaces are known as "outer-space" links and are compressed into an "inner-space" links inside the blend. The box in the blend is the organizing structure that organizes the elements in the blend.

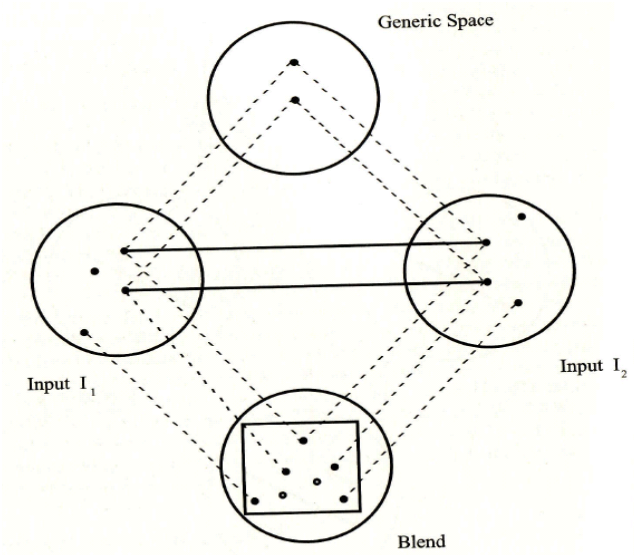


Figure 4. Simple Integration Network from [42].

d. Constitution principles

Fauconnier and Turner [42] provide a set of principles to govern the blending process. Blends are generated through a set of operators: composition, completion, and elaboration. The composition operator selectively projects elements from the input and generic spaces. The completion operator adds to the blended space additional elements and relations based on independently recruited frames. The elaboration operator models the process of a human being anticipating results or consequences by thinking or imagining into the future. After the blending process is completed, the projected or simulated counter-factual conclusions that resulted from the elaboration process can be back-projected into the input spaces to add meaning and understanding to them. Back-projection is a term that describes the adding of a hypothetical percept into the current situation.

e. Simple Network

There are four types of blending networks, depending on which organizing frame is used. An organizing frame is the frame or structure that is used to organize the elements in the blended spaces. The first is a simple blending network in which one input space contains the organizing frame while the other input space does not have a frame. The cross-mappings between input spaces are usually roles to values connections. The relevant parts of the frame in one input are projected with its roles, and the elements are projected from the other input as values to those roles in the blend. For example, the organizing frame may consist of a father-daughter relation while the other input space provides two individuals for the values to the roles in the frame.

f. Mirror Scope Network

The second one is a mirror network in which all spaces (inputs, generic, and blend) share the same organizing frame. The purpose for mirror network is mainly to compare the differences between two input spaces. Fauconnier and Turner [42] use the monk example in which the uphill going monk and the downhill going monk were walking on the same path. The organizing frame is the frame of “walking along a path”. The illustrated blended space may not have the same organizing frame because it has two monks instead of one. In our case, we will use the definition that all spaces have the same organizing frame

g. Single Scope Network

The third one is a single scope network in which there are at least two input spaces, each with a different organizing frame. However, only one of the organizing frames is used to organize the blend. Fauconnier and Turner [42] use the illustration of a company's chief executive officer (CEO) "fighting" with another company's CEO. The illustration blends the business context with the boxing sport context. The frame that is being used is from the boxing input space to illustrate the win-lose characteristic from the sport context.

h. Double Scope Network

The fourth one is a double scope blend in which both inputs have different organizing frames, and the blended space's organizing frame is made up of parts of each of those frames and has an emergent structure of its own; that is, a new type of organizing frame is created through a double-scope blend. Fauconnier and Turner [42] cite the illustration of a computer desktop, which is a combination of a computer frame and an office desktop frame. The computer desktop is not an office desktop but the frame of an office desktop provides the context of a familiar working environment. Meanwhile, the computer frame situates the desktop within the computing environment, thus creating the emergent frame of a computer desktop.

i. Governing Principles

The number of blends that can be generated is potentially huge by virtue of the combinatorial permutations of input space elements. However, many of the blends generated are not meaningful. Fauconnier and Turner [42] have identified a set of eight governing principles to provide a way for evaluating and selecting blends. The governing principles include compression, topology, Pattern Completion, Integration, Promoting Vital Relation, Web, Unpacking and relevant. These governing principles are collectively known as optimality principles. Some principles can conflict with other principles so not all principles need necessarily be satisfied fully but sufficiently and optimally.

j. Applications

The original text by Fauconnier and Turner [42] contains a lot more details and the reader is referred to it for a more thorough treatment of the theory, including many examples illustrating the theory and principles. Conceptual blending has since been applied to understanding formal expressions in linguistics [43], explaining metaphorical reasoning [44], understanding counterfactual reasoning [45], analyzing mathematical evolution [46], and developing human computer interfaces [47]. Other applications of conceptual blending can be found on the Blending and Conceptual Integration web portal (http://markturner.org). Computational creativity, in particular, involves a specific from of conceptual blending known as double scope blending, and has been applied to machine poetry generation [48] and the generation of animation characters [49]. Tan and Kowk [50] applied the double scope blending to generate novel and creative scenarios for sense-making in a maritime security domain.

Ozkan [51] implements a threat assessment model, using a multi-agent system and conceptual blending theory, to mimic how a human expert assesses the intention of an incoming air threat. In another thesis, Tan [52] also implements threat assessment using conceptual blending for surface warfare based on cues such as platform type, position, flag, destination, heading, speed, communication, activity, origin, and ESM to establish various forms of violations to determine the track’s intention through a weighting strategy in terms of “friendly,” “neutral,” “potentially hostile,” or “unknown.” In yet another thesis, Tan [53] uses the conceptual blending theory to develop a threat assessment, resource assignment, and plan generation model. While these theses show that multi-agent system and conceptual blending theory can be used to introduce cognitive intelligence into a computational model, these theses only use simple and mirror scope blending, which requires domain knowledge and cannot be used to predict novel situation.

2. Relating Conceptual Blending to Relational Time-series Prediction

Conceptual blending, though vividly espoused by Fauconnier and Turner [42], poses significant challenges to computational modelling. Key among these is that the mechanisms underlying some of the human cognitive processes described by Fauconnier and Turner are still not clear to them. For example, they mentioned that humans are able to activate appropriate frames for conceptual blending but did not explain how such frame activation is achieved. Henceforth, certain simplification will have to be made in order to facilitate the design and implementation of a workable computational model of conceptual blending to improve prediction accuracy on relational time series.

With reference to our problem on relational time-series prediction, the mental spaces are our situations, which comprise of a set of percepts. Input spaces can either be previously encountered situation or the current situation. The blended space is a new situation. The generic space contains common percepts in both input spaces. Back-projection adds the target percept to the input spaces as prediction. The summary of correspondence between conceptual blending and relational time-series prediction is given in Table 1.

Conceptual Blending	**Relational** Time-series **Prediction**
Mental space	Situation: a set of percepts
Dots in mental space	Object constants or variables
Input space 1	One previous situation
Input space 2	Current situation
Generic space	A set of common percepts in input space and 2 after unification
Blend	Previous situation with unification from current situation
Cross space mapping	Unification
Identity mapping	Unification by identity vital link
Back projection	Adding unified target percept to the input space
Organizing frame	The relations and constant types used in the blend
Simple Network	Not applicable because frame cannot be created for unknown domain
Mirror Network	Statistical Lookup table
Single Scope Network	To be described in more details
Double Scope Network	To be described in more details

Table 1 Correspondence between Conceptual Blending and Relational Time-series Prediction.

We will describe how conceptual blending can be implemented computationally in the following sub sections.

a. Simplex Network

Simplex network is a type of blending network in which, one input space contains the structure while the other input space contains only the constants. These structures can be added manually into the knowledge base to represent some known situations with their associated target percept. This is useful for making predictions when the system starts with zero knowledge or that there is no appropriate situation found in our knowledge base that matches the current situation. Such no-match circumstances are commonly found in statistical lookup table and variable matching. An example of such structure can be that (troll and agent are co-located) ├ (troll hit agent) where "├" is the prediction operator, (troll hit agent) is the target percept and (troll and agent are co-located) is the organizing frame. Note that "troll" and "agent" are object type, not objet constant. "co-located" and "hit" are relation.

An example is described in Figure 5. . In the diagram, we have the relational time-series on the left and the integration network on the right. Input space 1 contains the structure in which the percepts are just relations and variables indicated by ?x. The green arrow input space 1 is the target percept. Input space 2 contains the constants. The constants and the variables are matched base on their types. The blend is the input space 1, with constants from input 2. The blue solid arrow symbolizes the back-projection of the target percept to the input space 2. The simplex network has little use when the application domain is unknown because there is no way to hand-create percept sets.

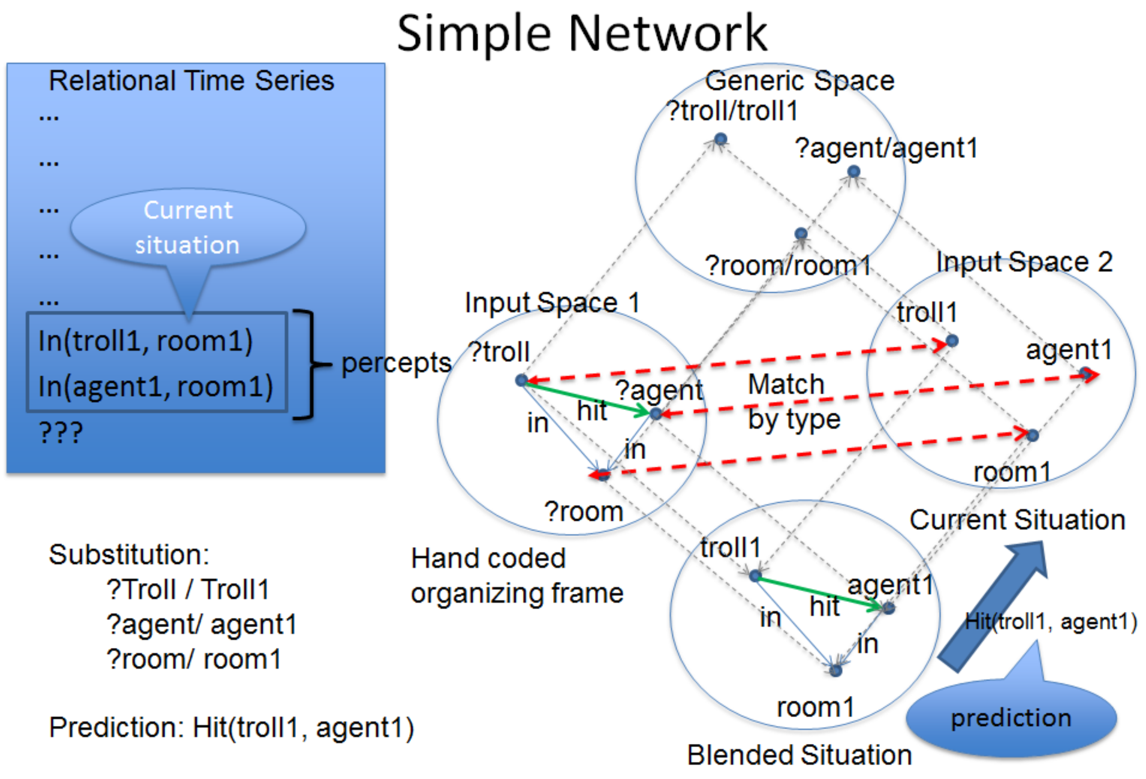


Figure 5. Simplex Blending Network for Structure to Constant Matching.

b. Mirror Network

The mirror network [42] is a blending network that has all spaces (input, generic and blend) having the same organizing frame. The situation matching approaches can be considered as an example of the mirror network when we compare the current situation with the previous situation to look for one that matches exactly as in statistically lookup table, or matches by unification as in variable matching. The blending network here has one input space representing a previous situation, and another input space representing the current situation. The input space 1 is added to the blended space. The purpose of our blend is not to discuss the differences between mapped objects in both input spaces, but to allow the target percepts to be identified and added into the current situation input space as prediction. In the constant mode, the target percept from the previous situation is back-projected exactly to be the next percept of the current situation.

An example is given in Figure 6. . Both input space 1 and 2 have exactly the same situation except that input space 1 has a target percept (green arrow). Recall that each situation comprises of a set of percepts. Each percept is made of a relation with one or more object constants. The first constant in the percept is always the relation constants. The relation can refer to an action or state change (indicated by '+'), which are inconsequence on the blending operation. In this blend, the purpose is to look for another situation that is exactly the same as the current situation. When it is found, the target percept is simply added to the input space 2 as a possible prediction.

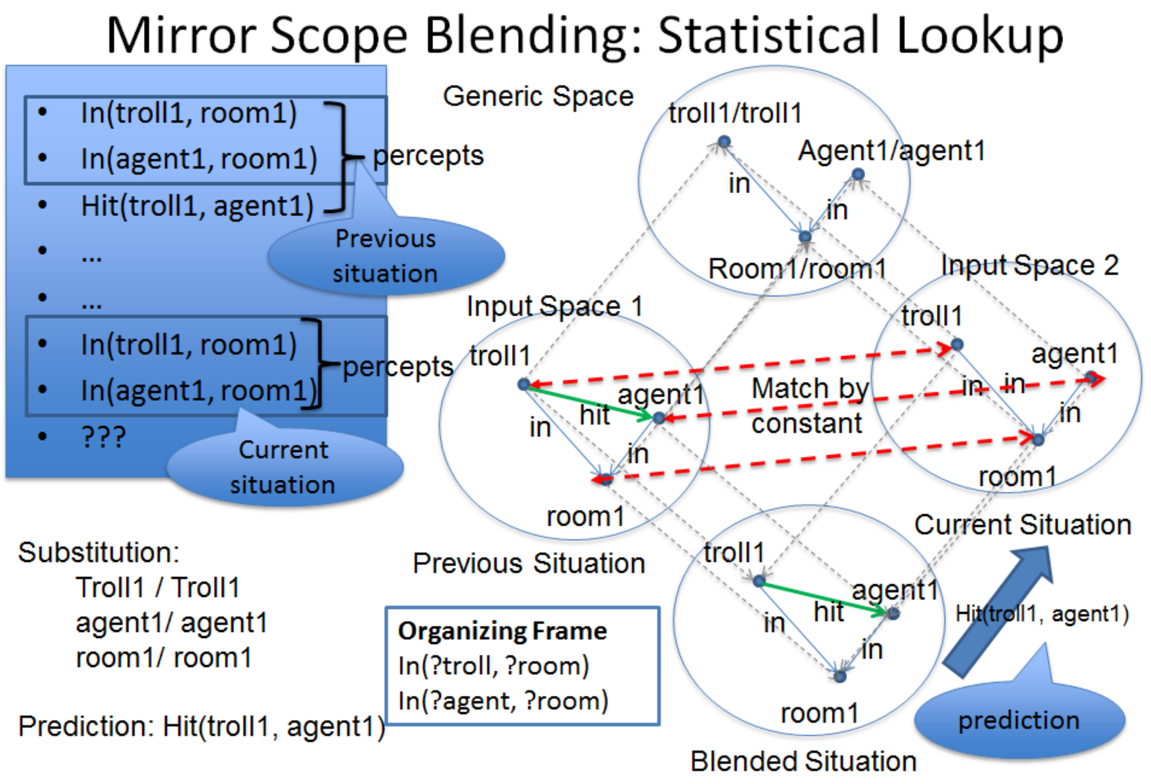


Figure 6. Statistical Lookup Table as Mirror Scope Network

In variable matching, both input spaces may not be exactly the same but share the same set of relations. In Figure 7. , the current situation contains agent2 while the previous situation contains agent1. Both constants are different but are of type agent. In this case, the current situation is associated to the previous situation through a unification process. Finding the unification is equivalent to a graph isomorphism problem. Troll1 is unified with troll2 while agent is unified with agent2. Hence, the bindings can be used to replace the predicted percept structure with constants from the current situation.

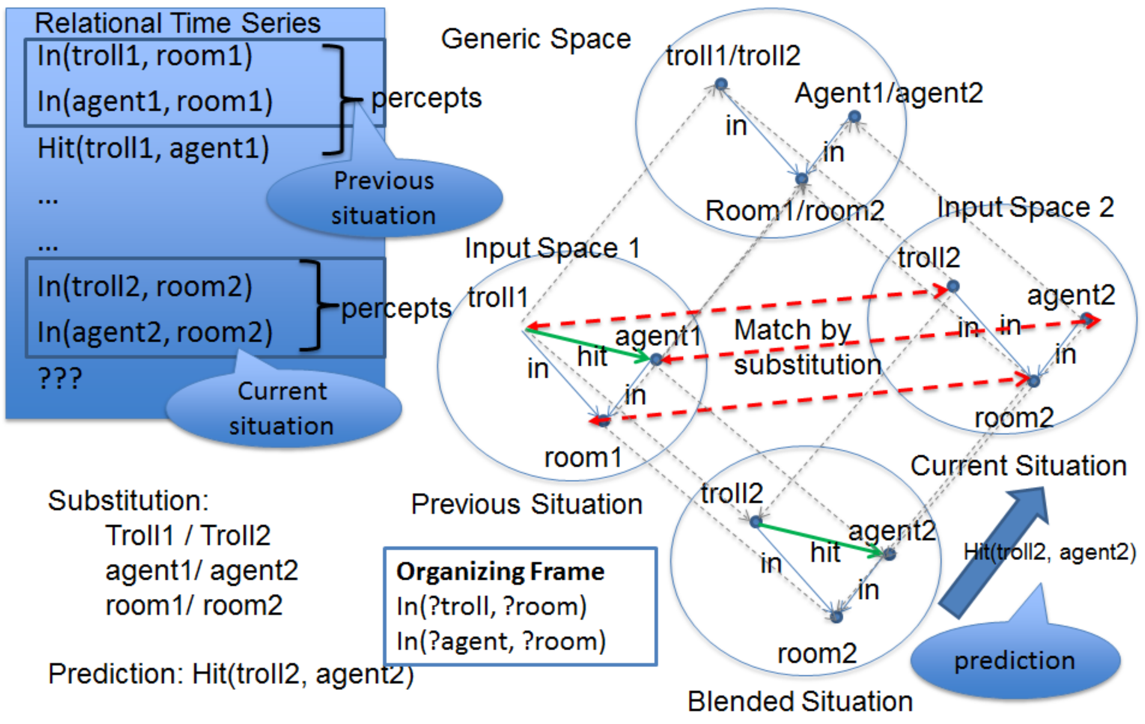


Figure 7. Variable Matching as Mirror Scope Network.

c. *Single-scope Blending Network*

Single-scope blending [42]is a type of conceptual blending which both input spaces have different organizing frames, and one of them is used in the blended space. In our context, input space 1 is one of the previous situations. Input space 2 is the current situation. Both situations are not only different in the object constants, but also in the relations and object types. The blended space uses the organizing frame of input space 1 and some constants of input space 2 if there is a unification of each of those constants to constant in input space 1. The constants from input space 2, if not unified with any other constant, will not be substituted. The generic space contains the common percepts found in both input spaces with the constant binding. A previous situation is chosen such that the generic space is maximized. The substituted target percept with the highest count of occurrence is back-projected to input space 2 as the next likely percept.

The main difference between single scope and mirror scope lies in the selection of input space 1, which is the previous situation. Mirror scope requires exact matching or matching by substitution while single scope relax the requirements further by choosing the previous situation with the most similar structure as the current situation.

An example of single scope blending is described in Figure 8. . We have a current situation [In(agent2, room2), In(dagger1, room2), In(troll2, room2)], which is assigned as input space 2. Given input space 2, we search through all previous situations to find the most similar one as the input space 1. Suppose that previous situation [In(troll1, room1), In(agent1, room1)] is found to be the most similar one and I assign input space 1. This previous situation has a target percept [Hit(troll1, agent1)]. The organizing structure of input space 1 is [In(?troll, ?room), In(?agent, ?room)], which describes that there is a troll in a room and an agent in the same room. Input spaces 1 and 2 have many differences. Firstly, the rooms are not different. Secondly, the trolls are different. Thirdly, the agents are different. Lastly, the current situation has a dagger in the room. The set of unifications found are [Troll1:Troll2, agent1:agent2, room1: room2]. To generate a prediction, we take the target percept and substitute the elements with the unification. Hit(troll1, agent1) becomes Hit(troll2, agent2)

Cross space mappings associate constants from both concepts. The mappings process or unification can be expressed as finding a graph injection from input space 1 to input space 2. Each input space is viewed as a graph. The nodes of the graph are the object constants in the percepts, the relations in the percepts form the links, and we try to unify as many constants as possible. If all constants are unified, we have a bijection between the graphs. Otherwise, it is called injective homomorphism. We assume that percepts with arity more than 2 (two object constants in a percept) have been converted to an equivalent set of percepts with arity 2. Since two situations may be different, we attempt to find the largest subgraph in one concept that can be isomorphically matched to a subgraph in another concept. This is equivalent to the problem of finding a subgraph isomorphism between two graphs, whose associated decision problem is NP-complete. We use a recursive backtracking method to identify the largest common subgraph in both previous and current situation. The backtracking method is complete because it searches through all possible unifications to find a set with maximum possible mappings. When every constant in one percept in concept 1 can be unified with by one constant in another percept in concept 2, we have one common percept if both percepts have the same relation. The total number of common percept is the similarity score. Once the largest subgraph is identified, the constants in the subgraph of one concept can be substituted by the corresponding constants in the other subgraph since they are mapped in the subgraph isomorphism process.

The generic space contains the common percepts in both concepts after the substitution is applied. We look for a situation that maximizes the generic space. It is the most similar situation. In the case of multiple situations that share the same similarity score, the latest one is used.

d. Common Percepts: COMMON(s1, s2)

The common percepts of two situations s_1, s_2 are the percepts that appear in both situations and form an injective function between s_1, s_2. If $|s_1| = |s_2| = |\text{COMMON}(s_1, s_2)|$, there is a bijective function between s_1, s_2.

e. Similarity Score: SIMSCORE(s1, s2)

The similarity score: $\text{SIMSCORE}(s_1, s_2) = \frac{2|\text{COMMON}(s_1,s_2)|}{|s_1|+|s_2|}$.

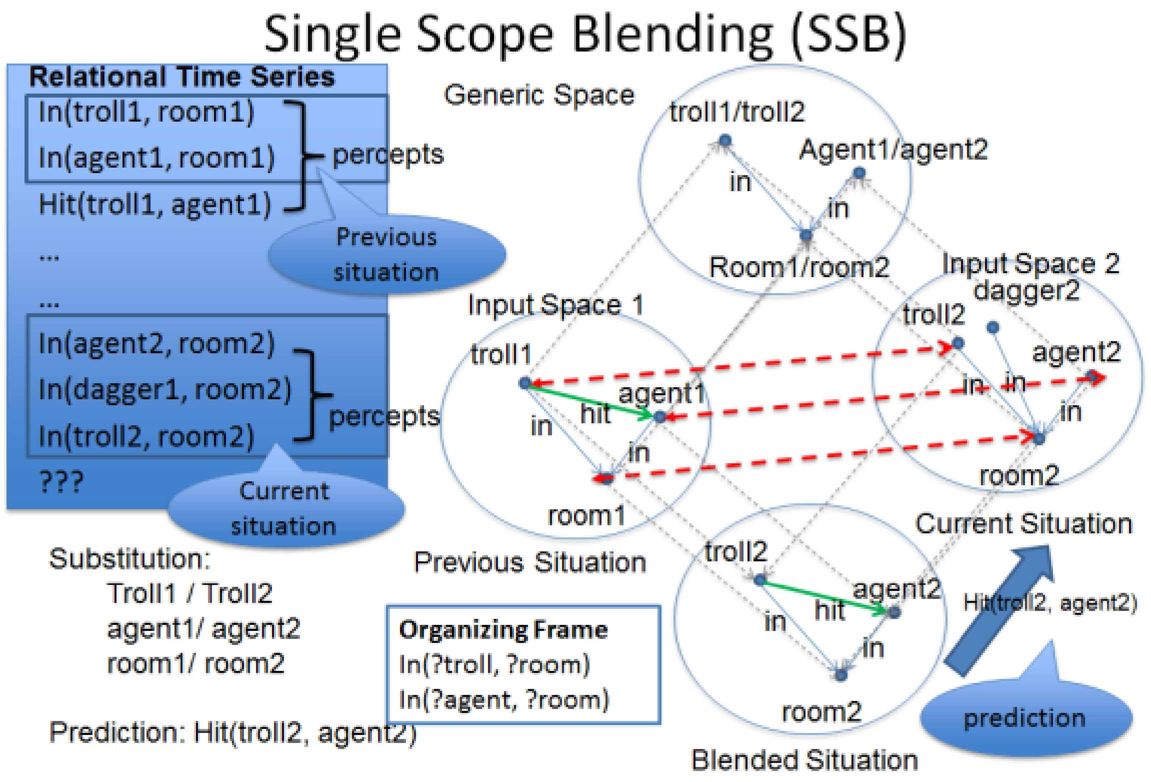


Figure 8. Single-scope Blending Network.

f. Double Scope Network

In double scope network [42], all input spaces have different frames and the organizing frame of the blend comprises of different elements from the structures of both input spaces. The central idea is to create a new structure, one that we have not seen before such that the structure is useful to reason about the current situation. A cartoon example is given in Figure 9. . In the figure, we have a current situation that describes a new situation (a Pegasus) that we have not seen before. In order to predict its capability, we have to find something similar. However, if we only found a horse and a bird in our previous situation, we can create a new structure that combines the horse and the bird structures. Note that we can have more than two input spaces. The new structure may allow a better understanding of the new situation. However, the resultant meaning of the new structure depends on the parts of the old structures added to the new structure. It is possible and common that many structure generated are nonsensical. We will describe our exploration of the double scope blending in a later chapter.

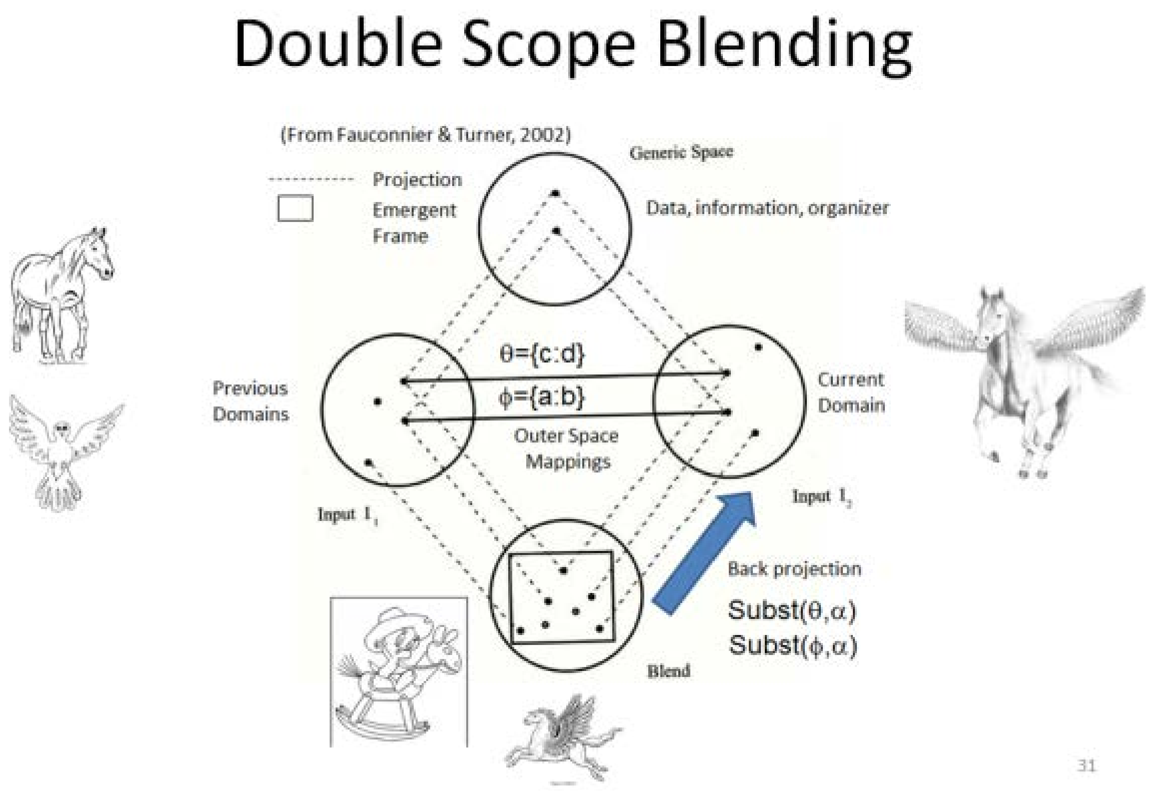


Figure 9. A Cartoon Example for Double-scope Blending, modified from [42].

g. Integrated Networks

Fauconnier and Turner [42] describe that these four types of integration networks run as a single mental process and not in isolation. A simple way to integrate these four networks can be described in Figure 10. . When the system starts and we have no previous situation, we can fall back on some default frame. In our application, since we assume that we do not know the domain, we cannot hand write any general knowledge for this purpose. Nevertheless, there are some general knowledgebase that might be useful for certain general reasoning purpose such as MIT's ConceptNet and Princeton University's WordNet. If the exact situation can be found, we will use the mirror scope network. Otherwise, we can use single scope or double scope, depending on the rate of new situation encounter. In our experiment, we saw that prediction accuracy is slightly better for double scope blending when rate of new situation encounter is high.

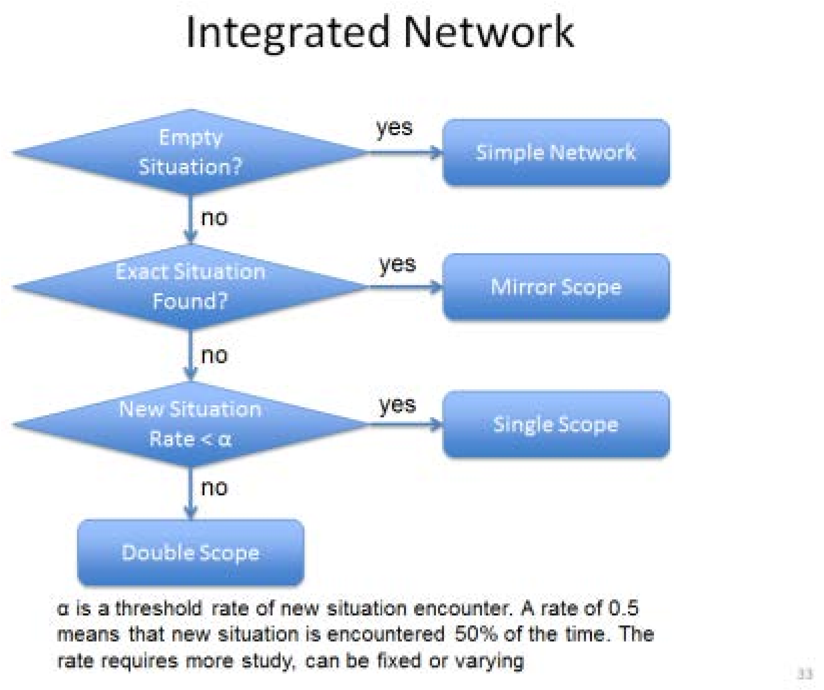

Integrated Network
Empty Situation?
yes
Simple Network
no
Exact Situation Found?
yes
Mirror Scope
no
New Situation Rate < α
yes
Single Scope
no
Double Scope
α is a threshold rate of new situation encounter. A rate of 0.5 means that new situation is encountered 50% of the time. The rate requires more study, can be fixed or varying
33

Figure 10. Integrated Network.

D. SITUATION LEARNING

Situation learning [8] is an unsupervised sequence learning technique that takes a sequence of situation-target tuples from a relational time-series and forms a more concise set of situation-target tuples, stored in a container, by combining situation-target tuples that have the same situation into one situation-target tuples. Given a situation-target tuple, if the situation does not exactly match with any situation in the container, the situation-target is added into the container. If the situation exactly matches with one situation in the container but the target does not match with any targets of the situation in the container, the target percept is added into the situation-target tuple and updates all data count. If the situation exactly matches with one situation in the container and the target exactly matches with one target of the situation in the container, we just need to update data count.

We will illustrate situation learning using the example timed percepts given in Figure 1. . The set of situation-targets learned is given in Figure 12. .

P_i	Percepts	Descriptions
P_1	loc(Ed, road, 1, +)	Ed is at location road
P_2	loc(Fox1, road, 2, +)	Fox1 is at location road
P_3	go (Fox1, east, 3, e)	Fox1 go east
P_4	loc(Fox1, road, 5, -)	Fox1 is NOT at location road
P_5	loc(Fox2, road, 10, +)	Fox2 is at location road
P_6	go (Fox2, east, 11, e)	Fox2 is going east
P_7	loc(Fox2, road, 13, -)	Fox2 is NOT at location road

Figure 11. An example of relational time-series repeated from Figure 1. .

Situation		Target	
{}	1	loc(Ed, road, +)	1

{loc(Ed, road, +)}	2	loc(Fox1, road, +)
		loc(Fox2, road, +)
			1
			1

{loc(Ed, road, +) loc(Fox1, road, +)}	1	go(Fox1, east, e)	1

{loc(Ed, road, +)
loc(Fox1, road, +)
go(Fox1, east, e)}
	1	loc(Fox1, road, -)	1

{loc(Ed, road, +) loc(Fox2, road, +)}	1	go(Fox2, east, e)	1

{loc(Ed, road, +)
loc(Fox2, road, +)
go(Fox2, east, e)}
	1	loc(Fox2, road, -)	1

Figure 12. A Collection of Situations (Left Column) and Targets (Right Column).

When the learning process starts, there is no percept. The current situation is an empty set.

When the first percept loc(Ed, road, +) arrives, it becomes the target percept of the current situation, which is empty. The situation-target tuple ({},loc(Ed, road, +)) is added into the container at the first row in the table of Figure 12. . The current situation is updated to be {loc(Ed, road, +)}.

When the second percept loc(Fox1, road, +) arrives, it becomes the target percept of the current situation {loc(Ed, road, +)}. The situation-target tuple ({loc(Ed, road, +)},loc(Ed, road, +)) is added into the container at the second row in the table of Figure 12. . Assuming we use a time window of 1sec, the current situation is updated to be { loc(Ed, road, +), loc(Fox1, road, +)}.

When the third percept go(Fox1, east, e)arrives, it becomes the target percept of the current situation { loc(Ed, road, +), loc(Fox1, road, +)}.The situation-target tuple ({ loc(Ed, road, +), loc(Fox1, road, +)}, go(Fox1, east, e)) is added into the container at the third row in the table of Figure 12. . The current situation is updated to be { loc(Ed, road, +), loc(Fox1, road, +), go(Fox1, east, e)}.

When the fourth percept loc(Fox1, road, -) arrives, it becomes the target percept of the current situation { loc(Ed, road, +), loc(Fox1, road, +), go(Fox1, east, e)}. The situation-target tuple ({loc(Ed, road, +), loc(Fox1, road, +), go(Fox1, east, e)}, loc(Fox1, road, -)) is added into the container at the fourth row in the table of Figure 12. . The current situation is updated to be ({loc(Ed, road, +)} since loc(Fox1, road, -) is the '-' percept that remove its corresponding loc(Fox1, road, +) interval percept and go(Fox1 east e) is not in the 1sec time window from loc(Fox1, road, -).

When the fifth percept loc (Fox2, road, +) rrives, it becomes the target percept of the current situation {loc(Ed, road, +)}. The situation already exists in the container, and is found at the second row of the table in **Error! Reference source not found.**. Therefore, loc (Fox2, road, +) is added as a second target of situation {loc(Ed, road, +)} at the second row of the table in **Error! Reference source not found.**. Note that the count of situation {loc(Ed, road, +)} is incremented to 2. The current situation is updated to {loc(Ed, road, +), loc (Fox2, road, +)}

When the sixth percept go(Fox2, east, e) arrives, it becomes the target percept of the current situation {loc(Ed, road, +), loc (Fox2, road, +)}. The situation-target tuple {loc(Ed, road, +), loc (Fox2, road, +)}, go(Fox2, east, e)) is added into the container at the fifth row in the table of Figure 12. . The current situation is updated to be { loc(Ed, road, +), loc (Fox2, road, +), go(Fox2, east, e) }

When the final percept loc(Fox2, road, -) arrives, it becomes the target percept of the current situation { loc(Ed, road, +), loc (Fox2, road, +), go(Fox2, east, e) }. The situation-target tuple { loc(Ed, road, +), loc (Fox2, road, +), go(Fox2, east, e) }, loc(Fox2, road, -)) is added into the container at the sixth row in the table of Figure 12. . The current situation is updated to be { loc(Ed, road, +)}

Situation learning has a low complexity and is capable of learning from a relational time-series of possibly unknown, high entropy, non-stationary and noisy environment.

Note that this is not a Markovian approach since the sequential property is removed from each situation ad that a percept may have been received long time ago and still remain true even though other later percepts have become false.

Situation learning addresses the problem of relational time-series learning and prediction by turning the learning and prediction problem into a situation matching and simple inference process. Situation learning stores the percepts in predicate form and allows prediction techniques to use the predicates for inference. For each situation not found in the set of situation-target tuples $\{(s_i,t_i)\}$, situation learning creates a new situation-target tuple, and allow prediction techniques to immediately use it for prediction. Each situation can contain any combination of percepts, regardless of how large the state space is. It manages probabilistic data by having multiple target percepts. Noisy data is managed by a simple creation of additional situation-target pair for new situations. The disadvantage is that situation learning requires the homomorphism from timed percepts to simplified percepts to be sufficiently strong that the same situation can occur reasonably frequently. If the homomorphism is too weak, situations will tend to be distinct and the number of situation-target tuples will continue to grow and will affect the computation time when prediction is required.

E. DISCUSSION

Many possible approaches for learning and prediction of relational time-series such as rule-based systems and finite state machines assume that detailed domain knowledge is known. While these approaches have work well in many applications, they will fail on prediction task in unknown environments. Unknown environments require agents to be robust and flexible, as well as to be able to learn and to adapt in new environments. While a Bayesian network learning agent is capable of online learning, the structures of the knowledge representation are usually fixed. Structural or rule learning are usually limited and done offline due to the exponential complexity. We need structural flexibility or multiple structures to account for the complex nature of the relational time series.

Methods such as inductive logic programming or Markov models are either logic-constrained, have strict sequence requirements, or are based on propositional representations. While Markov models and variants thereof have found many successful applications, their strict sequence requirement prevents them from being used for noisy situations. Likewise, strict logic constraints do not allow inductive logic programming to learn rules with multiple or noisy outcomes. Reinforcement learning is not designed for relational time-series learning and prediction. Furthermore, many methods assume propositional data representations even though the relational formalism is a more natural way of representing the world of objects. While statistical relational learning may allow statistical inference, its highly constrained topological network structures prevent it from use in unknown environments. Hence, these methods are hard to generalize to predict percepts that have not been seen before.

Time-series predictions have been widely used in real world prediction such as weather forecast, economics data forecast, utility demand forecast, etc. Sapankevych and Sankar [9] have done a comprehensive survey of time-series prediction techniques using support vector machines. Most of the prediction techniques are based on machine learning that learn a nonlinear model from the data. The data usually contains real values, which can be modeled through regression analysis. The relational time-series that we are interested in is a time-series of predicated categorical data.

In analogical reasoning, there is a concept of source and target domains where the target and source are from different domain but similar in some significant aspect. The inference method takes the source domain and project onto the target domain to deduce missing details. For example, given a source that says that air flows from a high pressure point to a low pressure one, when projected onto the water domain gives us: water flows from a high pressure point to a low pressure one. The inference method of single scope and double scope blending is similar to analogical reasoning.

However, conceptual blending is different than analogical reasoning for some subtle differences. The first difference is the identification of the mappings between the source and target. In analogy, the mapping is based on significant nodes or relations such as cause-effect [54], while conceptual blending defines a set of possible 15 vital links such as “part-whole, is-a, etc. In our application, we restrict to the “identity” vital link because we do not assume the availability of any knowledge that we can leverage. The use of MIT’s Concept Net or Princeton University’s Word Net can infer other forms of relation as demonstrated in [50]. Identity vital link means that two constants are unified only if both constants are structurally similar, or both constants are of the same type. The identity vital link serves as a constraint in identifying association between constants. This constraint may limit the amount of creativity (such as relating a sword to a dagger) but it turns out that most constants in our applications require strong type constraint. For example, if an IP-address constant is unified with a protocol constant, or that a boat is unified with a velocity constant, the constructed prediction will be wrong.

The second difference between traditional analogy and our problem lies on the connectedness of each situation. In other analogy problems, each situation is fully connected such that the constructed graph with constants as nodes and relation as links is a connected graph. However, in our relational time-series prediction problem, most situations are not connected. As a result, the traditional efficient way of solving a sub-graph isomorphism problem using backtracking will not work. To avoid NP-completeness complexity and yet still able to allow disconnected situation, we need new technique for identifying the association between target and source constants.

The third difference is lies in the use of multiple sources for the transfer to the target in double scope blending. In traditional analogy, there is only one source. In double scope blending, there are multiple sources, which can come from multiple previous situations and even come from the target, which is the current situation.

While we can argue that conceptual blending is different than analogy because of the subtle differences, we can also see conceptual blending as a more detailed specification of the general analogy theory. If conceptual blending is similar to analogical reasoning, the primary contribution of this dissertation is the use of analogical reasoning on relational time-series prediction.

F. CONCLUSIONS

Prediction in known environments can be solved using knowledgebase system or Bayesian inference with a predetermined structure. When the environment is unknown but highly repetitive, probabilistic and Markov approaches will work well. When the environment is unknown but stationary with limited relational and object variety, Bayesian and Markov will take some time to learn. When the environment is unknown, stationary and contains huge varieties and constants, Bayesian becomes infeasible when most percept count are similar or just one. Markov may work if the noise level is low. When the environment is unknown, non-stationary and noisy, both Bayesian and Markov techniques will fail.

THIS PAGE INTENTIONALLY LEFT BLANK

III. DETAILS OF PREDICTION TECHNIQUES USED IN EXPERIMENTS

A. INTRODUCTION

Given a set of situation-target tuples and a current situation, the prediction can be derived by some of the inference techniques discussed in chapter 2. Figure 5 shows the basic idea. The black dots in Figure 13. are simplified percepts in the situation and grey dots are target simplified percepts. These techniques provide means to generate zero, one or more predictions given the current situation and a set of situation-targets. The details of inference networks are given in the sections that follow.

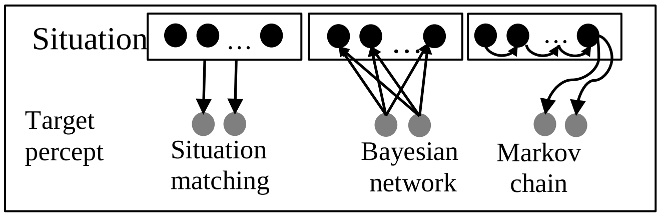


Figure 13. Possible Problem Formulations for Prediction.

The following subsections describe details of six prediction techniques that we implemented: two variants of situation matching, two variants of Bayesian network, a variable-order Markov model, and variance of single-scope blending. Each of these techniques represents the situation-target differently and uses different techniques to make predictions.

B. STATISTICAL LOOK-UP TABLE

A lookup table is a list of situation-target tuples. Two situation-target tuples (s_1,t_1) and (s_2,t_2) where s_1=s_2 will be merged to form one tuple (s_1,$t_1 \cup t_2$ in the lookup table. The situation-target tuples are added into a lookup table in the following manner. If the situation is new, a new entry is created that contain the situation-target tuple and inserted into the last row of the lookup table. If the situation already exists in the lookup table, the target is merged with the existing target percepts. Note that the target percepts are independent of one another given the situation of the tuple.

A statistical lookup table represents the set of situation-target as a lookup table, and searches the lookup table for a situation that exactly matches the current situation. If a match is found, the target percept with the highest number of occurrence is selected to be the predicted percept. If a match is not found, no prediction will be returned.

1. Learning for Statistical Lookup Table

A situation-target container C is a lookup table that contains a set of situation-target tuples $st_i = (s_i, t_i)$. Let S be a set of situations in C and Let $f_t(s_i)$ be a function that return the set of target percepts of s_i, i.e., a set of percepts t_i such that $(s_i, t_i) \in C$. Given the current percept t_c and the current situation s_c, we update container C based on the situation-target tuple $st_c = (s_c, t_c)$ as follows:

$$new\ C(st_c) = \begin{cases} C.add(st_c) & , \quad s_c \notin S \\ f_t(s_c).add(t_c) \wedge C.update(s_c) & , \quad s_c \in S, t_c \notin f_t(s_c) \\ f_t(s_c).update(t_c) \wedge C.update(s_c) & , \quad s_c \in S, t_c \in f_t(s_c) \end{cases}$$

Where:

$\mathrm{C.add}(\mathrm{st}_c)$ adds $\mathrm{st_c}$ into the container C.

$f_{\mathrm{t}}(\mathrm{s}_c).add(t_c)$ adds t_c as a target percept of s_c in $f_{\mathrm{t}}(\mathrm{s}_c)$

$f_{\mathrm{t}}(\mathrm{s}_c).update(t_c)$ updates the count of t_c in $f_{\mathrm{t}}(\mathrm{s}_c)$

$C.update(s_c)$ update the count of s_c in C

Essentially, if the current situation does not exactly match with any situation in the container, we add the situation-target into the container. If the current situation exactly matches with one situation in the container but the target does not match with any targets of the situation in the container, we add the target percept into the situation-target tuple and update all data count. If the current situation exactly matches with one situation in the container and the target exactly matches with one target of the situation in the container, we just need to update data count.

2. Prediction for SLT

Given the current situation s_c, generate the prediction as follow:

$$prediction(s_c) = \begin{cases} None & , \quad s_c \notin S \\ arg \max_{t \in f_t(s_c)} Prob(t|s_c) & , \quad s_c \in S \end{cases}$$

Essentially, if the current situation is in the container, we return the target percept with the highest count given s_c

3. Example for SLT

During learning, if the current situation is `{loc(Ed road +), loc(Fox2 road +), go(Fox2 east e)}`, it will match the last row in Figure 12. and the counter will increment from 1 to 2. If the target percept is found in the right column of the same row, the number of occurrence of the target will be incremented. If the target percept is not found, we will add the target percept into the right column of the same row. If the current situation is `{loc(Ed road +), loc(Fox3 road +), go(Fox3 east e)}`, it will not match with any situation in Figure 12. and will be added as a new row with count 1. The target percept will be added into the right column of the new row.

During prediction, if the current situation is `{loc(Ed road +), loc(Fox2 road +), go(Fox2 east e)}`, it will match the last row in Figure 12. and the prediction will be `loc(Fox2 road -).` If the current situation is `{loc(Ed road +), loc(Fox3 road +), go(Fox3 east e)},` it will not match with any situation in Figure 12. and no prediction will be returned. The probability of the prediction is computed as $Prob(t|s_c) = \frac{a}{b}$ where a is the number of occurrence of `loc(Fox2 road -)` as the target percept of `{loc(Ed road +), loc(Fox2 road +), go(Fox2 east e)}`, which is 1, and b is the number of occurrence of the matched situation, which is 1.

C. VARIABLE MATCHING (VM)

The Variable Matching technique replaces all constants in the predicate of a percept with variables. Instants that occur multiple time will be replaced by the same variable. The predicates are not replaced by variables. The matching of situations becomes the problem of variable matching by unification. A unification is a set of variable bindings, e.g. θ(α, β)={?a:?b, …} where variable ?a from situation α is bound to variable ?b in situation β. SUBST(θ, β) denotes the result of applying substitution θ to situation β. Two situations α and β are said to match if α = SUBST(θ, β). There is no prediction when there is no match between the current situation and any situation in the situation table. Finding matches is equivalent to a graph isomorphism problem.

An example of the constant to variable representation is shown in Figure 14. . The variable representation of Figure 14. is given in Figure 15. .

Constant	Variable
loc(Ed road +)	Loc(?x ?y +)
loc(Ed grass +)	loc(?x ?z +)

Figure 14. Constant versus Variables Representation.

{}	1	loc(?a ?b +)	1

{loc(?a ?b +)}	2	loc(?c ?b +)
		loc(?d ?b +)
			1
			1

{loc(?a ?b +) loc(?c ?b +)}	2	go(?c ?d e)	2

{loc(?a ?b +)
loc(?c ?b +)
goE(?c ?d e)}
	2	loc(?c ?b -)	2

Figure 15. Variables Representation of Figure 14. .

1. Learning for VM

A variablized situation-target is a situation-target with variables instead of constants. A variablized situation-target container C contains a lookup table of variablized situation-targets where each variablized situation-target $st_i = (s_i, p_i)$. Let S be a set of variablized situations that appears in C and let $f_t(s_i)$ be a function that returns the set of variablized target percepts of s_i, i.e., a set of variablized percepts p_i such that $(s_i, t_i) \in C$. Given the current variablized percept t_c and the current variablized situation s_c just prior to p_c, we update C based on the variablized situation-target tuple $st_c = (s_c, t_c)$ as follows:

$$new\ C(st_c) = \begin{cases} C.add(st_c) & , \quad s_c \notin S \\ f_t(s_c).add(t_c) \wedge C.update(s_c) & , \quad s_c \in S, t_c \notin f_t(s_c) \\ f_t(s_c).update(t_c) \wedge C.update(s_c) & , \quad s_c \in S, t_c \in f_t(s_c) \end{cases}$$

Where:

$\mathrm{C.add}(\mathrm{st_c})$ adds $\mathrm{st_c}$ into the container C.

$f_{\mathrm{t}}(\mathrm{s}_c).add(t_c)$ adds t_c as a target percept of s_c in $f_{\mathrm{t}}(\mathrm{s}_c)$

$f_{\mathrm{t}}(\mathrm{s}_c).update(t_c)$ updates the count of t_c in $f_{\mathrm{t}}(\mathrm{s}_c)$

$C.update(s_c)$ update the count of s_c in C

Essentially, if the variablized current situation does not exactly match with any variablized situation in the container, we add the variablized situation-target into the container. If the variablized current situation exactly matches with one variablized situation in the container but the variablized target does not match with any variablized targets of the situation in the container, we add the variablized target percept into the situation-target tuple and update all data count. If the variablized current situation exactly matches with one variablized situation in the container and the variablized target exactly matches with one variablized target of the situation in the container, we just need to update data count.

2. Prediction for VM

A variablized situation-target container C contains a lookup table of variablized situation-targets where each variablized situation-target $st_i = (s_i, t_i)$. Let $\theta(s_a, s_b)$ be a set of unifications that bind elements in situation s_a and s_b and $\mathrm{SUBST}(\theta, s_b)$ denotes the result of applying substitution θ to situation s_b. Let S be a set of variablized situations that appears in C and let $f_t(s_i)$ be a function that return the set of variablized target percepts of s_i, i.e., a set of variablized percepts t_i such that $(s_i, t_i) \in C$. Given the current situation s_c, generate the prediction as follow:

$prediction(\mathrm{s}_c) =$

$$\begin{cases} \mathrm{SUBST}\left(\theta, \arg\max_{t \in f_t(s_i)} \mathrm{Prob}(t|s_i)\right) & , \quad \exists\theta\exists s_i \in S, \mathrm{SUBST}(\theta, s_i) = s_c \\ \text{none} & , \quad \text{otherwise} \end{cases}$$

Essentially, if we can find a variablized situation that matches the current situation by unification, we return the substituted variablized target percept that occurs the highest number of time as the prediction.

3. Example for VM

During learning, if the current situation is `{loc(Ed road +), loc(Fox3 road +], go(Fox3 east e)}`, the variable representation `{loc(?a ?b +), loc(?c ?b +), goE(?c ?d e)}` will match the last row in Figure 15. and the counter will in increment from 2 to 3. The variable representation of the target percept will be added into the right column of the same row appropriately. If the current situation is `{loc(Ed road +), loc(Fox3 road +), loc(Fox3 east +)}`, it will not match with any situation in Figure 15. and will be added as a new row with count 1.

During prediction, if the current situation is `{loc(Ed road +), loc(Fox3 road +], go(Fox3 east e)}`, the variable representation `{loc(?a ?b +), loc(?c ?b +), goE(?c ?d e)}` will match the last row in Figure 15. with binding θ={Ed:?a, road:?b, Fox3:?c, east:?d} and the substituted prediction is `loc(Fox3 road -)`. If the current situation is `{loc(Ed road +), loc(Fox3 road +), loc(Fox3 east +)}`, it will not match with any situation in Figure 15. and no prediction will be returned

The variablized matching should find many more matches than the statistical lookup table when the number of new situation is high where the current situation is often not exactly match with any situation in the lookup table. While the variablized matching addresses the possible problem of too little data for variablized matching, it may find more misleading matches if the constants that are unified are not really compatible.

D. MULTIPLE SIMPLE BAYESIAN (MSB)

When the relational time-series is decomposed into a set of situation-target tuples, we can have one simple Bayesian network for each target percepts as the parent node, and the situation being the child nodes, effectively forming multiple simple Bayesian networks.

1. Learning for MSB

Let P be a set of percepts previously encountered. Each percept $p_i \in P$ has a set of children, formed by a set of situations s_i that belongs to the situation-target tuples in the lookup tables such that the target percept is p_i. Given a current situation-target tuple (s_c, t_c), the statistical properties of s_c and t_c are updated as follows:

Increment_count (t_c)

$\forall$p in s_c, Increment_count (p | t_c)

Essentially, we count the number of data points for the target percept and data points for each child percept in the current situation given the target percept

2. Prediction for MSB

Let C be a set of percepts previously encountered. Each percept p_i has a set of children, formed by a set of situations s_i that immediately precede p_i. Given the current situation s_c, generate the prediction as follow:

$$prediction(s_c) = \begin{cases} None & , \quad \forall p \in C, \mathrm{Prob}(p|s_c) = 0 \\ arg\max_{p \in C} Prob(p|s_c) & , \quad \text{otherwise} \end{cases}$$

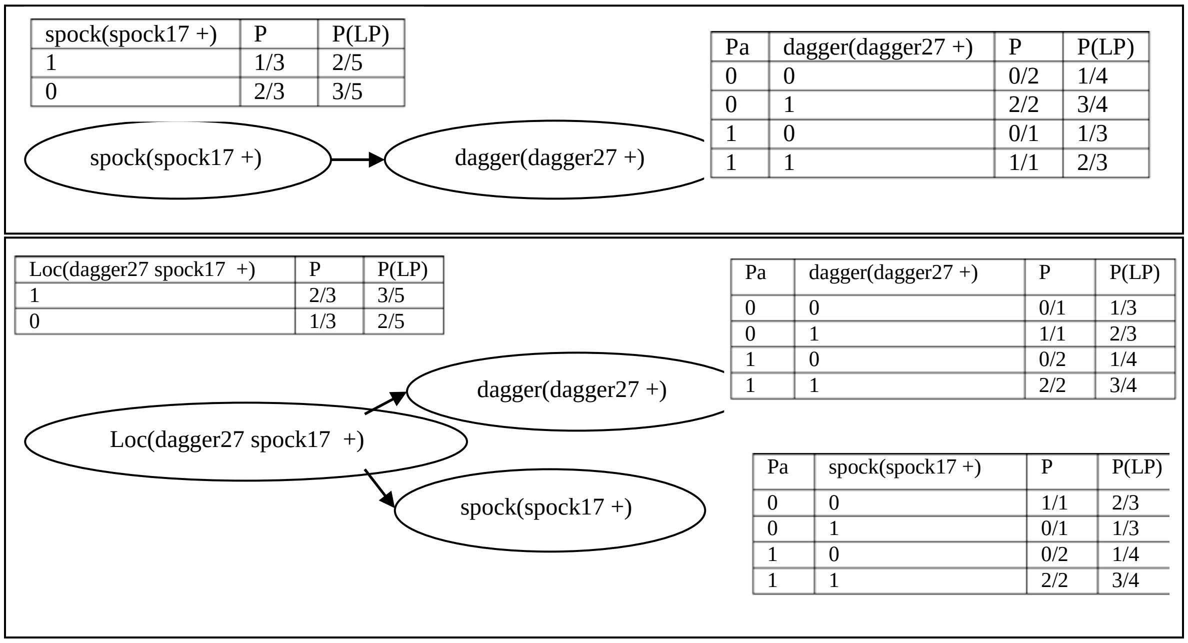
Essentially, we compute the probability of all parent percepts given the current situation and return the one with the highest condition probability as the prediction.

3. Example for MSB

During learning, we create a naïve Bayesian network for each distinct target percept. Note that different situation-target tuples that have the same target will be combined into the same simple Bayesian network, since they have the same target, which is the parent of the Bayesian network. The distinct target percept is the parent while the children are the percepts that have occurred at least once in the situations that immediately precede the parent. Supposed that we have 3 situations given in Figure 16. where percepts above '=>' belong to a situation and percepts below are the targets. There are two distinct target percepts, so we create 2 networks as shown in Figure 17. .

Situation1	Situation2	Situation3
dagger(dagger27 +)	dagger(dagger27 +)	dagger(dagger27 +)
	spock(spock17 +)	spock(spock17 +)
=>	=>	=>
spock(spock17 +)	Loc(dagger27 spock17 +)	Loc(dagger27 spock17 +)

Figure 16. Examples of Three Situations. Situation3 is a Repeat of Situation2.



spock(spock17 +)	P	P(LP)
1	1/3	2/5
0	2/3	3/5

Pa	dagger(dagger27 +)	P	P(LP)
0	0	0/2	1/4
0	1	2/2	3/4
1	0	0/1	1/3
1	1	1/1	2/3

Loc(dagger27 spock17 +)	P	P(LP)
1	2/3	3/5
0	1/3	2/5

Pa	dagger(dagger27 +)	P	P(LP)
0	0	0/1	1/3
0	1	1/1	2/3
1	0	0/2	1/4
1	1	2/2	3/4

Pa	spock(spock17 +)	P	P(LP)
0	0	1/1	2/3
0	1	0/1	1/3
1	0	0/2	1/4
1	1	2/2	3/4

Figure 17. Multiple Simple Bayesian networks for Figure 16. .

During prediction, we compute the probability of all parent percepts given the current situation. The parent percept with the largest conditional probability will be retuned as the prediction. Suppose that the current situation in Figure 18. .

Let A refers to dagger(dagger27 +)

Let B refers to spock(spock17 +)

Let C refers to Loc(dagger27 spock17 +)

Let P1 refers to parent spock(spock17 +)

Let P2 refers to parent Loc(dagger27 spock17 +)

We compute the probability of the parent as follow:

$$P(P1 = 1|A = 1) = \frac{P(A = 1|P1 = 1) * P(P1 = 1)}{P(A = 1|P1 = 1) * P(P1 = 1) + P(A = 1|P1 = 0) * P(P1 = 0)}$$

$$= \frac{\frac{1}{1} * \frac{1}{3}}{\left(\frac{1}{1} * \frac{1}{3}\right) + \left(\frac{2}{2} * \frac{2}{3}\right)} = \frac{1}{3} = 0.333$$

$$P(P2 = 1|A = 1) = \frac{P(A = 1|P2 = 1) * P(P2 = 1)}{P(A = 1|P2 = 1) * P(P2 = 1) + P(A = 1|P2 = 0) * P(P2 = 0)}$$

$$= \frac{\frac{2}{2} * \frac{2}{3}}{\left(\frac{2}{2} * \frac{2}{3}\right) + \left(\frac{1}{1} * \frac{1}{3}\right)} = \frac{2}{3} = 0.667$$

Current Situation 1
dagger(dagger27 +)

Figure 18. A Current Situation 1.

Current Situation 2
dagger(dagger27 +)
spock(spock17 +)

Figure 19. A Current Situation 2.

Suppose that the current situation is as shown in Figure 19. . We compute the probability of parent as follow:

$$P(P1 = 1|A = 1, B = 1) = \frac{P(A = 1|P1 = 1) * P(B = 1|P1 = 1) * P(P1 = 1)}{\sum_{x\in(0,1)} P(A = 1|P1 = x) * P(P1 = x)}$$

$$= \frac{\frac{2}{3} * \frac{1}{2} * \frac{1}{3}}{\left(\frac{2}{3} * \frac{1}{2} * \frac{1}{3}\right) + \left(\frac{3}{4} * \frac{1}{2} * \frac{2}{3}\right)} = \frac{4}{13}$$

$$P(P2 = 1|A = 1, B = 1) = \frac{P(A = 1|P2 = 1) * P(B = 1|P2 = 1) * P(P2 = 1)}{\sum_{x\in(0,1)} P(A = 1|P2 = x) * P(B = 1|P2 = x) * P(P2 = x)}$$

$$= \frac{\frac{2}{2} * \frac{2}{2} * \frac{2}{3}}{\left(\frac{2}{2} * \frac{2}{2} * \frac{2}{3}\right) + \left(\frac{1}{1} * \frac{0}{1} * \frac{1}{3}\right)} = 1$$

4. Effect of Number of Child Nodes

We used the benchmark environment in [8] and vary the time window for 40 sequences of 100 percepts. The variation of the prediction accuracies as a function of time window are as shown in Figure 20. . It appears that the larger the time window, the lower the prediction accuracy. A larger time window is expected to perform poorer because child percepts that are further away in time are more likely to be independent of the current percept.

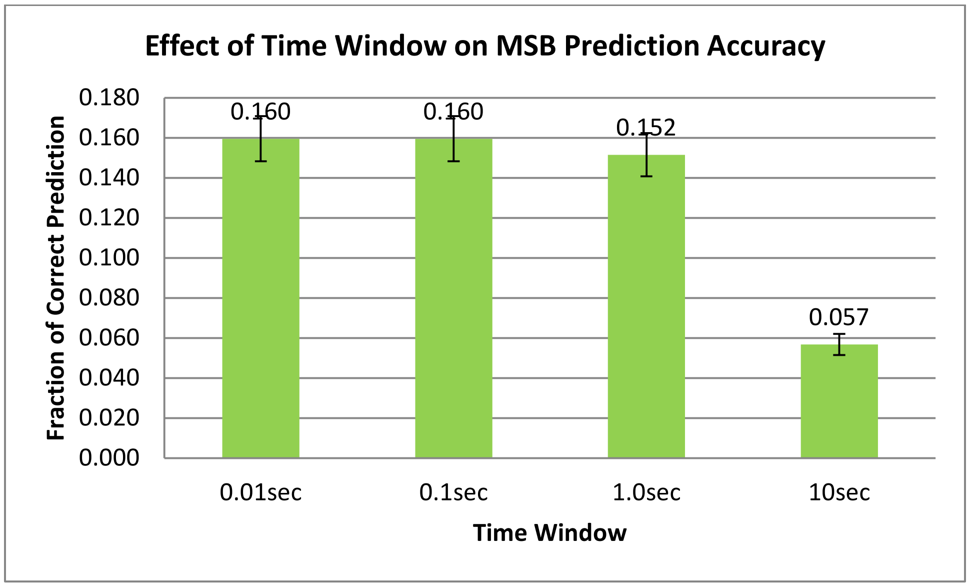


Figure 20. Effect of Number of Children for Multiple Simple Bayesian on Pymud.

E. SIMPLE BAYESIAN MIXTURE (SBM)

Simple Bayesian mixture is an improvement over multiple simple Bayesian to allow it to learn certain functions such as Exclusive-OR. A simple Bayesian mixture contains probability mixture densities, constructed by normalizing a linear combination of two or more Bayesian networks probability densities having the same parent and child percepts. In multiple simple Bayesian, we have one distribution for one parent-child network. In Simple Bayesian mixture, the same distribution for one parent-child network is divided into several weighted distributions. Simple Bayesian mixture is implemented using the Estimate and Maximize (EM) algorithm.

1. Learning for Simple Bayesian Mixture

In multiple simple Bayesian, we compute $\forall p \in C, Prob(p|s_c)$ where C is a set of all target percepts. In Simple Bayesian Mixture, we have multiple distributions C_i for each simple Bayesian network. $\forall p \in C_i$ compute $\sum_{i=1}^{k}, a_i Prob(p|s_c)$. a_i is the weight of a distribution, so $\sum_{i=1}^{k} a_i = 1$. Each data point (one situation-target st_i) is fractionally assigned to the distributions in C. The fractional assignment for a data point x in each distribution j is computed as:

$$f_j(x) = \frac{a_j Prob_j(x)}{\sum_i^C a_i Prob_i(x)}$$

The weight a_j for distribution C_j and total number of data point N is updated:

$$a_j = \sum_x \frac{f_j(x)}{N}$$

The distributions in C are allocated using an Estimate and Maximize (EM) algorithm. Given a new data point x and the number of distributions k (initialized to zero):

Estimate:

If k is zero, skip this step.

Otherwise compute the fractional assignment of x as above

Maximize:

If the spawn rule dictates, create a new distribution based on x with a fractional assignment of one.

Otherwise, update all components using x and fractional assignments.

Spawn Rule: $\sum_i^C a_i Prob_i(x) < \frac{1}{2^{k+1}}$

2. Prediction for SBM

$$prediction(s_c) = \begin{cases} None & , \quad \forall p \in C, \sum_i^C a_i Prob_i(x) = 0 \\ \underset{p \in C}{argmax} \sum_i^C a_i Prob_i(x) & , \quad \text{otherwise} \end{cases}$$

3. Effect of Number of Distributions

We used the benchmark environment in [8] and vary the number of distributions for 40 sequences of 100 percepts. The variations of the prediction accuracy and computation time over number of distribution are as shown in Figure 21. and Figure 22. . We observe that SBM2 produces the optimal results since there is insignificant difference between SBM2, SBM3, SBM4 and SBM5, but SBM2 has the shortest computation time as compared to SBM3, SBM4 and SBM5.

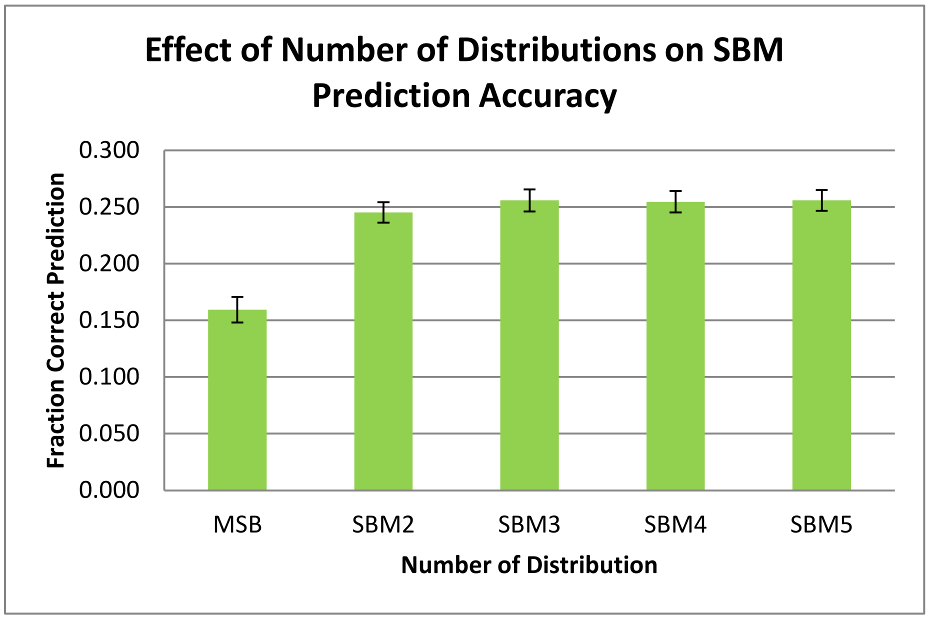


Figure 21. Effect of Number of Distribution on Accuracy for Simple Bayesian Mixture. SBM2Means SBM with Two Distributions.

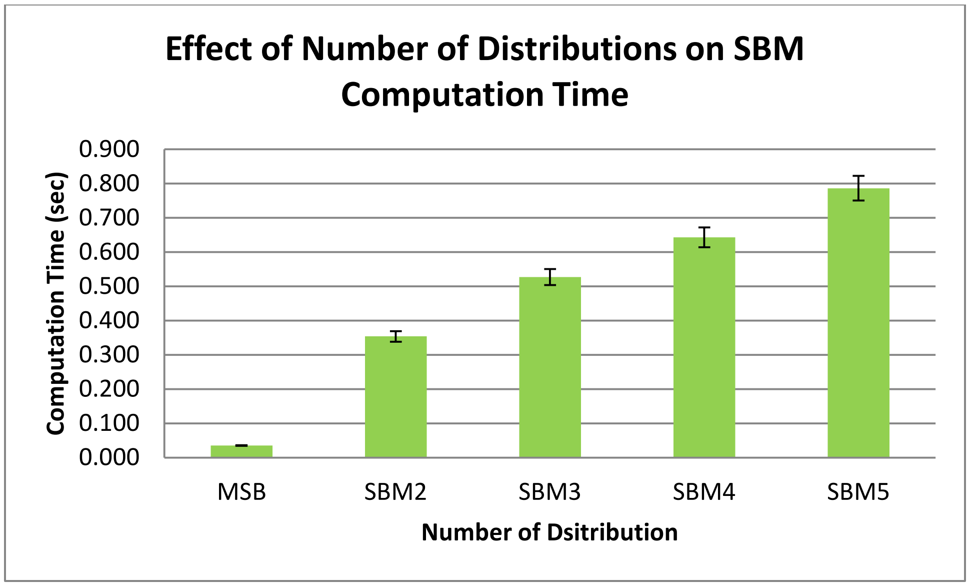


Figure 22. Effect of Number of Distribution on Time for Simple Bayesian Mixture.

F. VARIABLE-ORDER MARKOV MODEL

A variable-order Markov model is an extension to the Markov chain models in which a variable order is used in place of a fixed order. A Markov chain is chain of finite states such that each state transition respects the Markov property, which means that the probability distribution of future state conditioned on the present state is independent of earlier past states. It is a Markov process with a discrete state space. If the order is one, each state depends on the most recent state. If the order is two, each state depends on the two most recent states in a fixed sequence. If the order is variable, each state can depend on different number of states in a fixed sequence. Hence, it is called Variable-Order Markov model (VOMM). A relational time-series of order n is $x^n = x_1, x_2, \ldots, x_{n-1}, x_n$ where n is the order.

We implemented a VOMM model using context trees [55]. Suppose we have a letter sequence: 'A', 'B', 'R', 'A', 'C', 'A', 'D', 'A', 'B', 'R', 'A'. Each letter can represent each percept in our domain. We can build a maximum 2-order context tree as shown in Figure 23. . Any path from the root to any node denotes a reversed Markov chain. For example, the path [Root, ‘A’, ‘R’] denotes a Markov chain ‘RA' with a target ‘C’. Another example: the path [Root, ‘A’] denote a Markov chain ‘A’ with target percepts ‘B’, ‘C’ and ‘D’. The number beside the target percept denotes the number of occurrence. Note that a path from the root note to any node denotes a single previous situation.

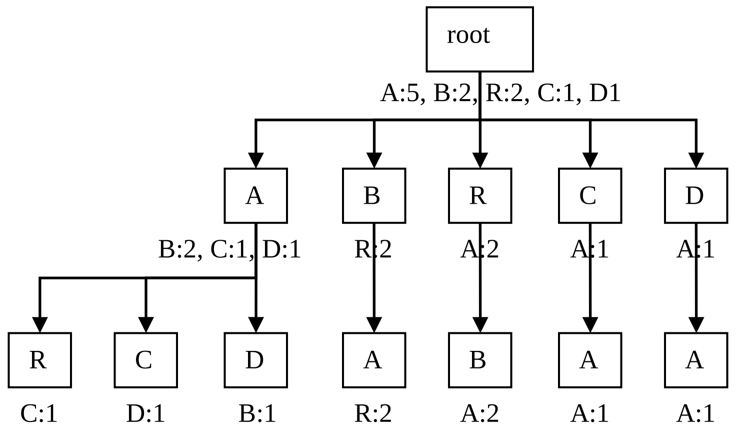


Figure 23. Maximum Order-2 Context Tree for Letter Sequence A-B-R-A-C-A-D-A-B-R-A.

1. Learning for VOMM

Let C be a context tree that store a set of situation-target $st_i = (s_i, p_{ij})$ as branches from the root to the leaf nodes as shown in Figure 23. . The nodes that are one hop away from the root are situations of Markov model order 1. The characters underneath each node are the target percepts while the number represents the number of occurrences of the percept given the node.

Given the current situation $s_c = [p_1p_2…p_{n-1}p_n]$ and the target percept p_c, we store the situation-target $st_c = (s_c, p_c)$ by looking for p_n among the child nodes of root node, p_{n-1} among the child nodes of p_n, until p_1 among the child nodes of p_2. If we are able to find the entire $[p_1p_2…p_n]$, we just add p_c at the deepest node down the path, which is p_1, if p_c does not already exist as target percept, or increment its count otherwise. The current percept p_c is also added at each level of the path to enable variable-order Markov mode. If we are unable to find a $[p_1p_2…p_n]$ branch, we just create a new branch at the point when the Markov chain differs. For example, given $s_c = [p_1p_2]$, if we can find p_2 at the first level but not p_1 at the second level, we create a new p_1 node and append it as child node of p_2 node, and add the p_c at the p_1 node.

2. Prediction for Variable-order Markov Model

Let C be a context tree that store a set of situation-target $con_i = (s_i, p_{ij})$ as branches from the root to the leaf nodes as shown in Figure 23. . Given the current situation $s_c = [p_1 p_2 \ldots p_n]$, we traverse down the tree by looking for p_1 at the first level, p_2 at the second level, until the nth level. If p_n is reached, return the percept at the node with the highest count. If p_n cannot be reach reached, return a percept at the lowest level reached.

3. Example for VOMM

Let us assume that the alphabets in the context tree in Figure 23. represent percepts. During learning, if the current situation contains a single percept 'A', we will traverse the tree from root to 'A' and add the target percept in 'A'. If the current situation is 'DA', we traverse the path 'A', 'D' and add the target percept in 'D'. If the current situation is 'BA', we will not find the path 'A', 'B'. Hence, we will add another node 'B' under 'A' and add the new target percept inside 'B'. If the current situation is 'E', we create a new node under the root.

During prediction, we try to maximize the order matching. If the current situation is "DA," we traverse down the path "A," "D" and return "B" as the prediction. If the current situation is "ZA," we will traverse down the path "A." Since we cannot find 'Z' in 'A', we stop at 'A' and return 'B' as the prediction.

4. Effect of Maximum Order

We used the benchmark environment in [8] and vary the maximum order for 40 sequences of 100 percepts. The environment is controlled to remove a random component from the sequence used in the actual experiments. The prediction performances as a function of maximum order are shown in Figure 24. . We observe that the results plateau at maximum order 6, though statistically insignificant. We use a maximum order of 10 for the benchmark comparison later.

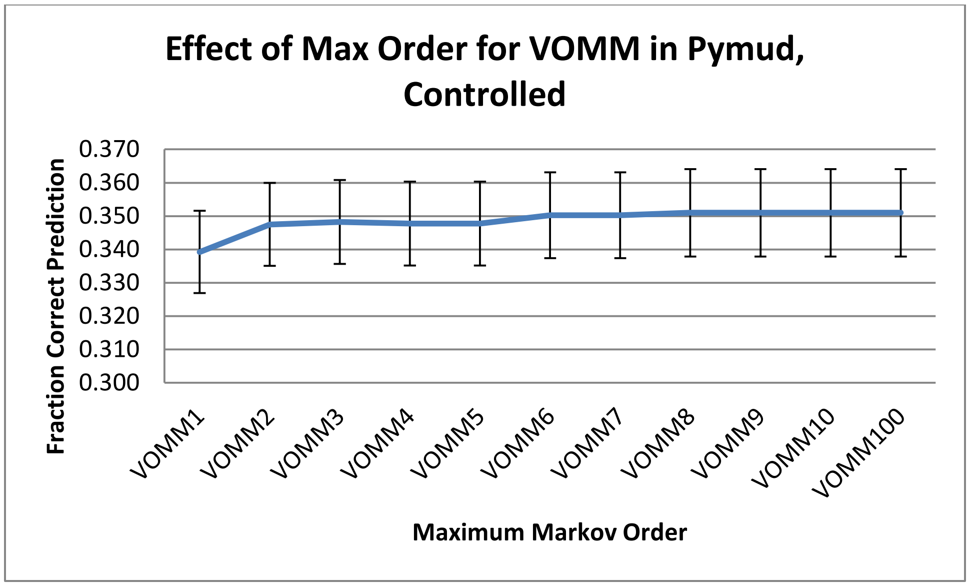


Figure 24. Effect of Maximum Order for Variable-Order Markov Model.

G. SINGLE-SCOPE BLENDING

The above prediction techniques are mainly activation approaches, other than variable matching. The activation approaches activate previously seen percepts as predictions. Such approaches cannot predict new percepts that have not been encountered. The variable matching technique, based on graph isomorphism, can generate a new percept for prediction. However, it requires full graph matching, which is susceptible to noise and complex environment in which most situation encountered are new. The proposed approach is to look at current cognitive science theory on how human creativity can be modeled. In the next section, we will explore the theory of Conceptual Blending [42]. This theory explains the human creative process, which may help to improve the prediction accuracy in unknown environments.

1. Overview

The computational models of the single-scope blending described in this section are built on the definitions given in the chapter on previous work and situation learning. Recall that situation learning transform the relational time-series into a set of situation-target tuples. The prediction task is to predict the next percept given the current situation and the set of situation-target tuples.

a. *Learning for =Single-scope Blending*

Single-scope blending uses the lookup table as in the statistical lookup table prediction techniques. A situation-target lookup table C contains a lookup table of situation-target tuples $st_i = (s_i, t_i)$. Let S be a set of situations s_i in C and let t_i be a set of target percepts of s_i. Given the current percept t_c and the current situation s_c, we store the situation-target tuple $st_c = (s_c, t_c)$ as follow:

$$new\ C(st_c) = \begin{cases} C.add(st_c) & , \quad s_c \notin S \\ f_t(s_c).add(t_c) \wedge C.update(s_c) & , \quad s_c \in S, t_c \notin f_t(s_c) \\ f_t(s_c).update(t_c) \wedge C.update(s_c) & , \quad s_c \in S, t_c \in f_t(s_c) \end{cases}$$

where:

$\mathrm{C.add}(\mathrm{st}_c)$ adds $\mathrm{st_c}$ into the container C.

$f_{\mathrm{t}}(\mathrm{s}_c).add(t_c)$ adds t_c as a target percept of s_c in $f_{\mathrm{t}}(s_c)$

$f_{\mathrm{t}}(\mathrm{s}_c).update(t_c)$ updates the count of t_c in $f_{\mathrm{t}}(\mathrm{s}_c)$

$C.update(s_c)$ update the count of s_c in C

b *Prediction for SSB*

Let C contains a lookup table of situation-target $st_i = (s_i, t_i)$. Let S be a set of situation that appears in C and let T be a set of target percepts of $s_i \in S$. Let Σ_i be the unification space that contains all possible unification between $s_i \in S$ and s_c. Let $\theta_i(s_i, s_c) \in \Sigma_i$ be the set of unification of the object constants in one previous situation $s_i \in S$ and current situation s_c. Let $Subst(\theta_i, \beta)$ be a function that substitutes the object constants in situation β using θ_i so that Subst(θ_i,β) = α = f(β). Let T_i be the target set of $s_{i.}$ Given the current situation s_c, generate the prediction as follow:

$$Prediction(s_c) = \underset{S_i \in S, \theta_i \in \Sigma i}{\arg\max}\big(SIMSCORE(subst(\theta_i, s_i), s_c)\big) \wedge subst\left(\theta_i, \underset{t_i \in T(s_i)}{\operatorname{argmax}} Prob(t_i | s_i)\right)$$

Essentially, we first find a situation among all previous situation $s_i \in S$ and a set of unification $\theta_i \in \Sigma_i$ that maximize the similarity score by $arg\max_{S_i \in S}\big(SIMSCORE(subst(\theta_i, s_i), s_c)\big)$. Then we find the highest probable target $t_i \in T_i$ such that $arg\max_{t_i \in T_i} probability\,(t_i|si)$. Next, we apply substitution on t_i: $t_i' = subst(\theta_i, t_i)$ and return t_i'

The high-level algorithm of single-scope blending that is implemented in this dissertation is described in **Algorithm 1:** . The for-loop in **Algorithm 1:** compares all previous situations with the current situation. Each previous situation takes turn to be the input space 1 while the current situation is the input space 2. The `generic()` function models the process of identifying common percepts for the generic space and generate a similarity score between input space 1 and 2. In the process, `generic()` also return a set of object constant bindings between the two input spaces. The previous situation with the highest similarity score will be selected as the actual input space 1 and the bindings are applied onto the selected situation to generate the prediction.

Algorithm 1: Single-scope blending

Input Con	A set of situation-targets st = { (s_i,ti) } where s_i is past situation and t_i are target percepts of s_i
Input s_c	Current situation
Description	Implement Definition 11

```
maxScore ← -1
maxGenericSpace = None
bestInputSpace1 = None
inputSpace2 ← $s_c$
for each $st_i$ in st
    inputSpace1 ← $st_i.s_i$
    genericSpace, θ ← generic (inputSpace1, inputSpace2)
    if genericSpace.SIMSCORE()> maxScore
        maxScore = genericSpace.SIMSCORE()
```

```
            bestInputSpace1 = inputSpace1
blend = generateBlend(θ, bestInputSpace1)
prediction = blend.targetPercept()
return prediction
```

The generic()function looks for unification of object constants from two input spaces, which is equivalent to a subgraph isomorphism problem. Algorithm 2 describes an implementation of the generic() using a backtracking process. The algorithm starts with the smallest possible common subgraph that can be found in both situations and add bindings into the common subgraph if the nodes in the bindings shared some structure properties. This algorithm is used in [41]. Algorithm 2a describes the algorithm using high level pseudo code while 2b is closer to the actual codes implemented on Python.

Algorithm 2a: Generic_backtrack (simplified)

Input s_1	Situation 1
Input s_2	Situation 2
Description	Take one constant from each situation and bind them. If there is at least one common edge, keep this mapping and continue to other unmapped constants. Otherwise, discard this mapping.

```
Function generic (s1, s2)
  Create root node of unbind constants
  Iteration
    Expand node on all possible constant bindings
    For each new binding,
      If no additional common percept found: continue
      Create new node of unbind constants
  Select the leave nodes with highest common percept count
Return best bindings and common percept count
```

Algorithm 2b: Generic_backtrack (Python style pseudo codes)

```
Function generic (s1, s2)
SC1 ← {s1 constants}
SC2 ← {s2 constants}
fringe ← [], θ ←[]
unmapped1 ← SC1
unmapped2 ← SC2
numSimilarAtoms  ← 0
fringe.append([unmapped1, unmapped2, numSimilarAtoms , θ])
solutionList ←[]
while fringe not empty:
  [unmapped1,unmapped2,numSimilarAtoms,θ]←fringe.pop()
  if unmapped1 is empty or unmapped2 is empty:
     solutionList .append(θ,numSimilarAtoms )
     continue
    for c1 in unmapped1:
     for c2 in unmapped2:
      newU1 = copy(unmapped1)
      newU1.remove(c1)
      newU2 = copy(unmapped2)
      newU2.remove(c2)
      score = COMMON(c1, c2, s1, s2, θ)
      newθ = copy(θ)
      θ.append( (c1, c2)   )
      if score == 0:
         continue
      numSimilarAtoms += score
      fringe.append([newU1,newU2,numSimilarAtoms,θ])
   (θ*,score)=element of solutionList with largest numSimilarAtoms
   similarityScore = SIMSCORE (s1,s2, numSimilarAtoms)
return similarityScore, θ*
```

Please refer to Figure 25. for an illustration of the backtracking implementation given in Algorithm 2b. The algorithm starts with a permutation of possible bindings. Each binding corresponds to a node in the tree in Figure 25. . These nodes are pushed into the fringe, which is a stack that holds the yet-to-process nodes. Each node in the stack is popped, evaluated, and other possible bindings are then added into the stack. The evaluation is a simple counting of the number of common percepts in both situations if the binding is accepted. For example, if we accept the (x1:x4) binding in the first level, we have an additional score of one because the binding produces one common percept, which is troll+(x1) and troll+(x4). There are additional two possible bindings (x2:x5) and (x3:x5) consistent with this one. The (x2:x5) binding does not contribute any additional common percepts. Hence, the additional score is zero. Since there is no other unmapped nodes in concept 2 for x3, the algorithm backtracks to evaluate (x3:x5), which contributes two common percepts in conjunction with (x1:x4). After evaluating (x3:x5), (x3:x5)is popped. Since there is no common percept for (x3:x5), the algorithm backtracks. A flow from the root to the leaf node constitutes one solution of bindings. The total score for a solution describes the similarity score of both situations. The path [(x1:x4), (x2:x5)] has a score of one while the path [(x1:x4), (x3:x5)] has a score of three. The path with the highest possible score indicates the maximum possible similarity score between two situations. It is possible that more than one path has the same similarity score.

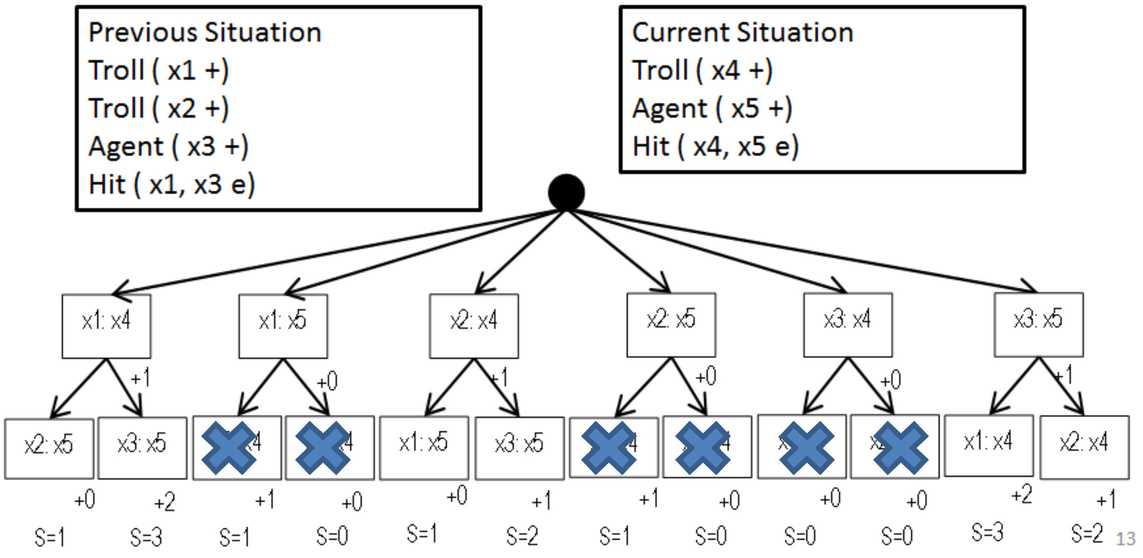


Figure 25. Backtracking Partial Matching process.

2. Backtrack with Heuristics

There are two problems with the above backtrack method. First, Algorithm 2 assumed a connected graph. A situation of percepts can be modeled as a graph: Object constants are the nodes while the relations are the edges of the graph. Falkenhainer [41] would remove isolated nodes to speed up processing. Our situations, when converted into a graph, are often not fully connected. For example, suppose we have another percept in both situations: (hit x6, x7), which is not connected to the other part of the graph. Suppose we have a common subgraph with the current binding (x1:x4). When the new binding (x6:x6) is attempted, there is no common percept and (x6:x6) will be ignored. This will cause suboptimal result in the prediction problem. The second problem is thatthere can be more than one set of bindings that will achieve the same similarity score. Many nonsensical bindings can results in same similarity score. For example, spock mapped to pitchfork, and both are at location xyz.

A more robust backtracking search is to remove the backtrack rule of algorithm 2 line 23 and 24, which is used by McGregor [56]. The effect is the same as the tree search described in Figure 25. , but with the blue cross removed. However, McGregor's [56] approach has a serious computational complexity problem since it literally searches the entire search space before determining which path to the leaf nodes is the best path of bindings. We introduce the following heuristics to improve the complexity. The improved backtrack algorithm is given in Algorithm 3a and 3b. Algorithm 3a describes the algorithm using high-level pseudo code, while 3b is closer to the actual code implemented on Python.

a. Type Check Heuristics

McGregor [56] proposes to use heuristics to prune the search tree by checking if the current binding is a valid partial solution. We can use a similar heuristics by checking for invalid constant type binding. For example, a constant of type "pitchfork" cannot be bind with a constant of type 'place."

b. Termination Condition

A new termination condition is introduced when a maximum score is found. Termination condition: g == min ($|s_1|$, $|s_2|$) where g is the score that counts the number of common percepts in situation s1 and situation s2. $|s_i|$ gives the number of percepts in the situation. For example, if situation s_1 has 3 percepts, while situation s_2 has 10 percepts, the maximum number of common percepts is 3. So, if we found a set of bindings that provides 3 common percepts, the algorithms can terminate.

Algorithm 3a: Similarity_Backtrack_Heuristic (simplified)

Input s_1	Situation 1
Input s_2	Situation 2
Description	Take one constant from each situation and map them. If there is at least one common edge, keep this mapping and continue to other unmapped constants. Otherwise, discard this mapping.

```
Function generic (s1, s2)
  Create root node of unbind constants
  Iteration
    Expand node on all possible constant bindings
    For each new binding,
      If different type: continue
      Create new node of unbind constants
      if maximum possible score found: break from loop
  Select the leave nodes with highest common percept count
Return best bindings and common percept count
```

Algorithm 3b: Similarity_backtrack_heuristic (Python-style pseudo codes)

```
SC1 ← {s1 constants}
SC2 ← {s2 constants}
fringe ← []
θ ←[]
unmapped1 ← SC1
unmapped2 ← SC2
numSimilarAtoms  ← 0
fringe.append([unmapped1, unmapped2, numSimilarAtoms , θ])
solutionList ←[]
while fringe not empty:
  [unmapped1,unmapped2,numSimilarAtoms,θ] ← fringe.pop()
  if unmapped1 is empty or unmapped2 is empty:
    solutionList .append(θ,numSimilarAtoms )
    continue
  for c1 in unmapped1:
    for c2 in unmapped2:
      if c1.type() ≠ c2.type()
            continue
      newU1 = copy(unmapped1)
      newU1.remove(c1)
      newU2 = copy(unmapped2)
      newU2.remove(c2)
      score = countCommonAtom(c1, c2, s1, s2, θ)
      newθ = copy(θ)
      θ.append( (c1, c2) )
      numSimilarAtoms += score
      fringe.append([newU1, newU2, numSimilarAtoms, θ])
      if numSimilarAtoms == min(|s1|,|s2|)
            solutionList .append(θ,numSimilarAtoms )
            Clear fringe
            terminate
(θ*,score)=element of solutionList with largest numSimilarAtoms
similarityScore = 2 * numSimilarAtoms  / (|s1| + |s2|)
return similarityScore, θ*
```

3. Semi-Greedy Best-first Search

We introduce an algorithm that performs significantly faster than algorithm 4 (backtrack with heuristics). The algorithm is given in algorithm 5. The differences between the algorithm 4 and 5 are given below.

a. Best-first Search

The fringe of the algorithm is implemented as a priority queue so that paths with higher potential will be given a higher priority. This is to allow a higher priority path to hit the termination condition faster than the lower priority path. An example is given in Figure 26. .

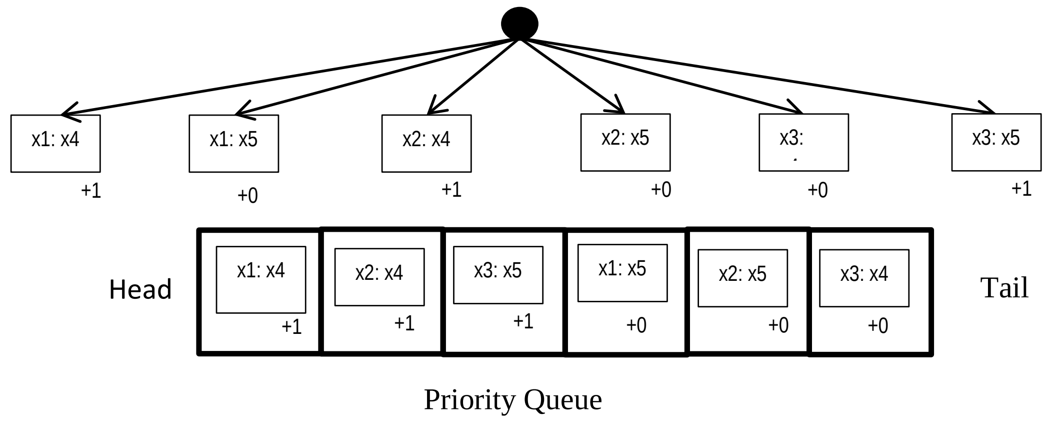


Figure 26. State of the Priority Queue after Processing the First Level of the Tree.

b. Greedy

Decrease fringe size to minimal (1 or 2). This has the same assumption as the above that if the front node has the best path, there is no need to explore other path. This is incomplete. Nevertheless, it can be shown that the prediction accuracy results of fringe size of 2 are statistically insignificant when compared to the complete search.

c. Semi-Greedy

Experiment shows that fringe size 2 achieves similar result as depth-first search and best-first search. Instead of fixing a fringe size as 1, we can vary it by allow paths with 2 highest scores to remain in the fringe.

d. Potential Filtering

Before the similarity score is computed, we compute its maximum value by the equation: p = `2 * min(|s1| + |s2|) / (|s1| + |s2|).` If p is smaller than the current global highest similarity score, computed from other situation, we can skip the current computation.

e. Integrated Mirror and Single-scope Blending

If two situations are exactly the same, expensive similarity computation can be avoided. Mirror and single-scope blending are integrated by first checking for exact matching. If there is exact matching, mirror scope is run. If there is no exact matching, single scope is run.

Algorithm 5a: Best-first search (simplified)

Input s_1	Situation 1
Input s_2	Situation 2

```
Function similarity (s1, s2)
  Create root node
  Iteration
    Expand tree on best node
    Compute SimilarityScore
    If maximum possible score found:
      break
  Keep the best scoring nodes
Return best bindings and similarity score
```

Algorithm 5b: Best-first search (Python style pseudo codes)

Input s_1	Situation 1
Input s_2	Situation 2
Input g_{global}	Current global highest score

```
SC1 ← {s1 constants}
SC2 ← {s2 constants}
If 2 * min(|s1| + |s2|) / (|s1| + |s2|) < $g_{global}$
      return
fringe ← []
maxFringeSize = 2
θ ←[]
unmapped1 ← SC1
unmapped2 ← SC2
g,h,f ← 0
fringe.append([unmapped1, unmapped2, g,h,f, θ])
solutionList ←[]
solutionHighestScore ← 0
while fringe not empty:
      [unmapped1, unmapped2, g,h,f, θ] ← fringe.pop()
      if unmapped1 is empty or unmapped2 is empty:
            solutionList .append(θ,g)
            solutionHighestScore = max(g, solutionHighestScore)
            continue
      for c1 in unmapped1:
            for c2 in unmapped2:
                  if c1.type ≠ c2.type: continue
                  newU1 = copy(unmapped1)
                  newU1.remove(c1)
                  newU2 = copy(unmapped2)
                  newU2.remove(c2)
                  g += countCommonAtom(c1, c2, s1, s2, θ)
                  θ.add((c1, c2))
                  if g == min(|s1|,|s2|)
                        solutionList .append(θ,numSimilarAtoms )
                        Clear fringe
                        terminate
                  h = heuristicsScore()
                  f = g + h
                  If f < solutionHighestScore
                        continue
                  fringe.append([newU1, newU2, g,h,f, θ])
      fringe.sort(by f)
      Trim fringe based on max fringe size
(θ*,score) = element of solutionList with largest numSimilarAtoms
similarityScore = 2 * g / (|s1| + |s2|)
return similarityScore, θ*
```

4. Attention-based Binding

We improve the greedy best-first search algorithm by looking at how the human eye focuses its attention during a visual search process, by targeting the high salience property first [57], [58]. The idea is to generate a pairing score when we pair two constants, based on its type and in and out degree. Two similar constant must have similar types and in and out degree. The Attention algorithm is given in Algorithm 6 and 7.

Algorithm 6: Attention (Python-style pseudo codes)

Input s_1	Situation 1
Input s_2	Situation 2
Description	Form all possible pairs from s1 and s2. Each pair has a similarity score based on name, type, in and out degree. Lexical sort them by pair similarity score. For unification mapping starting from the most similar pair

```
SC1 ← {s1 constants}, in and out degree
SC2 ← {s2 constants}, in and out degree
pairs ← getPairing(SC1,SC2)
g1AddedTerms ← {}
g2AddedTerms ←{}
path ← []
score ← 0
for [c1, c2, pairScore] in pairs:
    if not c1 in g1AddedTerms and not c2 in g2AddedTerms:
        g1AddedTerms[c1] = True
        g2AddedTerms[c2] = True
```

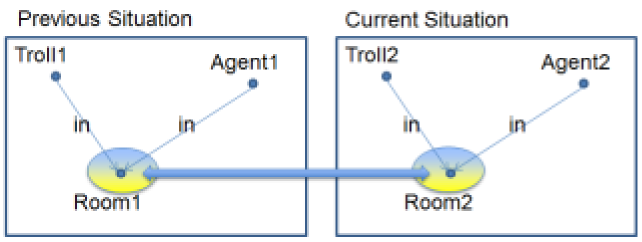
```
            θ.append([c1, c2])
numSimilarAtoms   = countCommonAtom(c1, c2, s1, s2, θ)
similarityScore = 2 * numSimilarAtoms  / (|s1| + |s2|)
return similarityScore, θ
```

Algorithm 7: getPairing

Input SC1	{s1 constants}, in and out degree
Input SC2	{s2 constants}, in and out degree
Input s_1	Situation 1
Input s_2	Situation 2
Description	Form pairings between constants in s1 and s2 if they have the same type. Compute score as [ExactNameMatch, typeScore, BothExactDegreeMatch, AtLeastOneDegreeMatch, DegreeDiff]

```
pairings ← []
for $C_1$ in SC1:
      for {$C_2$}in {$C_{2i}$}:
            #typeScore
            if $C_1$.type == $C_2$.type
                  typeS ← 1
            else
                  continue
            #exact match
            if g1Term == g2Term:
                  ExactS = 1
            else:
                  ExactS = 0
            #BothExactDegreeMatch
            outDiff= |outdeg($C_1$) - outdeg($C_2$)|
            inDiff= |indeg($C_1$) - indeg($C_2$)|
            if outDiff == 0 and inDiff == 0:
                  inoutS = 1
            else:
                  inoutS = 0
            #AtLeastOneDegreeMatch
            if outDiff == 0 or inDiff == 0:
                  oneS = 1
            else:
                  oneS = 0
            #DegreeDiff
            DegreeDiff = -(inDiff + outDiff)
            score = [ExactS, typeS, inoutS, oneS, DegreeDiff]
            pairings.append([$C_1$, $C_2$, score])
pairings.sort(key=lambda x: x[2], reverse=True)
return pairings
```

We will illustrate the attention-based algorithm using the example given in Figure 27. . We have two situations at the top left and right diagram in Figure 27. . For each constant in the two situations, we construct the structural properties table as shown in the table at the center of Figure 27. . The structural properties are compared in the bottom table. The score is a 5-tuple that indicates: (1) both constants of the same type, (2) both constants share the same name, (3) both constants have exactly the same in and out degree, (4) both constants similar in either in and out degree, and (5) total degree different. The pairs in the bottom table are sorted lexically by the score. The pair that appears at the top will be selected. The next pair will be selected if none of the constants in the pair has been selected.



constants	**In Degree**	**Out Degree**	**Type**
Troll1	0	1	D
Agent1	0	1	A
Location1	2	0	L
Troll2	0	1	D
Agent2	0	1	A
Location2	2	0	L

constant1	**constant2**	**Score**
Dragon - 1	Dragon2	[1, 0, 1, 1, 0]
	Agent2	[0, 0, 1, 1, 0]
	Location2	[0, 0, 0, 0, -3]
Agent - 1	Dragon2	[0, 0, 1, 1, 0]
	Agent 2	[1, 0, 1, 1, 0]
	Location2	[0, 0, 0, 0, -3]
Location - 1	Dragon2	[0, 0, 0, 0, -3]
	Agent2	[0, 0, 0, 0, -3]
	Location2	[1, 0, 1, 1, 0]

score = [Type, ExactNameMatch, BothExactDegreeMatch, AtLeastOneDegreeMatch, DegreeDiff]

Figure 27. Illustration of Attention-based search.

H. PERFORMANCE MEASUREMENT

In the experiments that follow, we use two metric of measuring the performance of each prediction techniques: prediction accuracy and run time. The predicted percept $p = r(c_1, c_2, \ldots, c_m)$ is said to be correct if the next percept $p' = r'(c_1', c_2', \ldots, c_m')$ is such that $p' = p, r' = r, c_i' = c_i$ for all $i = 1,2, \ldots, m$. Prediction accuracy $= \frac{c}{n}$ where n is the number of percept receive and c is the count of correct prediction. The run time is simply the time to complete each experiment of predicting all percepts in each relational time series.

Note that the prediction accuracy measurement does not include time prediction, but is based on next simplified percept only. A prediction may not occur immediately and another percept may arrive before the predicted percept arrives. There are several ways to consider prediction with a time interval. We can use a constant time interval and or time of occurrence between situation and target. For second method, we can collect the intervals compute mean and standard deviation (SD). If the predicted percept falls within 1SD, 2SD or 3SD, we can consider it correct. There are two ways of collecting the interval, from situation's perspective or target's perspective. As a result of all these complexity, we fall back on the strictest measure of effectiveness: predict the next percept. Prediction with time is studied in the sensitivity analysis in chapter 0 where 2SD is used on target's perspective.

A given situation can have multiple target percepts (possible predictions). Different prediction techniques are capable of different number of prediction. For Bayesian inference, all previously encountered percepts are possible. For Markov inference, the lower the order, the greater number of prediction. Situation-matching prediction depends on the number of target percepts. Hence, to be fair, we asked each technique to produce their best guess.

I. CONCLUSION

In this chapter, we describe six prediction algorithms. After a relational time-series has been decomposed into a set of situation-target tuples, we can apply different kind of prediction techniques in the inference process. The prediction techniques discussed are statistical lookup table, variable matching, variable-order Markov model, multiple simple Bayesian, simple Bayesian mixture and single-scope blending.

We have provided a few variants of algorithms that can be used to implement single-scope blending. The single-scope blending aims to maximize the generic space and run into the problem of subgraph isomorphism. The traditional subgraph isomorphism is solved by using backtracking algorithms, which is NP-complete and has a complexity of n! where n is the number of nodes in each situation, assuming same number of nodes. The best-first search and the attention based are algorithms that can potentially reduce the complexity from exponential to quadratic.

IV. EXPERIMENT 1: PYMUD—AN AGENT-BASED VIRTUAL ENVIRONMENT

A. INTRODUCTION

Modeling and Simulation (M&S) tools have been used widely in military training and analysis. The fidelity of the computational model is critical for any analysis that requires computational modeling to be meaningful. Many computational human models make decisions based on previous and current states alone. Kunde and Darken [3] show the fidelity of agent behaviors can be enhanced by prediction. This finding is in line with Kurby and Zacks [2] cognitive neuroscience studies that show human agents make decisions based on predicted future states. To enable decision making based on future states, the agent must have prediction capability.

Darken [8] developed situation learning to allow learning and prediction on a benchmark environment called Pymud, a text-based role-playing game. Darken [8] showed that the statistical lookup table and variable matching prediction techniques work very well over 200,000 percepts for this game. However, prediction accuracy is poor in the beginning of the relational time-series when there is a lack of situations learned of the environment. This shows that prediction is difficult in unknown environments, and the prediction accuracy is made worst when the environment is stochastic and noisy. On the chapter on previous work, the prediction accuracy significantly improves with Markov chain and Bayesian inference. However, these two methods cannot predict new percepts.

In this chapter, we compare the various forms of newly developed single scoping blending inspired algorithms with the other prediction techniques mentioned in chapter II on the Pymud role-playing game.

B. PYMUD

Pymud is a text-based role-playing virtual environment in which, a virtual agent is controlled by a human player. To create an unknown behavior, the virtual agent is programmed to choose his action randomly. Actions include “go eastward,” “pick up a weapon,” “equip with weapon,” “hit,” and many more. There are other agents (monsters) in the environment such as goblins, trolls and dragons. There are three types of weapons: pitchfork, dagger and sword. Each weapon varies in effectiveness against each type of monster. Each time a monster is killed, it will leave behind a weapon. Each monster, weapon, agent and location has a distinct name constant. The sequence of percepts describes what the agent sees, such as its location, weapons, and monsters. An example of a relational time-series from Pymud is listed in Figure 28. . The relational time-series is a sequence of percepts. Each percept has a time component, which is shown here as the first term. The second term is the simplified percept. The terms in the parentheses are the object constants. Note that the arity can be 1 or 2. The percept type is identified by the last argument in the percept such as ‘a’, ‘e’, ‘+’ and ‘–‘ where ‘a’ indicates that the percept is of action type, ‘e’ indicates that the percept is of event type, ‘+’ indicates the start of a new state and ‘–‘ indicates the end of a new state.

```
look(spock84, 0.0, a)
place(Paperville3, 0.0, +)
location(pitchfork74, Paperville3, 0.0, +)
pitchfork(pitchfork74, 0.0, +)
location(spock84, Paperville3, 0.0, +)
spock(spock84, 0.0, +)
get(pitchfork74, spock84, 2.75, a)
get(spock84, pitchfork74, 2.75, e)
location(pitchfork74, spock84, 2.75, +)
```

Figure 28. A Relational Time-series from Pymud.

C. UTILITY OF PREDICTION

There are many ways how the prediction can be used. The main one is to use the predictions to improve the behavior fidelity of the virtual human agent. For example, if we have a rule-based agent that has a single rule that says “if being hit=>run”. This rule says that if the agent is being hit, it should run away. If the agent encountered a red goblin and being hit, it will run away. The next time when the agent sees another red goblin, it will not run until being hit. Such a behavior is clearly suboptimal. If the agent is able to predict that the red goblin will hit it, it will run away the moment it realize that it is co-located with a red goblin.

Another motivation is to predict based on other agents encounter. If an agent observes that a red goblin hit another agent, when it sees another red goblin, it will run away. Other than penalty learning, it can also learn to predict positive rewarding states.

Reward and penalty details need not come from action percept. They can come from other percepts for state update such as a reduction in life or have a better weapon. All types of percepts can be important for decision making. Hence, we do not discriminate percepts but attempt to predict all percepts that arrive.

D. METHODOLOGY

Our prediction task is to predict the next percept the agent may see, given the past percept sequence, regardless of the type of percepts such as action or state update. The controls of the experiment are as follow.

1. Test Data

The test data were obtained from the Pymud text-based role-playing game. To create an unknown behavior, the virtual agent that is supposed to be controlled by a human player is programmed to choose his action randomly. Darken [8] tested the prediction performance by running the prediction algorithms on more than 250,000 percepts. The prediction accuracy gets increasingly better as the number of percepts increases. This is true because Pymud is a stationary environment and given sufficient number of percepts, the general characteristics of Pymud can be learned.

In this study, we want to know how the prediction techniques work in noisy and mostly new environments. We consider short relational time-series of 100 percepts only. To allow statistical significance testing, we use 40 different relational time-series of 100 percepts. To simulate noisy environments, before we make each prediction, we randomly select two simplified percepts in the current situation and exchange their position in the situation. For example, suppose we have percept A, B, C and D in a situation in this order: A>B>C>D. If the two random percepts selected are A and B, the situation becomes B>A>C>D.

2. One Vote

A given situation can have multiple target percepts (possible predictions). Different prediction techniques are capable of different number of prediction. For Bayesian inference, all previously encountered percepts are possible. For Markov inference, the lower the order, the greater number of prediction. Situation-matching prediction depends on the number of target percepts. Hence, to be fair, we asked each technique to produce their best guess.

3. Next Percept

A prediction may not occur immediately and another percept may arrive before the predicted percept arrives. There are several ways to consider prediction with a time interval. We can use a constant time interval and or time of occurrence between situation and target. For second method, we can collect the intervals compute mean and standard deviation (SD). If the predicted percept falls within 1SD, 2SD or 3SD, we can consider it correct. There are two ways of collecting the interval, from situation's perspective or target's perspective. As a result of all these complexity, we fall back on the strictest measure of effectiveness: predict the next one. Prediction with time is studied in the sensitive studies at chapter 0 where 2SD is used on target's perspective.

4. Prediction Accuracy

The predicted percept $\mathrm{p} = \mathrm{r}(\mathrm{c}_1, \mathrm{c}_2, \dots, \mathrm{c}_\mathrm{m})$ is said to be correct if the next percept $\mathrm{p}' = \mathrm{r}'(\mathrm{c}_1{}', \mathrm{c}_2{}', \dots, \mathrm{c}_\mathrm{m}{}')$ is such that $\mathrm{p}' = \mathrm{p}, \mathrm{r}' = \mathrm{r}, \mathrm{c}_\mathrm{i}' = \mathrm{c}_\mathrm{i}$ for all $\mathrm{i} = 1,2, \dots, \mathrm{m}$. Prediction accuracy = $\frac{c}{n}$ where n is the number of percept receive and c is the count of correct prediction

5. Hardware

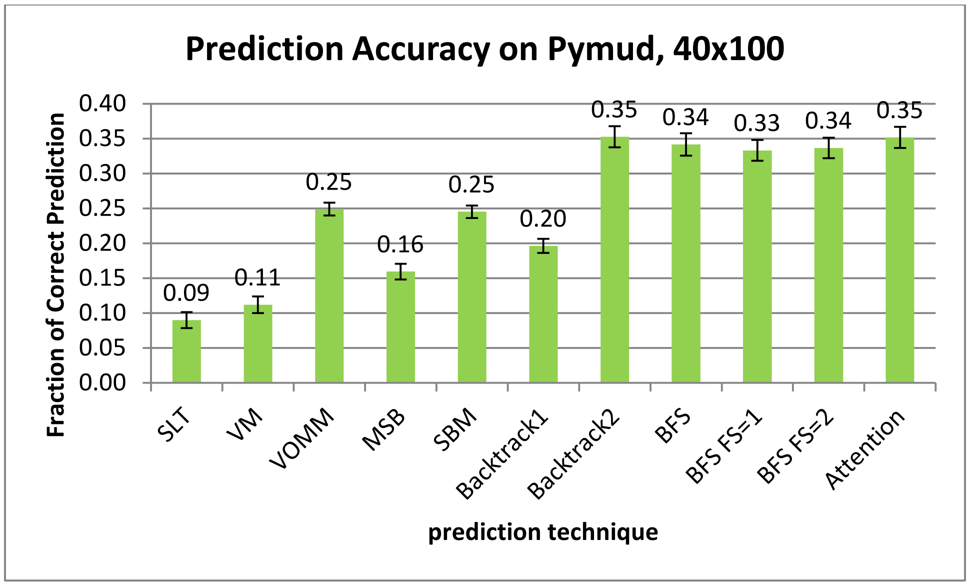
All experiments were run on a Dell XPS Laptop i7 1.87Ghz 16GB RAM with Windows 7.

6. Time Window Size

The time window used in the experiment is 0.1sec, the same time window used in the previous work experiment.

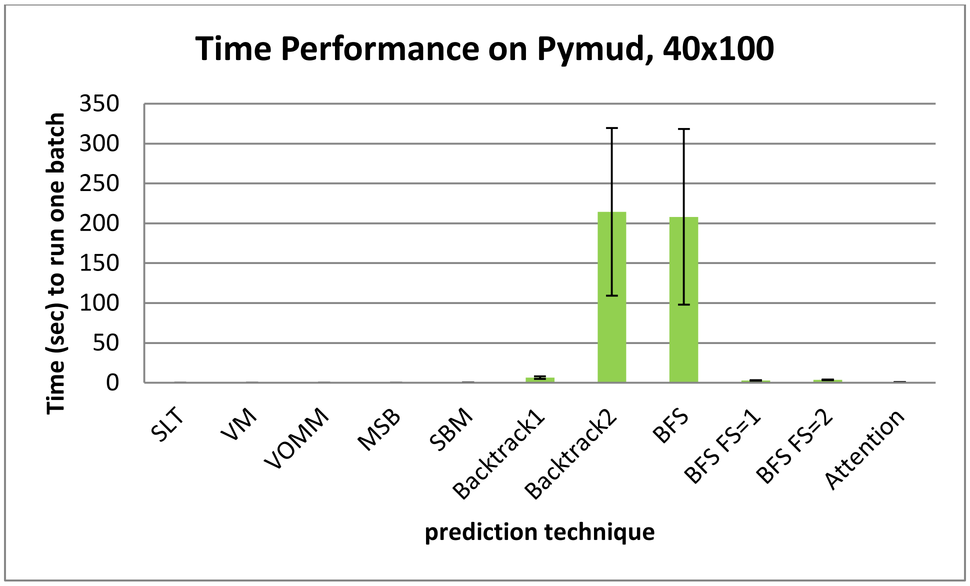
E. RESULTS

The prediction accuracies are shown in Figure 29. and the computation times are shown in Figure 30. . To allow better comparison of the computation times, the timings for the two slowest algorithms are removed in 0. In the results listed in the figure, Backtrack1 refers to the original backtrack algorithm listed in algorithm 2. Backtrack2 refers to a more complete backtrack algorithm listed in algorithm 4. BFS refers to algorithms 5. BFS FS=2 refers to algorithm 5 but with fringe size limited to 2. Attention refers to algorithm 7.



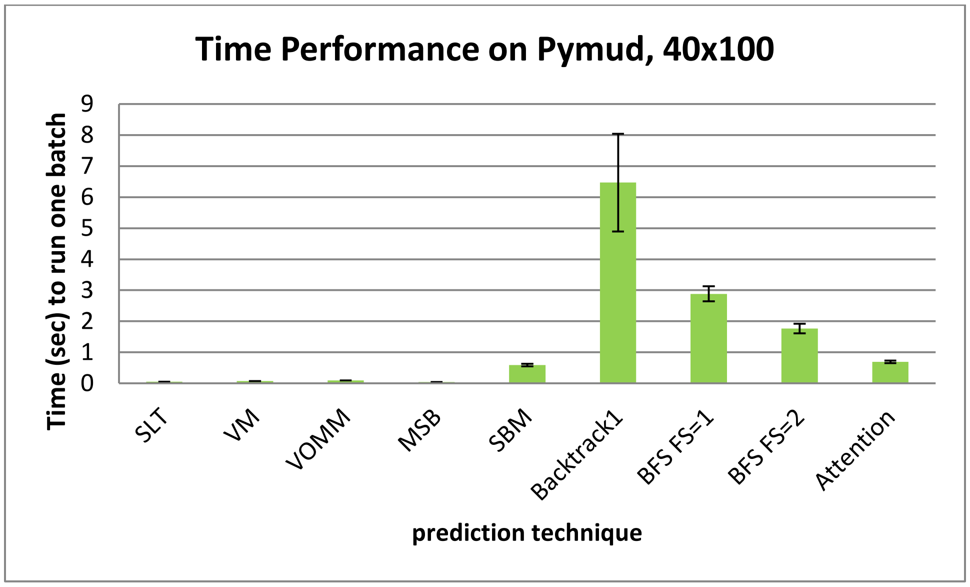
Legend: SLT: Statistical Lookup Table. VM: Variable Matching. VOMM: Variable-Order Markov Model. MSB: Multiple Simple Bayesian. SBM: Simple Bayesian Mixture. Backtrack1: Algorithm 2. BFS FS=x: Best-first Search with Fringe Size x

Figure 29. Comparison of Prediction Accuracy on Pymud.



Legend: SLT: Statistical Lookup Table. VM: Variable Matching. VOMM: Variable-Order Markov Model. MSB: Multiple Simple Bayesian. SBM: Simple Bayesian Mixture. Backtrack1: Algorithm 2. BFS FS=x: Best-first Search with Fringe Size x

Figure 30. Comparison of Computation Time on Pymud. SLT.



Legend: SLT: Statistical Lookup Table. VM: Variable Matching. VOMM: Variable-Order Markov Model. MSB: Multiple Simple Bayesian. SBM: Simple Bayesian Mixture. Backtrack1: Algorithm 2. BFS FS=x: Best-first Search with Fringe Size x

Figure 31. Comparison of Computation, with Bactrack2 and BFS removed.

F. DISCUSSION

Figure 29. shows that the worst of the single-scope blending algorithms perform better than the other previous techniques in prediction accuracy. Backtrack1 is a subgraph isomorphism approach that incrementally increases the size of common subgraph if and only if the newly added binding results in at least one common structure with the previously bind nodes. If the newly added binding are not connected to the previously bind nodes, it will not be added. It is only good for a connected graph and will ignore the unconnected nodes. It is incomplete for situation comparison in Pymud. As a result, it performs poorly as compared to the other single-scope blending techniques.

Backtracks2 performs better than Backtrack1 in prediction accuracy because it is a complete technique and searches the disconnected part of the graph. Backtracks2 has limitation. Firstly, the computation complexity is n! or $O(n^n)$ where n refers to the number of constant in one situation. Furthermore, in most cases where noisy situation are vastly different, there are many suboptimal bindings that will generate the same number of common percept. We need a more efficient algorithm and better algorithms.

The best-first search family of algorithms performs much better than the traditional backtracking algorithms in terms of prediction accuracy. The original best-first search algorithm described in algorithm 5 is complete. However, it has no computational time advantage because the heuristics are only useful when one situation is a complete subgraph of another situation. When the common percepts are only a subset of both situations, the termination condition will not be satisfied and the algorithm must process all necessary nodes in the fringe. Furthermore, the sorting process increases the computational complexity. To have a feel of how bad the sorting complexity is, Figure 32. shows the maximum fringe size encountered during the prediction events. Nevertheless, the fringe does indicate that the best solutions are found at the front of the queue. When the fringe size is reduced to 1 or 2, the computation time reduces significantly as shown in Figure 30. and 0. The statistical student-T test results for comparing the prediction accuracy of the Fringe size 1 and 2 with the original best-first search are given in Table 2. If our threshold p-value is 0.05 ($\alpha_{0.05}$), $BFS_{FS=1}$ fails the paired t-test significant test and we conclude that FS=1 is different than the original best-first search. However, the two sample t-test p-value shows that their overall prediction accuracies have no significant different. The FS=2 does passed both paired t-test and two-sample t-test. Hence, we can conclude that FS=2 is similar prediction performance to the original best-first search. The restricted best-first search is termed as Greedy best-first search.

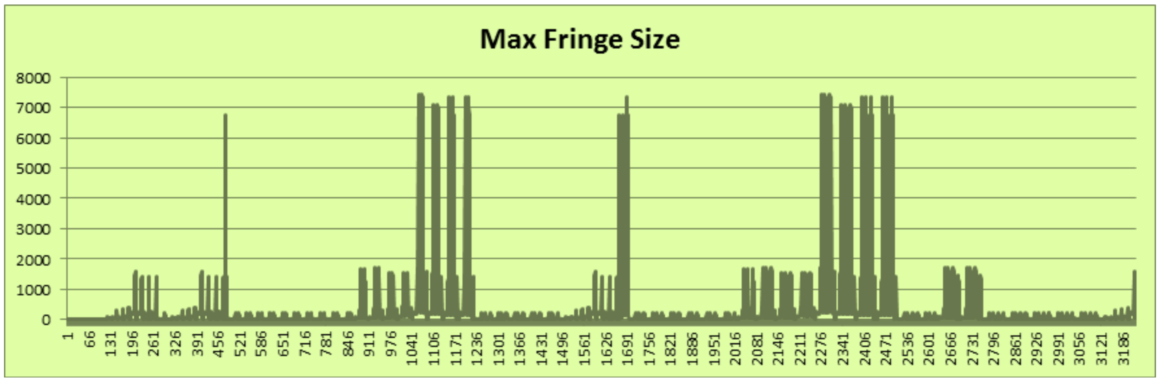


Figure 32. Maximum Fringe Size Encountered during the 40x100 Prediction Events.

	P-value BFS FS=1	P-value BFS FS=2
Paired t-test	0.018679441	0.149055981
Two sample t-test	0.698569455	0.819056942

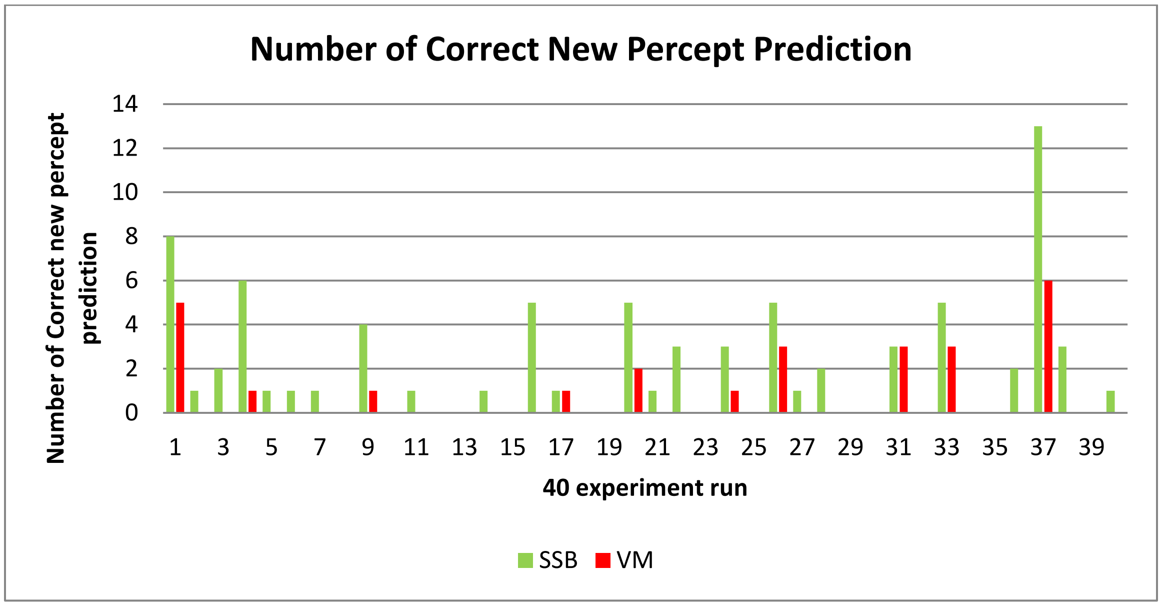
Table 2 Statistical Student T-Test Result. P-value for Comparing Original BFS with Fringe Size Restricted BFS on Computation Time.

The Attention algorithm performs similarly than the best-first search method in prediction accuracy and significantly better in computation time. The results of paired t-test and two sample group t-test are given in Table 3. The paired t-test shows that the best-first search and attention model are different but the two sample t-test indicates that their overall prediction accuracies are similar.

	P-value for comparing Best-first search and Attention
Paired t-test	0.008944283
Two sample t-test	0.652817524

Table 3 T-Test Result (p-value) for Comparing Original BFS with Attention Model.

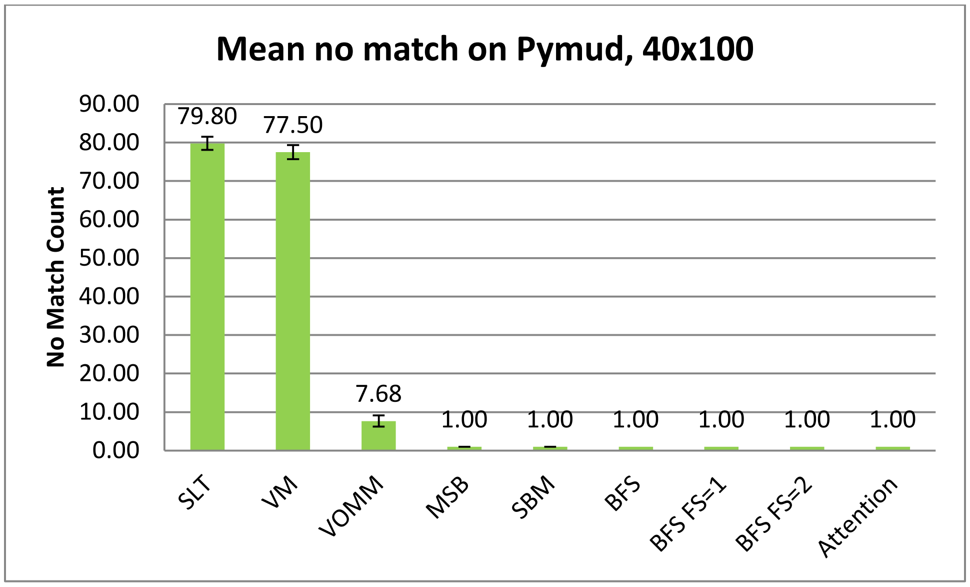
The single-scope blending and variable matching prediction techniques do not rely solely on recalling previously seen percepts. The bindings of constants between current and previous situations provide the capability to adapt previously-seen percepts to the current situation. Unlike the statistical lookup table or Bayesian and Markov chain techniques where predicted percepts are "activated" from a bunch of previously seen percepts, single-scope blending and variable matching generate predictions that may or may not have been seen before. Figure 33. shows the number of correct prediction generated by single-scope blending and variable matching that have not been seen before. The "generative" approach to prediction is the primary reason why single-scope blending performs much better than other techniques. Variable matching does not do equally well because it requires full graph isomorphism, which is hard to come by in a dynamic and noisy environment. Single-scope blending is superior to variable matching because subgraph isomorphism has the property of partial situation matching that allows the recall of similar situation in the face of noise. Bayesian and Variable-Order Markov Model also demonstrate partial matching. However, these techniques cannot associate similar constants and lack the ability to adapt recalled percepts to the current situation.



Legend: SSB: Single-scope Blending. VM: Variable Matching.

Figure 33. Number of Correct Prediction of New Percepts.

When the environment is often novel and noisy, the current situation can hardly match the learned situations in the memory. Figure 34. shows the mean number of no-match for 40 batches of 100 percepts. No-match occurs when the algorithm is unable to find a reasonable situation. Variable-Order Markov model handles the no-match problem by varying the order of Markov Model. The Bayesian techniques handle the problem using Laplace distribution.



Legend: SLT: Statistical Lookup Table. VM: Variable Matching. VOMM: Variable-Order Markov Model. MSB: Multiple Simple Bayesian. SBM: Simple Bayesian Mixture.

Figure 34. Comparisons of No-Match for Prediction Techniques in Conjunction with Situation Learning.

The variable-order Markov model handles novel situations better than statistical lookup table and variable matching. While the variable-order Markov model does not require exact percept to percept matching, and even allow partial matching, it requires exact sequential adjacency ordering. For example, the sequence of words [The blue fish is eating] will not match the sequence [The fish is eating]. In addition, variable-order Markov model treats each percept as a proposition.

The multiple simple Bayesian network is able to handle unseen situations with the smoothing n-gram models [59] of assigning probabilities to newly encountered percepts in the current situation. However, its performance is limited for several reasons. Firstly, it cannot predict unseen percept. Secondly, when there are too many novel percepts, the prior probability for each percept can be very low. The smoothing n-gram models assign a probability that can be unfairly large to new percepts. Thirdly, Simple Bayesian network cannot handle exclusive-OR relationship and there are percepts that are mutually exclusive. Thirdly, percepts in the sequence are not independent and identically distributed. The simple Bayesian mixture performs better than the multiple simple Bayesian. However, it also suffers some of the limitations found in multiple simple Bayesian.

G. CONCLUSION

We have implemented the situational learning method and prediction techniques on Python programming language and quantitatively compared the results on a role-playing game. We conclude that the Bayesian and Markov approaches perform better than the situation matching approaches. One surprising finding in this study is that non Markovian techniques can perform equally well than the Markovian one, even though the Markovian techniques are the popular techniques for sequence learning and prediction.

The results presented above show that the single-scope blending approach to prediction on relational time-series perform much better than the current techniques in the role-playing game benchmark environment. In addition, we have also showed that the two novel algorithms: Greedy best-first search and attention perform significantly better than the original backtracking approach for solving our subgraph isomorphism problem.

THIS PAGE INTENTIONALLY LEFT BLANK

V. EXPERIMENT 2: INTRUSION-ALERT PREDICTION

A. INTRODUCTION

A network intrusion-detection system [4] such as [5] is a critical device that screens all incoming packet for suspicious activities, either based on hand crafted signature rules, or abnormality detection. The network intrusion-detection system generates alerts in a time-series fashion.

There are two well-known issues with using a network intrusion-detection system. The first issue is that each network intrusion-detection system can generate tremendous amount of alerts. For example, Prof Rowe's Honeypot can potentially generate thousands of alerts per day. Such huge number of alert causes challenges for system administrators. Many alerts have lower priority. However, these alerts cannot be ignored because they may serve as prerequisite for higher priority alerts in the future. Alert predictions could justify ignoring earlier alerts. A second issue is that network intrusion-detection systems are retrospective defense in which alerts indicate current or previous attacks. When high priority alerts are generated, damages might have been done already.

B. INTRUSION-ALERT PREDICTION

There are several approaches that can help network administrators to deduce higher level information such as intents of attacker, by summarizing network intrusion-detection system alerts. One of the goals of higher level knowledge is to predict future attacker's action, which can be observed as future network intrusion-detection system alerts. The process of summarizing network intrusion-detection system alerts involve normalization [60], aggregation [61][62], and correlation. We will review some of the correlations techniques here since it is the part where prediction of future alerts can be made. Techniques of alert correlation include instant base, rule base, statistical, and temporal.

1. Instant

Instant base approaches [63][64][65] correlate alerts by matching the alerts against a library of known scenarios. Each scenario is described by a formal model such as attack graph, hierarchical tree, etc. Scenario development requires human expert to meticulously analyze each attack scenario, and to represent it in the format that the system understand. The limitations of Instant Base approaches are that, scenario development is expensive and that the library of historical scenarios cannot detect new kinds of attack [66]. An example of correlation graph is as shown in Figure 35. .

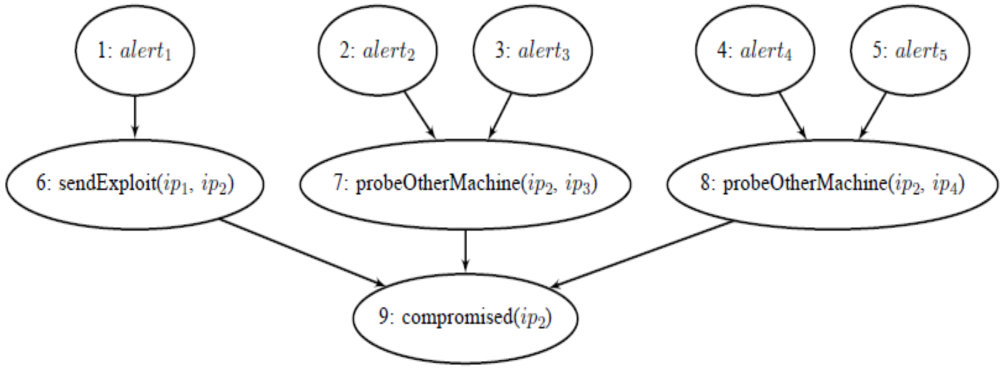

1: $alert_1$
2: $alert_2$
3: $alert_3$
4: $alert_4$
5: $alert_5$
6: sendExploit(ip_1, ip_2)
7: probeOtherMachine(ip_2, ip_3)
8: probeOtherMachine(ip_2, ip_4)
9: compromised(ip_2)

Figure 35. Correlation Graph from [67].

2. Rule Base

Rule-based approaches [64][67][68] address some limitations of instant base approaches by using rules. Sundaramurthy et. al. [67] shows that by using generic rules, they are able to identify attack scenarios that were not considered during the development phase. The rules usually directed towards the pre and post condition of each pair of alerts. Two alerts are correlated if one alert generate a post condition that matches the pre-condition of the second alert. In Sundaramurthy et. al. [67]'s approach, facts from the previous step will be added into prolog, which will generate and infers more facts. All facts (added or inferred) will be put together to form a graph, called scenario. As with all knowledge based system, Rule based approaches suffer the limitations of rule management, as well as the need for human expert to convert domain knowledge into a set of rules.

3. Statistical

Both Instant and Rule base suffer because they rely on human expert for knowledge. Statistical techniques allow automated mining of knowledge. Qin [69] proposed a Bayesian Network approach that runs offline on a set of historical records to learn the network structures and the conditional probability tables. Each alert pairs are analyzed based on their attributes to form different evidences. These evidences are used to compute the probability of correlation (Figure 36.). The limitations of statistical techniques offline batch training on selected training set. It assumes that the problem of NIDS prediction is stationary and the selected training sets are representative of what the network will be expecting.

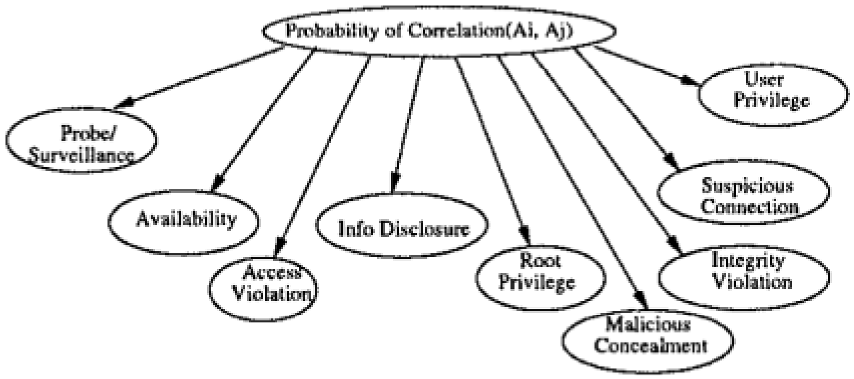


Figure 36. Probabilistic Reasoning Model from [69].

4. Temporal

Cuppens [64] proposed a time series-based statistical analysis method that aims to test if a time-series variable X correlates with another time-series variable Y. However, the approach suffers a limitation of failure to correlate two alerts if there are random time delays between them. Li et. al. [11] proposed a sequential approach. During the offline training, the algorithm divides the entire list of processed alerts into multiple shorter sequences by using a sliding window. The sequences are then fused to form a minimal set of sequence that best represent the set of sequences. The author writes that the detection performance decreases in the face of new attack strategies. The sequence diagram generated is shown in Figure 37. . The limitation of Li et. al. [11]'s approach is the need for offline batch training.

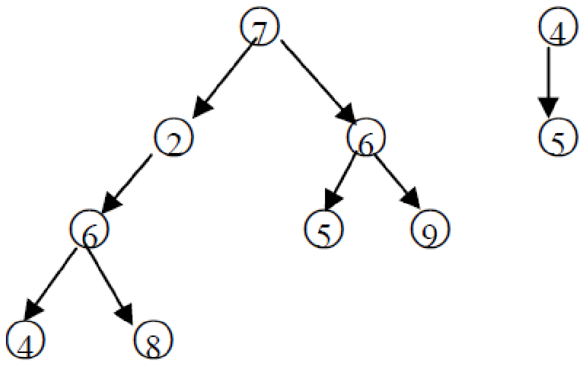


Figure 37. Sequence Diagram from [11]. Each Node Represents an Event. Each Edge Represents a Transition of Event.

5. Discussion

One important purpose of correlation is the ability to predict future action of the attackers amid noisy and incoming alert sequence. One major limitation of approaches above is the failure to learn the knowledge online. Since attack strategy changes in a fast pace fashion [11], the current alert must be incorporated into the knowledgebase as soon it is received. Darken‘s situation learning [8] is capable of assimilating the latest percept into the knowledgebase, and use that updated knowledge to make a prediction. Our situation learning and prediction approach can be used in network intrusion-detection system Snort alerts prediction since the network intrusion-detection system alert sequence is a time-series and that each network intrusion-detection system alert is in relational form since each alert has a predicate (Alert ID) and that attributes (protocol, IP addresses, etc.) in the alert. Our Bayesian approach is similar to Qin [69] but is able of online, learning. Our variable Markov model is similar to Li et. al. [11] but is capable of online learning and partial matching.

C. UTILITY OF PREDICTION

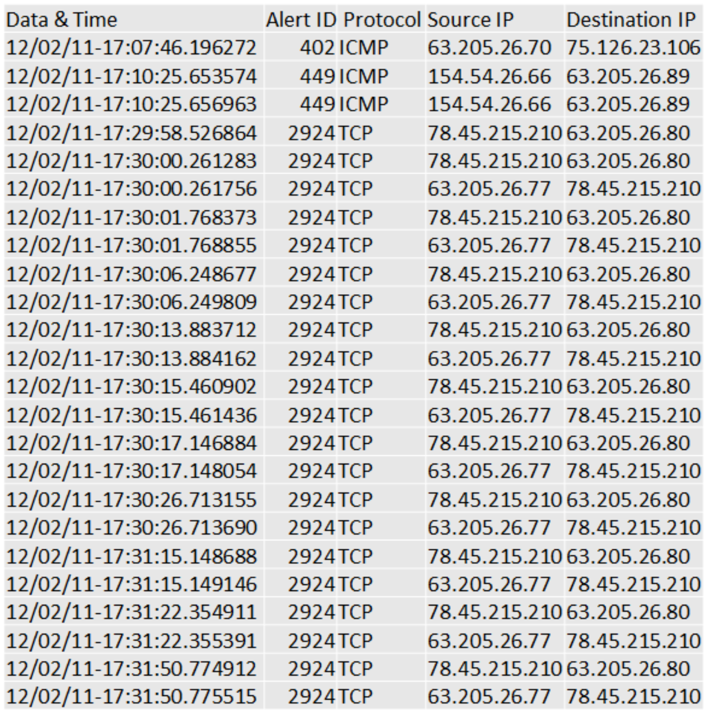
Alert prediction is useful for network defense. The first motivation is to provide the network administrators early warning of a potentially harmful attack so that they can take appropriate action to avoid the attack. The second motivation is for cross networks alert prediction. Situation learning learns a set of situation that can be shared with other situation learning systems. If an attack is encountered in one network, we can immediately use that data to monitor and predict similar attack on all other networks. The third motivation is to allow anticipatory approach to cyber deception. Our predictions may tell what the hacker wants so that we can deploy our deceptions to achieve our objectives such as luring him away from the production networks.

D. METHODOLOGY

Our prediction task is to predict the next alert the system administrator may see, given the past alert sequence. The controls of the experiment are as follow.

1. Test Data

Professor Rowe has a honeynet setup in his lab with a Snort network intrusion-detection system. Two alert sequences (labeled as dataset 1 and 2) were collected from the honeynet for this research, described in [70]. The alert sequences were obtained from Internet traffic trying to connect to the honeynet. An example of the alert sequence is listed in Figure 38. . A summary of the dataset is given in Table 4. The total column is the total number of alerts. If two alerts are exactly the same but arrived at different time, they are said to be repeated. If two alerts are different even through the Snort ID are the same, both are considered as different distinct alerts. The distinct alert column refers to the number of the distinct alerts encountered, excluding the repeated alert. Repeat rate in Table 4 is computed as the division of the total number of alert by the number of distinct alert. It is a measure of number of high frequency alerts. The Entropy is a measure of uncertainty of a random variables defined in Shannon [71]. In our context, the random variable is the occurrence of alerts. Entropy is computed as: E = $-\sum_{i}^{n} p(x_i) log_2(p(x_i))$ where $p(x_i)$ is the probability of alert x_i.



Data & Time	Alert ID	Protocol	Source IP	Destination IP
12/02/11-17:07:46.196272	402	ICMP	63.205.26.70	75.126.23.106
12/02/11-17:10:25.653574	449	ICMP	154.54.26.66	63.205.26.89
12/02/11-17:10:25.656963	449	ICMP	154.54.26.66	63.205.26.89
12/02/11-17:29:58.526864	2924	TCP	78.45.215.210	63.205.26.80
12/02/11-17:30:00.261283	2924	TCP	78.45.215.210	63.205.26.80
12/02/11-17:30:00.261756	2924	TCP	63.205.26.77	78.45.215.210
12/02/11-17:30:01.768373	2924	TCP	78.45.215.210	63.205.26.80
12/02/11-17:30:01.768855	2924	TCP	63.205.26.77	78.45.215.210
12/02/11-17:30:06.248677	2924	TCP	78.45.215.210	63.205.26.80
12/02/11-17:30:06.249809	2924	TCP	63.205.26.77	78.45.215.210
12/02/11-17:30:13.883712	2924	TCP	78.45.215.210	63.205.26.80
12/02/11-17:30:13.884162	2924	TCP	63.205.26.77	78.45.215.210
12/02/11-17:30:15.460902	2924	TCP	78.45.215.210	63.205.26.80
12/02/11-17:30:15.461436	2924	TCP	63.205.26.77	78.45.215.210
12/02/11-17:30:17.146884	2924	TCP	78.45.215.210	63.205.26.80
12/02/11-17:30:17.148054	2924	TCP	63.205.26.77	78.45.215.210
12/02/11-17:30:26.713155	2924	TCP	78.45.215.210	63.205.26.80
12/02/11-17:30:26.713690	2924	TCP	63.205.26.77	78.45.215.210
12/02/11-17:31:15.148688	2924	TCP	78.45.215.210	63.205.26.80
12/02/11-17:31:15.149146	2924	TCP	63.205.26.77	78.45.215.210
12/02/11-17:31:22.354911	2924	TCP	78.45.215.210	63.205.26.80
12/02/11-17:31:22.355391	2924	TCP	63.205.26.77	78.45.215.210
12/02/11-17:31:50.774912	2924	TCP	78.45.215.210	63.205.26.80
12/02/11-17:31:50.775515	2924	TCP	63.205.26.77	78.45.215.210

Figure 38. Alert Sequence from Snort Network Intrusion-detection System.

	Total	Distinct alert	repetitive rate	Entropy	Duration
DataSet1	6482	1590	4.08	7.46	2 months
DataSet2	9619	4304	2.23	11.08	2 weeks

Table 4 Properties of Snort Alert Dataset.

The charts given in Figure 39. plot the cumulative distribution function for distinct alerts in their respective datasets. In dataset 1, 40% of the distinct alerts make up 80% of the total number of alert. In dataset 2, 55% of the distinct alerts make up 80% of the total number of alert. The repetitive rate and entropy are important characteristics of the dataset because both represent the number of distinct alerts in a batch and describe the variability of the dataset. As the number of distinct alert in each batch increases, the entropy increases and repetitive rate drops. When entropy is low, meaning the dataset is highly repetitive and involves a small quantity of distinct alert, prediction techniques are expected to have better prediction accuracy. On the other hand, when entropy is high, prediction accuracy is expected to drop.

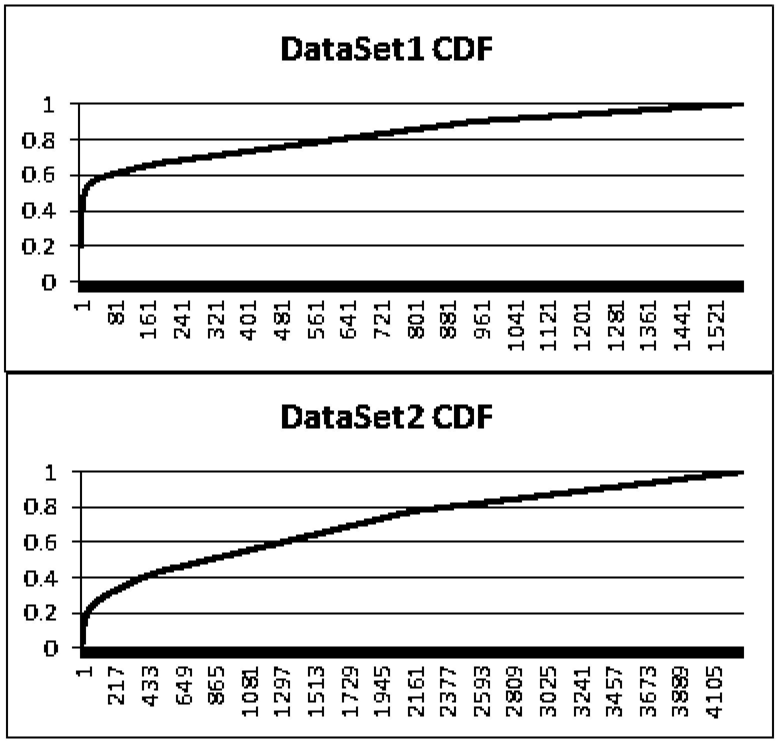


Figure 39. Cumulative Frequency of Distinct Alert in 2 Dataset. X-axis is the Cumulative Count of Distinct Alert Records Order by Decreasing Frequency of Those Records. Y-axis is the Cumulate Distribution Function.

2. Percept of Arity 2

A Snort alert is a percept. An alert sequence is a percept sequence. Snort alerts come in relational table form with many fields. Some of the fields contain irrelevant data for prediction. In this experiment, we use the fields: Snort ID, protocol and source and destination IP Address. The relational representation of an alert is "Snort ID (protocol, source IP, destination IP)". The arity is 3.

The single-scope blending algorithms are designed to work on percepts of arity 1 and 2, because of the underlying graph based representation. This means that each percept must have a relation and one or two object constants. Therefore, we must transform one 3-arity alert percept into an intermediate representation, which is a group of multiple arity-2 percepts. We must also account for multiple 3-arity alerts that form a situation. In a situation, we assign a record number to each alert by the order in the situation. The first alert is record 1, the second alert is record 2, and so on. In a percept, when a record number is assigned, we relate the field elements in the alert to that record number using the following method: $R_n(C_1, C_2, \ldots, C_q) = F_1 (Rn, C_1), F_2 (Rn, C_2), \ldots F_q (R_n, C_q), = F_i (R_j, C_i)$, If a record has four fields, we convert to arity 2 by saying that field i of record n is C, where i is the field or column number, n is the record number and C is the constant. An example of how the table form is converted to the intermediate form is illustrated in Table 5 and Table 6. Note that we also add arity-1 percept to mean that it is a type percept.

	Field 0:	**Field 1:**	**Field 2:**	**Field 3:**
Record	**ID**	**Protocol**	**IP**	**IP**
1	2924	TCP	63.205.26.77	78.45.215.210
2	2924	TCP	78.45.215.210	63.205.26.80

Table 5 Example of Two Snort Alerts in Relational Table Form.

Relation	**Constant 1**	**Constant 2**
Field0	Record0	2924
Predicate-Type	2924	
Field1	Record0	TCP
Protocol-Type	TCP	
Field2	Record0	63.205.26.77
IPAddress-Type	63.205.26.77	
Field3	Record0	78.45.215.210
IPAddress-Type	78.45.215.210	
Record-Type	Record0	
Field0	Record1	2924
Field1	Record1	TCP
Field2	Record1	78.45.215.210
Field3	Record1	63.205.26.80
IPAddress-Type	63.205.26.80	
Record-Type	Record1	

Table 6 Example of Two Snort Alerts in Relational Percept Form with Arty 1 and 2.

3. Time Window Size

The time window used in the experiment is 0.1sec.

4. One Vote

A given situation can have multiple target percepts (possible predictions). Different prediction techniques are capable of different number of prediction. For Bayesian, all previously encountered percepts are possible. For Markov, the lower the order, the greater number of prediction. Situation matching prediction depends on the number of target percepts. Hence, to be fair, we ask each technique to produce their best guess.

5. Next Percept

A prediction may not occur immediately and another percept may arrive before the predicted percept arrives. There are several ways to consider prediction with a time interval. We can use a constant time interval and or time of occurrence between situation and target. For second method, we can collect the intervals compute mean and standard deviation (SD). If the predicted percept falls within 1SD, 2SD or 3SD, we can consider it correct. There are two ways of collecting the interval, from situation's perspective or target's perspective. As a result of all these complexity, we fall back on the strictest measure of effectiveness: predict the next one. Prediction with time is studied in the sensitive studies at chapter 0 where 2SD is used on target's perspective.

6. Prediction Accuracy

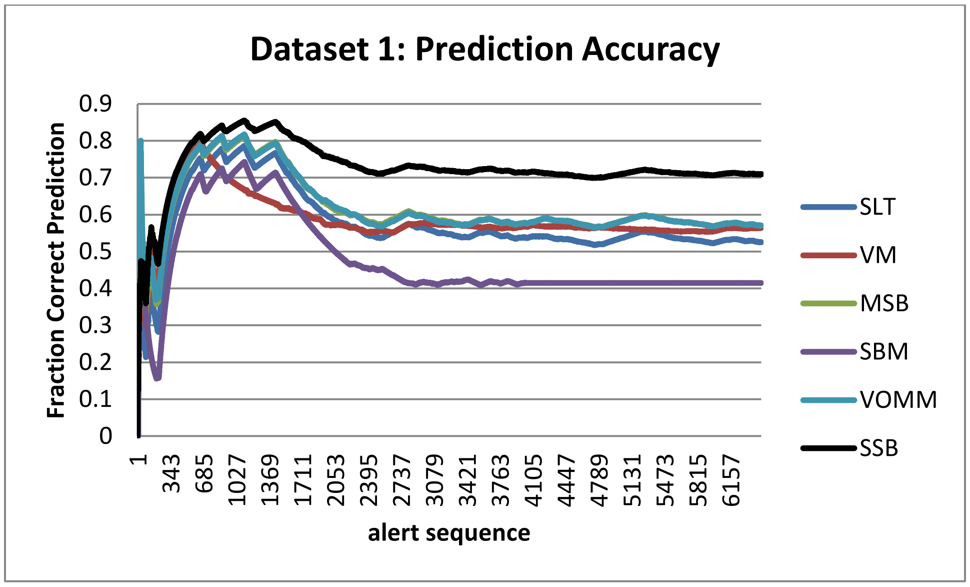
The predicted percept $\mathrm{p} = \mathrm{r}(\mathrm{c}_1, \mathrm{c}_2, \dots, \mathrm{c}_\mathrm{m})$ is said to be correct if the next percept $\mathrm{p}' = \mathrm{r}'(\mathrm{c}_1{}', \mathrm{c}_2{}', \dots, \mathrm{c}_\mathrm{m}{}')$ is such that $\mathrm{p}' = \mathrm{p}, \mathrm{r}' = \mathrm{r}, \mathrm{c}_\mathrm{i}' = \mathrm{c}_\mathrm{i}$ for all $\mathrm{i} = 1,2, \dots, \mathrm{m}$. Prediction accuracy = $\frac{c}{n}$ where n is the number of percept receive and c is the count of correct prediction

7. Hardware

All experiments were run on a Dell XPS Laptop i7 1.87Ghz 16GB RAM with Windows 7.

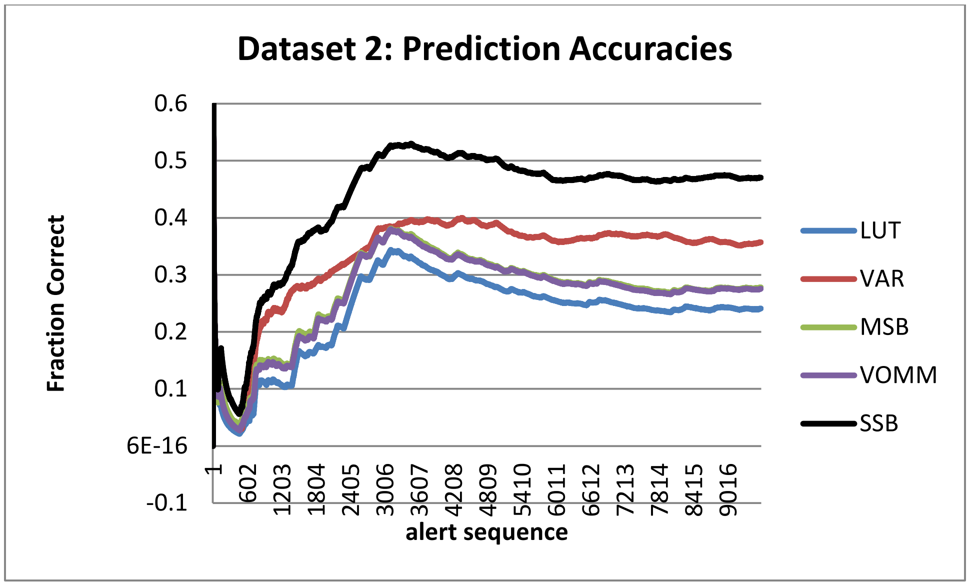
E. RESULTS

The prediction accuracies of the prediction techniques are illustrated in Figure 40. and Figure 41. for dataset1 and dataset2, respectively. The single-scope blending technique used here is the attention technique. Separate runs were conducted to compare between best-first search and attention but no significant differences were found. The final prediction of dataset 1 and 2 are repeated in Figure 42. for easy comparison.



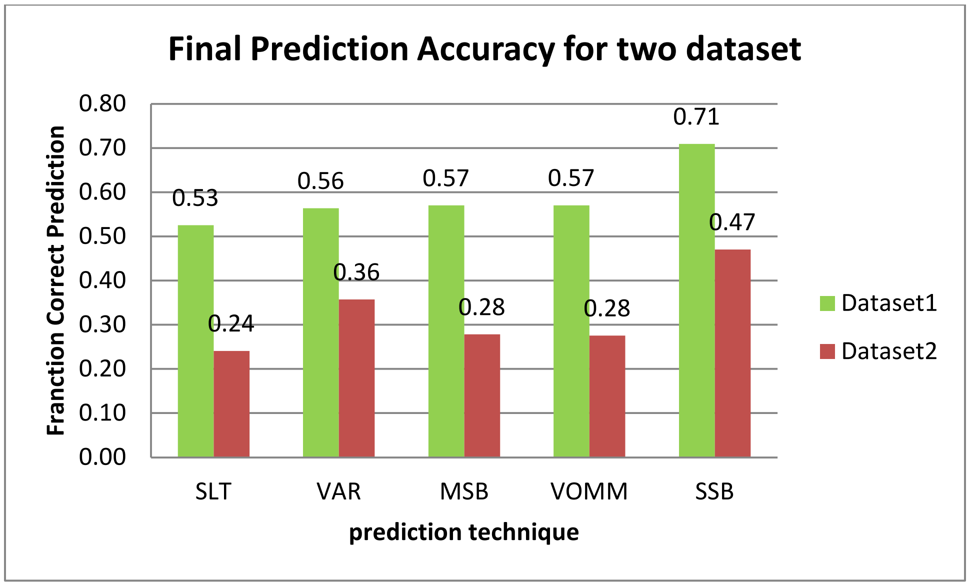
Legend: Statistical Lookup Table (SLT), Variable Matching (VM), Multiple Simple Bayesian (MSB), SimpleBayesian Mixture (SBM), Variable-Order Markov Model (VOMM), Single-scope Blending (SSB.)

Figure 40. Dataset 1: Prediction Accuracy.



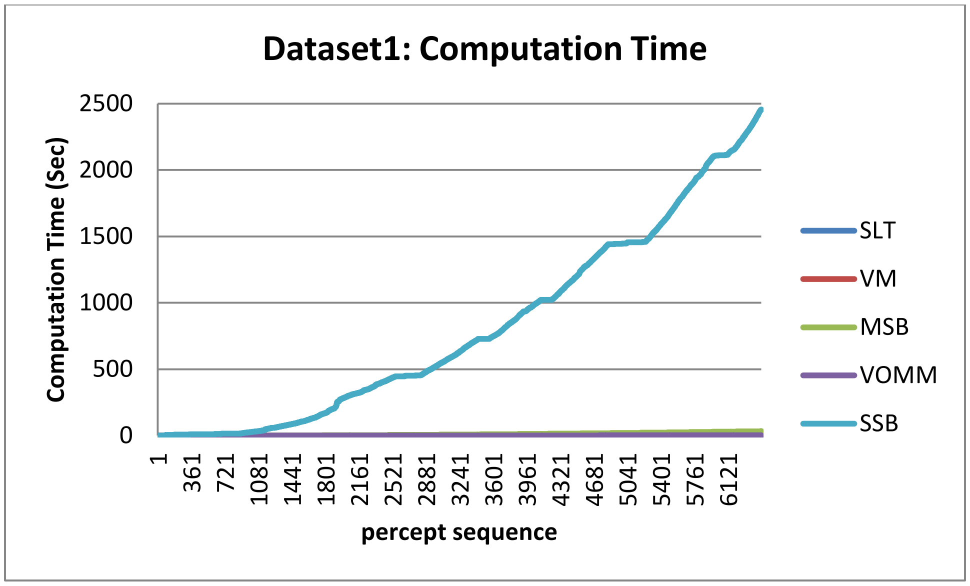
Legend: Statistical Lookup Table (SLT), Variable Matching (VM), Multiple Simple Bayesian (MSB), Simple Bayesian Mixture (SBM), Variable-Order Markov Model (VOMM), Single-scope Blending (SSB.)

Figure 41. Dataset 2: Prediction Accuracy.



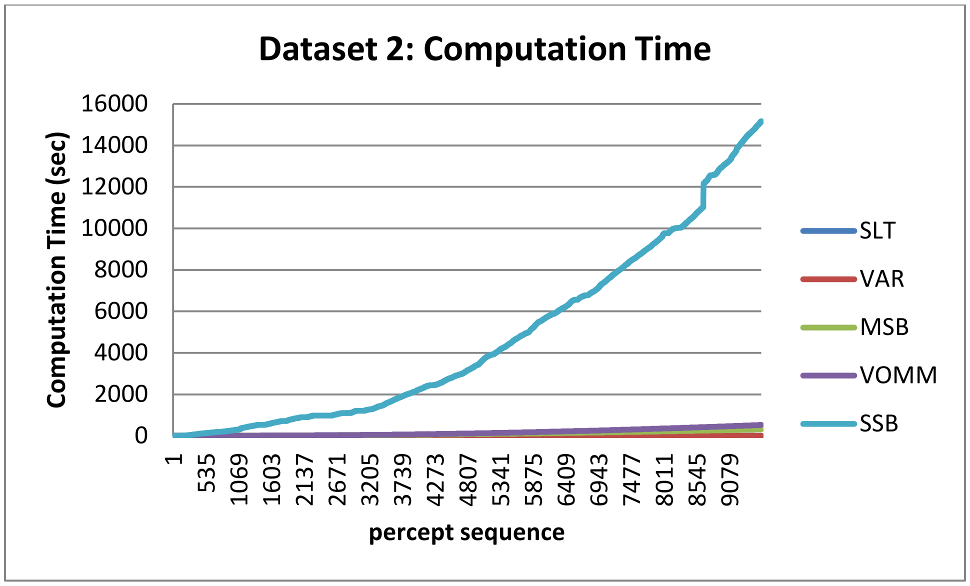
Legend: Statistical Lookup Table (SLT), Variable Matching (VM), Multiple Simple Bayesian (MSB), Simple Bayesian Mixture (SBM), Variable-Order Markov Model (VOMM), Single-scope Blending (SSB.)

Figure 42. Final Prediction Accuracies for Dataset 1 and 2.



Legend: Statistical Lookup Table (SLT), Variable Matching (VM), Multiple Simple Bayesian (MSB), Simple Bayesian Mixture (SBM), Variable-Order Markov Model (VOMM), Single-scope Blending (SSB.)

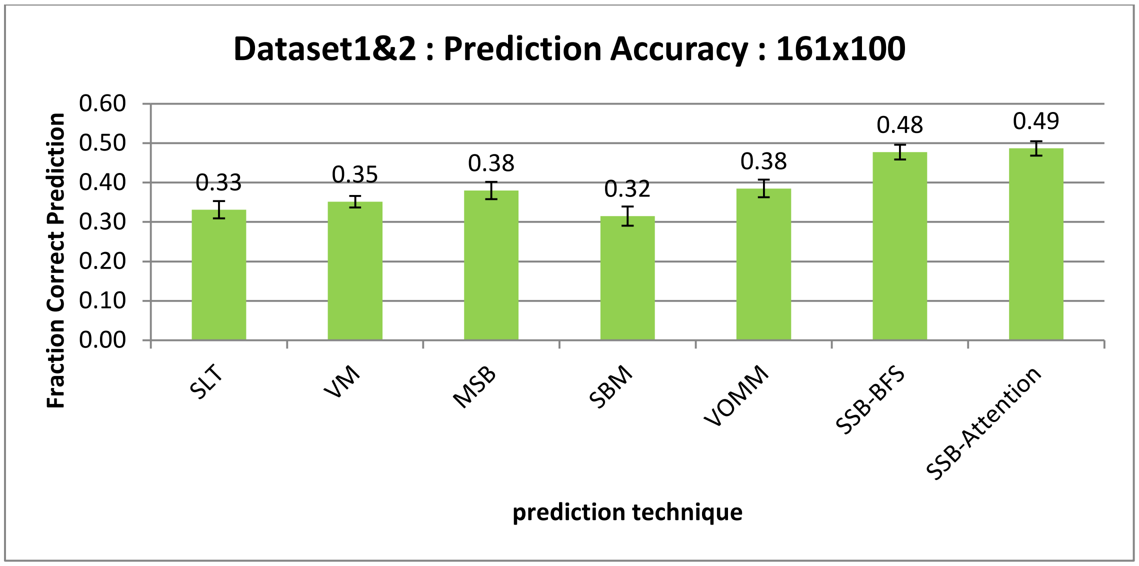
Figure 43. Dataset 1: Computation Time.



Legend: Statistical Lookup Table (SLT), Variable Matching (VAR), Multiple Simple Bayesian (MSB), Simple Bayesian Mixture (SBM), Variable-Order Markov Model (VOMM), Single-scope blending (SSB

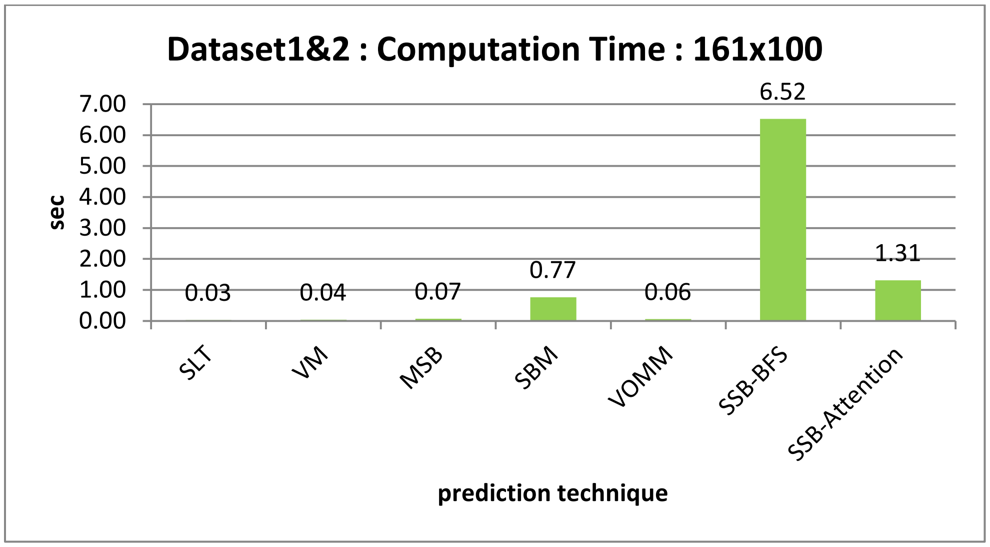
Figure 44. Dataset 2: Computation Time.

To allow us to do statistical tests, both datasets were combined into one and subsequently divided into 161 sequences of 100 alerts in each sequence. The prediction and computation time results are given in Figure 45. and Figure 46. , respectively. The paired t-test and two group t-test for prediction accuracies and computation as compared to the SSB-attention are given in Table 7 and Table 8, respectively.



Legend: Statistical Lookup Table (SLT), Variable Matching (VM), Multiple Simple Bayesian (MSB), Simple Bayesian Mixture (SBM), Variable-Order Markov Model (VOMM), Single-scope blending (SSB).

Figure 45. Dataset 1 and 2: Prediction Accuracies from 161 batches of 100 alerts.



Legend:Statistical Lookup Table (SLT), Variable Matching (VM), Multiple Simple Bayesian (MSB), Simple Bayesian Mixture (SBM), Variable-Order Markov Model (VOMM), Single-scope blending (SSB).

Figure 46. Dataset 1 and 2: Computation Time from 161 batches of 100 alerts.

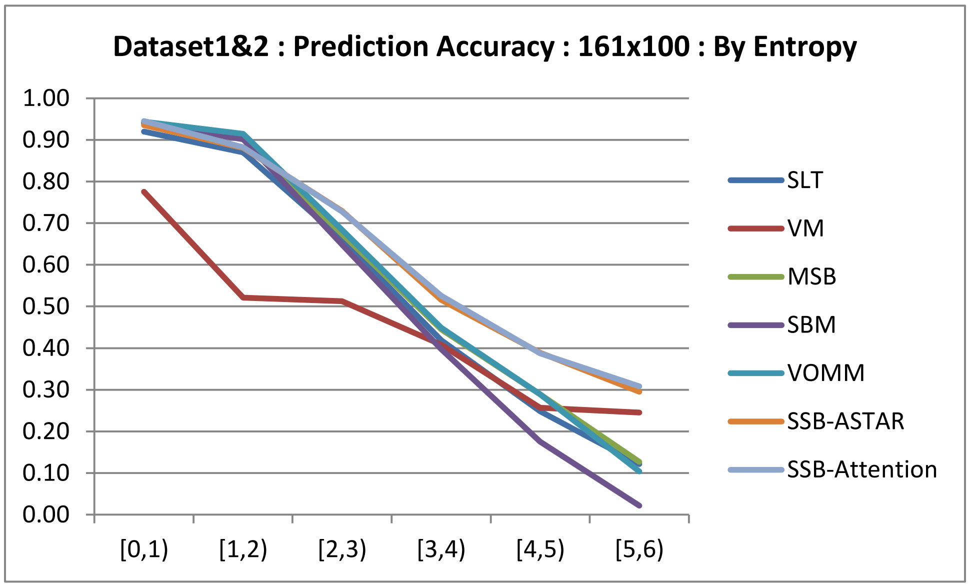
Test	SLT	VM	MSB	SBM	VOMM	SSB-BFS	SSB-attention
Paired T-test	0.000	0.000	0.000	0.000	0.000	0.000	1.000
Group T-test	0.000	0.000	0.000	0.000	0.000	0.711	1.000

Table 7 Statistical Significant Test with SSB-attention on Prediction Accuracies on Dataset1&2 161x100.

Test	SLT	VM	MSB	SBM	VOMM	SSB-BFS	SSB-attention
Paired T-test	0.000	0.000	0.000	0.000	0.000	0.002	1.000
Group T-test	0.000	0.000	0.000	0.000	0.000	0.003	1.000

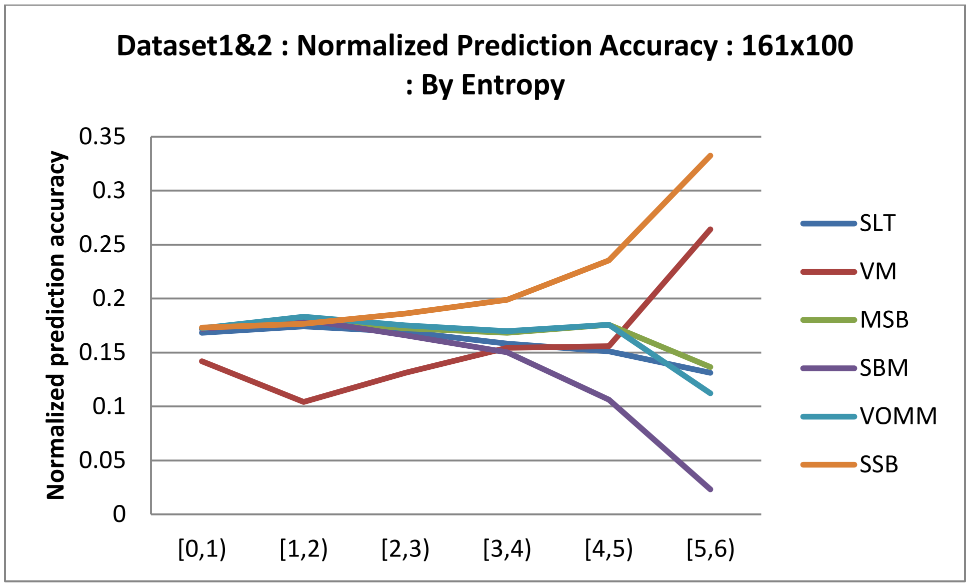
Table 8 Statistical Significant Test with SSB-attention on Computation Time on Dataset1&2 161x100.

We have also measured entropy for the run results. The prediction accuracies and computation time by entropy are as shown in Figure 47. , Figure 48. , and Figure 49. , respectively. The result of paired t-test and two-group t-test are given in 0 and 0, respectively.



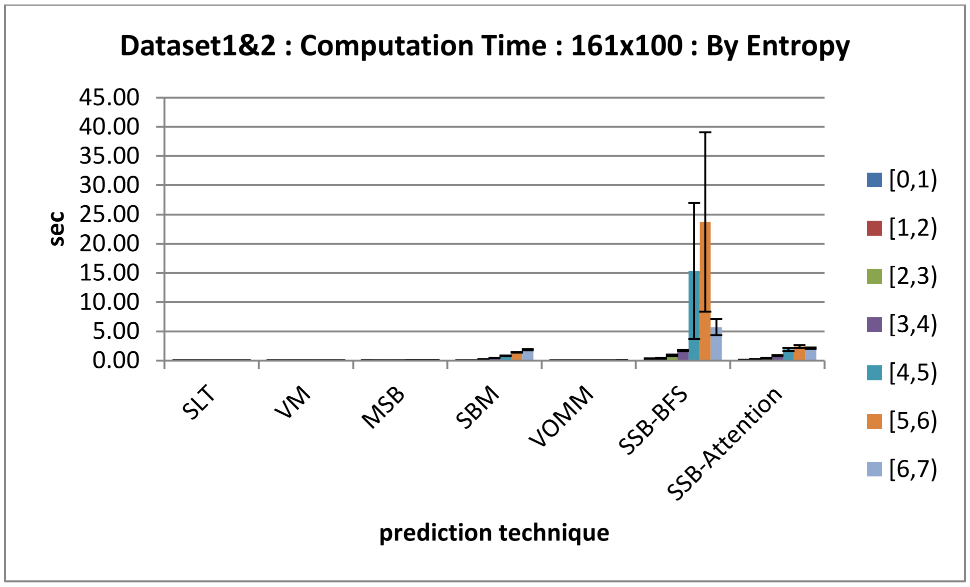
Legend: Statistical Lookup Table (SLT), Variable Matching (VM), Multiple Simple Bayesian (MSB), Simple Bayesian Mixture (SBM), Variable-Order Markov Model (VOMM), Single-scope Blending (SSB).

Figure 47. Dataset 1 and 2: Prediction Accuracies from 161 batches of 100 alerts.



Legend: Statistical Lookup Table (SLT), Variable Matching (VM), Multiple Simple Bayesian (MSB), Simple Bayesian Mixture (SBM), Variable-Order Markov Model (VOMM), Single-scope Blending (SSB).

Figure 48. Dataset 1 and 2: Normalized Prediction Accuracy over Entropy



Legend: Statistical Lookup Table (SLT), Variable Matching (VM), Multiple Simple Bayesian (MSB), Simple Bayesian Mixture (SBM), Variable-Order Markov Model (VOMM), Single-scope Blending (SSB).

Figure 49. Dataset 1 and 2: Computation Time from 161 batches of 100 alerts.

entropy	SLT	VM	MSB	SBM	VOMM	SSB-BFS	SSB-attention
[0,1)	0.293	0.278	0.188	0.187	0.245	0.144	1.000
[1,2)	0.029	0.035	0.022	0.018	0.030	0.028	1.000
[2,3)	0.000	0.000	0.000	0.001	0.000	0.000	1.000
[3,4)	0.000	0.000	0.000	0.000	0.000	0.000	1.000
[4,5)	0.000	0.000	0.000	0.000	0.000	0.123	1.000
[5,6)	0.000	0.000	0.000	0.000	0.000	0.021	1.000
[6,7)	0.000	0.000	0.000	0.077	0.000	0.014	1.000

Legend: Statistical Lookup Table (SLT), Variable Matching (VM), Multiple Simple Bayesian (MSB), Simple Bayesian Mixture (SBM), Variable-Order Markov Model (VOMM), Single-scope Blending (SSB).

Table 9 Paired T-test on Prediction Accuracies on Dataset1&2 161x100 by Entropy. Colored values represent significant difference compared with SSB-attention.

entropy	SLT	VM	MSB	SBM	VOMM	SSB-BFS	SSB-attention
[0,1)	0.279	0.291	0.224	0.298	0.243	0.417	1.000
[1,2)	0.019	0.024	0.016	0.040	0.022	0.144	1.000
[2,3)	0.000	0.000	0.000	0.003	0.000	0.002	1.000
[3,4)	0.000	0.000	0.000	0.000	0.000	0.000	1.000
[4,5)	0.000	0.000	0.000	0.000	0.000	0.133	1.000
[5,6)	0.000	0.000	0.000	0.000	0.000	0.022	1.000
[6,7)	0.000	0.000	0.000	0.035	0.000	0.019	1.000

Legend: Statistical Lookup Table (SLT), Variable Matching (VM), Multiple Simple Bayesian (MSB), Simple Bayesian Mixture (SBM), Variable-Order Markov Model (VOMM), Single-scope Blending (SSB).

Table 10 Group T-test on Prediction Accuracies on Dataset1&2 161x100 by Entropy. Colored values represent significant difference compared with SSB-attention.

F. DISCUSSION

From the results above (Figure 40. , Figure 41. and Figure 42.), the single-scope blending approach clearly outperformed the other techniques in prediction accuracy. In Figure 40. , most of the prediction accuracies reach steady state at around 0.57 while single-scope blending stabilizes at 0.71. The difference is about 25%. Dataset 2 is a more challenging alert sequence, given its high entropy and low repetitive rate. From Figure 41. , we observe that the Multiple Simple Bayesian and the Variable-Order Markov Model score badly on prediction accuracy. The Variable Matching is the second best to

Single-scope blending. The difference between Single-scope blending and Variable Matching is about 30%. The single-scope blending however, suffers in terms of computation time as shown in Figure 43. and Figure 44. .

Figure 45. and Figure 46. show the prediction accuracies and computation time comparison among the predictors for 161 batches of 100 alerts, generated by combining dataset 1 and 2. The significant tests for comparing the predictor's performance against the attention technique are shown in Table 7 and Table 8. Table 7 says that all predictors except the best-first search are significantly different than the attention technique in both paired and 2-group T-test. The best-first search and the attention techniques have similar prediction accuracies as shown by the 2-group t-test. We also observe in Table 8 that the attention technique is significantly faster than the best-first search.

Figure 41. shows that performance difference between the attention predictor and the other predictors are more significant in dataset2, which has higher entropy. To explore this further, the prediction results in Figure 45. are dissected by entropy and plotted in Figure 47. . When entropy is less than 1, there is no significant difference among the predictors as shown in both paired and 2-group t-test in in 0 and 0. When entropy is at least 1, there is significant difference between the attention predictor and the other predictors. As entropy increases, the difference in prediction accuracy increases. Note that like the attention predictor, the variable matching prediction accuracy tends not to decrease as much as the rest as entropy increases. This is because the variable matching predictor is also able to predict unseen percept. It is interesting to note that as entropy increases, the computation time increases. This is because as entropy increases, the number of new situations increases, which should increases the computation time. The computation time at entropy [6,7) appear to decrease even though the entropy is greater. It is likely an outlier.

0 and 0 provide more insight into why Single-scope blending performs much better in dataset 1. 0 shows that the Single-scope blending correctly detected 59.56% of the 1590 alert classes in dataset 1. Detection is defined as the correct prediction of a distinct alert at least once. The explanation is illustrated in 0. 0 reads: There are 643 distinct alerts that occur only once. Out of these 643 occurrences, Single-scope blending

detected 163 while multiple simple Bayesian and variable-order Markov model detected none. This shows that Single Scope bending is able to predict unseen categorical data. As the number of occurrence increases, multiple simple Bayesian and variable-order Markov model catch up eventually and achieve similar accuracy performance as the Single-scope blending when occurrences reach 10.

	SSB	MSB	VOMM
Alert Class Detected	947	379	375
%	59.56%	23.84%	23.58%

Legend: Statistical Lookup Table (SLT), Variable Matching (VM), Multiple Simple Bayesian (MSB), Simple Bayesian Mixture (SBM), Variable-Order Markov Model (VOMM), Single-scope blending (SSB).

Table 11 Dataset 1: Alert Class Detection.

Number of Occurrence	Number of Alerts	SSB Detects	MSB detects	VOMM detects
1	643	163	0	0
2	751	621	230	242
3	52	34	27	14
4	88	80	77	74
5	5	0	0	0
6	11	10	8	8
7	3	3	1	1
8	2	2	2	2
9	4	4	3	3
10	3	3	3	3

Legend: Statistical Lookup Table (SLT), Variable Matching (VM), Multiple Simple Bayesian (MSB), Simple Bayesian Mixture (SBM), Variable-Order Markov Model (VOMM), Single-scope Blending (SSB).

Table 12 Dataset 1: Effect of Frequency on Detection Rate.

0 and 0 provide more insight into why single-scope blending performs much better on dataset 2. 0 shows that the single-scope blending correctly detected 47.63% of the 9619 alert classes in dataset 2. Detection here refers to the correct prediction of a distinct alert at least once. The explanation is illustrated in 0. 0 reads: There are 2157 alert classes that occur only once. Out of these 2157 occurrences, single-scope blending detected 455 while multiple simple Bayesian and variable-order Markov model detected none. As the number of occurrence increases, multiple simple Bayesian and variable-

order Markov model catch up eventually. Unlike dataset 1, the detection performances do not even out when the number of occurrence reaches 10.

	SSB	MSB	VOMM
Distinct **Alert Detected**	2050	807	644
%	47.63%	18.75%	14.96%

Legend: Statistical Lookup Table (SLT), Variable Matching (VM), Multiple Simple Bayesian (MSB), Simple Bayesian Mixture (SBM), Variable-Order Markov Model (VOMM), Single-scope Blending (SSB).

Table 13 Dataset 2: Alert Class Detection.

Number of Occurrence	Number of Alerts	SSB Detects	MSB detects	VOMM detects
1	2157	455	0	0
2	1652	1226	477	332
3	122	73	80	48
4	197	170	141	146
5	24	10	11	12
6	45	36	27	30
7	19	9	5	8
8	16	12	8	10
9	7	4	4	5
10	8	5	5	5

Legend: Statistical Lookup Table (SLT), Variable Matching (VM), Multiple Simple Bayesian (MSB), Simple Bayesian Mixture (SBM), Variable-Order Markov Model (VOMM), Single-scope Blending (SSB).

Table 14 Dataset 2: Effect of Frequency on Detection Rate.

The limitation of the single scope blending is the computation complexity. Figure 43. and Figure 44. show the run time is much slower than the current prediction techniques. Nevertheless, is the attention technique is much faster than the traditional backtracking approaches.

In a separate study, Khong [72] studied the alert prediction accuracies by using a tool called Pytbull to attack a simulated network. Pytbull is capable of 11 classes of attacks that amount to more than 250 types of attack. In the setup, there were three attackers that run Pytbull and three victim machines that run the SNORT intrusion-detection system. The prediction accuracy is Figure 50. . In this experiment, the attackers randomly chose a victim at every 10 seconds interval. The result is shown here to illustrate that the prediction accuracy is consistent with the results from our dataset.

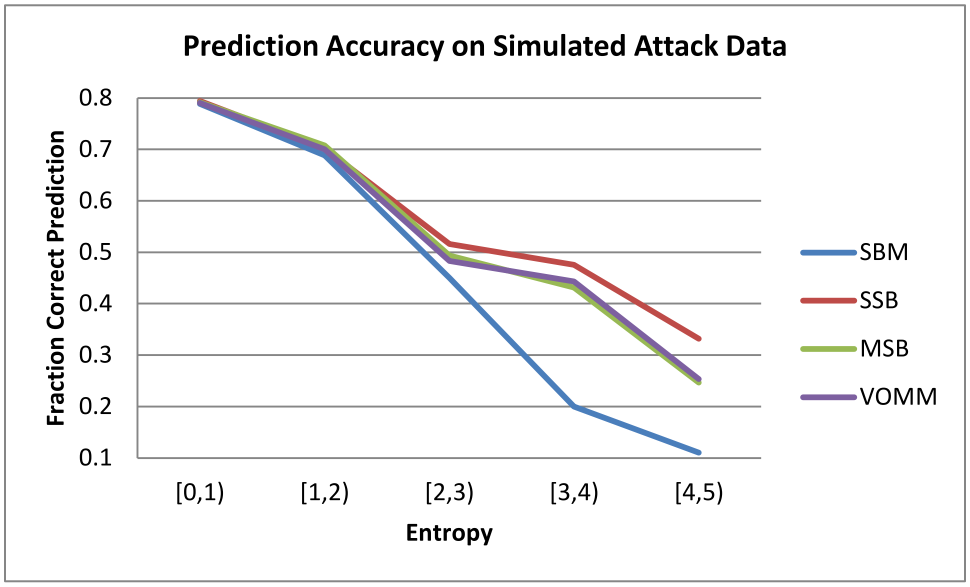
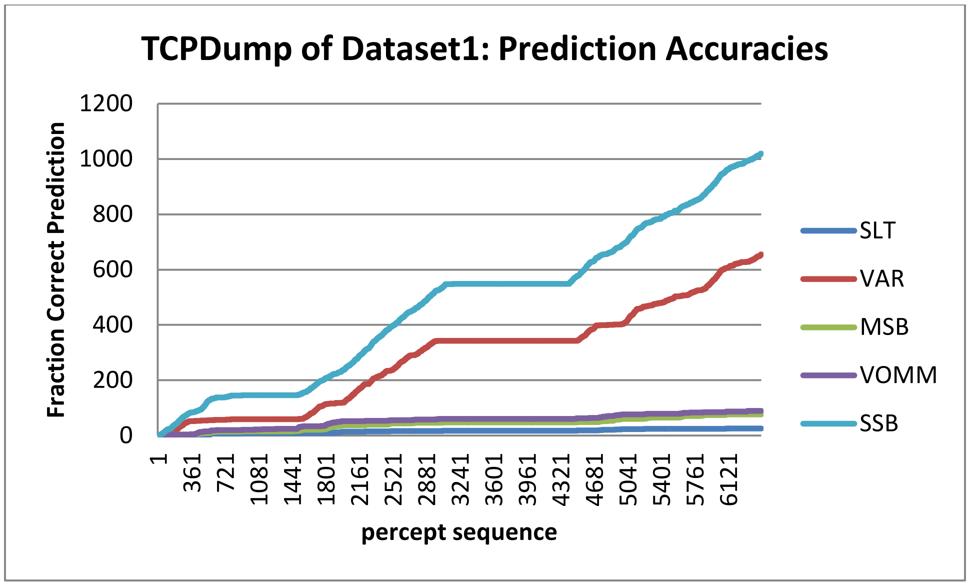


Figure 50. Prediction Accuracy on Simulated Attack Data

We have also run the prediction algorithms on the TCPDump file that accompanied the Snort Alert sequence. The results are listed in Figure 51. . Predicting the network traffic flow can be useful in cyber security since it is a precursor to alerts being generated. Dataset 1 has about 6480 alerts but has about 300,000 TCP data. Figure 51. only shows the first 6000 prediction. The single-scope blending and variable matching predictors obtained much better prediction accuracy than the other predictors. The single-scope blending predictor is also significantly better than the variable matching predictor.



Legend: Statistical Lookup Table (SLT), Variable Matching (VM), Multiple Simple Bayesian (MSB), Simple Bayesian Mixture (SBM), Variable-Order Markov Model (VOMM), Single-scope Blending (SSB).

Figure 51. Dataset 1 TCPDump: Prediction Accuracy.

G. CONCLUSION

In this experiment, we show that relational time-series learning and prediction can be used to predict cyber intrusion alert. This is the first time that such integrated online learning and prediction capability is demonstrated on cyber alert prediction. We have also showed that the single-scope blending performs much better than the current prediction techniques. We have also provided an analysis that explains why is single-scope blending better, which is due to the ability to predict at dynamic and changing environments. The ability to predict unseen categorical data point shows that it can handle new and unknown domain much better than the other prediction techniques.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. EXPERIMENT 3: SIMKIT EVENT PREDICTION

A. INTRODUCTION

We also tested prediction for a class of simulations that run on discrete event simulation. Discrete event simulation has been widely used in many kinds of studies in the Naval Postgraduate School. For example, students from the System Engineering Department built a Simkit simulator to study ship-to-ship and ship-to-shore supplies transfer in riverine operations [73]. In another study, a discrete event simulator was developed to model the San Francisco Harbor to look into port security matters [74]. Tan [53] also uses a discrete event simulator to model the Singapore harbor. The Combat XXI simulator, which is also a discrete event simulator, is a combat simulation tool used by the US Army Training and Doctrine Command (TRACDOC).

One distinct feature about discrete event simulation is the scenario evolvement through a sequence of events occurrence. A discrete even is usually specified by a time of occurrence, an event name, and a list of zero or more attributes of the event. Therefore, a discrete event can be regarded as a percept in our context and the sequence of discrete events is a relational time series. The sequence of discrete event exists in all discrete event simulation. By showing that relational time-series prediction can be used to predict events on a discrete event simulator, it suggests that we can use it successfully on other discrete event simulators because the underlying representation is the same.

B. DISCRETE EVENT SIMULATION

We used a discrete event simulation toolkit called Simkit. Many discrete event simulation library are built on Simkit [6], a set of Java programming language libraries that support easy development of discrete event simulators that are based on event graph and listener framework. Discrete event simulations execute by running through a sequence of scheduled events. This list of events is managed by the event scheduler in Simkit. Each event is equivalent to a relational percept. In Figure 52. , event A is a percept of arity zero while event B is a percept of arity one. The event graph in the figure is read as: the occurrence of Event A schedules Event B to occur after time t when condition (i) is met. j is the input value that is required for the argument k in Event B. A sequence of events is a natural relational time-series that exists in all Simkit simulators or possibly any discrete event simulations.

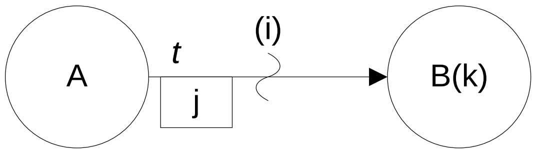


Figure 52. Event Graph (from [75]). Circles are Events. The Arguments in the Event Parentheses are the Attribute of the Event. The Arrow Marks the Relation between Event A and Event B. t is the Time of Event B after Event A Has Occurred. J Is the Parameter That is Passed from Event A to event B. The i in the Parentheses above the Curvy Line Is the Condition for the Relation.

Several events can be grouped together to form one component. The mechanism that allows inter-component communication is the listener framework. Figure 53. shows a component "Listener" that listens to the events occurrences in the "Source" component. Professor Buss provided a Java class SimpleEventDumper that extends from BasicSimEntity to subscribe to interested events that are generated by the scheduler. We simply need to connect any simulated entities to the SimpleEventDumper with a listener adaptor.

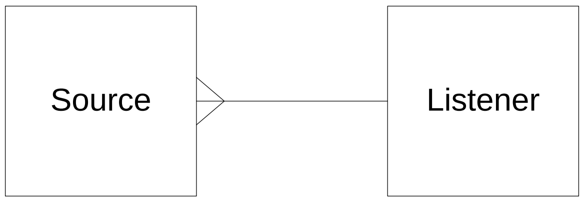


Figure 53. SimEventListener Relationship: Component Listener "Hears" all of Component Source's Events [76].

C. MARITIME SIMULATION

We use the Harbor Simulator in [53]. A snapshot of the simulator is shown in Figure 54. . The simulator models the movement of ship movements as discrete events. Each discrete event for movement indicates a start moving event with a start location and velocity, or a stop moving event. A ship model moves through a path with a sequence of start and stop moving events. In the scenario, each ship is assigned a path, which is randomly chosen from a list of predefined paths that model the sea line of communication. A sample sequence of events generated from the maritime simulator is given in Figure 55. . Each event contains the ship ID, starting location and velocity. In discrete event simulation, states trajectories are piecewise constant. However, moving actors such as moving ships move in constantly changing continuous space. Therefore, the locations of moving actors are modeled implicitly and are not considered states in discrete event simulation. Instead, the states that represent the implicit states are a combination of the start location and velocity. With a start location and velocity, we can calculate the exact location from the equation of motion at a given time. An actor with a constantly changing direction can be modeled as a sequence of changing velocity with possible same starting location. While there are other information available in the discrete event simulation, we only use the fields as shown in Figure 55. .

D. UTILITY OF PREDICTION

The purpose of using relational time-series prediction with discrete event simulation is to allow software agents to make predictions based on the events encountered instead of looking at the ground truth information available in the event scheduler. In [53], there are patrol craft chasing after suspicious watercraft based on suspicious craft’s current location. Henceforth, the patrol crafts always end up at the suspicious craft’s previous location. The behavior can be easily enhanced by chasing after the predicted location.

Another possible application is to demonstrate a predictive decision support system that is capable of anticipating shipping events. If a ship deviates too much from our prediction, meaning if we keep failing to predict a ship’s next event, this ship is highly suspicious. This is in line with a recent neuroscience theory [2] that describes that if the prediction keeps failing, cognition is in a state of instability and there is a need to activate another segment (a set of percepts with regard to a context such as a particular room, action or actor) to improve prediction. The new segment might be “piracy” instead of “transit” or “Enter-Harbor” segment. Changing segment is beyond the scope of this demonstration. Another possible purpose is to model a human monitoring the maritime traffic. The human model can flag out a suspicious ship based on prediction error.

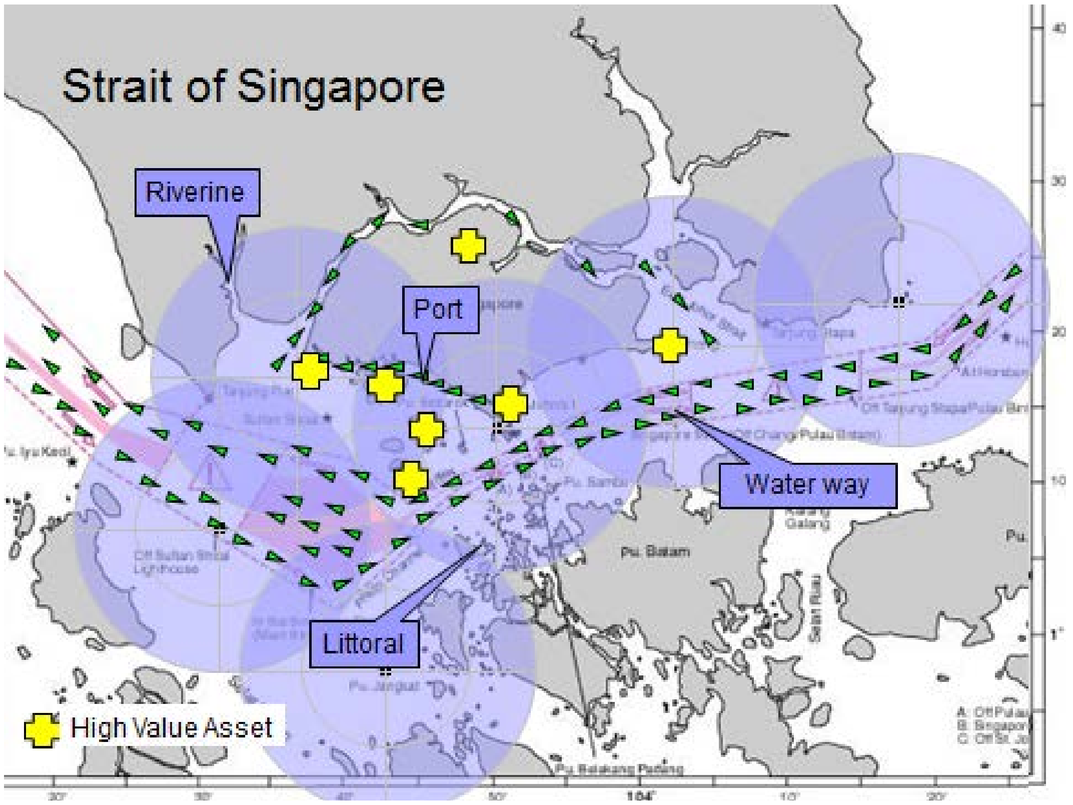


Figure 54. A Simkit-based Singapore Harbor Simulator. The Green Triangles are Ship Movements.

```
Time   EventName objectID    location          velocity
136027 StartMove SmallBoat75 (-40.000,311.000) [-3.401,-0.249]
136039 EndMove   SmallBoat75 (-81.000,308.000) [0.000,0.000]
136073 StartMove SmallBoat76 (-177.000,320.000)[-2.202,-2.603]
136088 EndMove   SmallBoat76 (-210.000,281.000)[-3.402,-0.235]
136088 StartMove SmallBoat76 (-210.000,281.000)[-3.402,-0.235]
136113 EndMove   SmallBoat76 (-297.000,275.000)[3.009,-1.605]
136113 StartMove SmallBoat76 (-297.000,275.000)[3.009,-1.605]
136128 EndMove   SmallBoat76 (-252.000,251.000)[2.576,2.234]
136128 StartMove SmallBoat76 (-252.000,251.000)[2.576,2.234]
136161 EndMove   SmallBoat76 (-169.000,323.000)[0.000,0.000]
137876 StartMove SmallBoat77 (-84.000,108.000) [0.000,0.000]
```

Figure 55. A Relational Time-series of Small-boat Events.

E. METHODOLOGY

Our prediction task is to predict the next event the agent may see, given the past event sequence. The controls of the experiment are as follow.

1. Test Data

Each discrete event corresponds to one percept. The event name is the relation. The object ID, location and velocity are the object constants of the relational percept. We obtained five sequences of roughly1400 discrete events from separate simulation runs of the maritime simulator. Five runs are sufficient to show the statistical significance of the differences in prediction performance. While there are several classes of ship in the scenario, we only collect the sequence of events on one class: the small boats. The reason is that suspicious craft are usually the small boats. In this experiment, we only work on the small boat. When the simulation starts, each small boat will select one of the pre-fixed paths randomly and execute its movement.

2. Percept of Arity 2

A discrete event is a percept. An event sequence is a percept sequence. Discrete events come in relational table form with many fields. Each event has an arity greater than two. In this experiment, we use the fields: Event Name, object ID, velocity and location. The relational representation of an alert is "Event_Name (object_ID, location, velocity)". The arity is 3. Therefore, we must transform one 3-arity alert percept into an intermediate representation, which is a group of multiple arity-2 percepts. We must also account for multiple 3-arity alerts that form a situation. In a situation, we assign a record number to each event by the order in the situation. The first event is record 1, the second event is record 2, and so on. In a percept, when a record number is assigned, we relate the field elements in the event to that record number using the following method: $R_n(C_1, C_2, \ldots, C_q) = F_1 (Rn, C_1), F_2 (Rn, C_2), \ldots F_q (R_n, C_q), = F_i (R_j, C_i)$, If a record has four fields, we convert to arity 2 by saying that field i of record n is C, where i is the field or column number, n is the record number and C is the constant. The conversion process is the same as the one used in network alert prediction.

3. One Vote

A given situation can have multiple target percepts (possible predictions). Different prediction techniques are capable of different number of prediction. For Bayesian, all previously encountered percepts are possible. For Markov, the lower the order, the greater number of prediction. Situation matching prediction depends on the number of target percepts. Hence, to be fair, we ask each technique to produce their best guess.

4. Next Percept

A prediction may not occur immediately and another percept may arrive before the predicted percept arrives. There are several ways to consider prediction with a time interval. We can use a constant time interval and or time of occurrence between situation and target. For second method, we can collect the intervals compute mean and standard deviation (SD). If the predicted percept falls within 1SD, 2SD or 3SD, we can consider it correct. There are two ways of collecting the interval, from situation's perspective or target's perspective. As a result of all these complexity, we fall back on the strictest measure of effectiveness: predict the next one. Prediction with time is studied in the sensitive studies at chapter 0 where 2SD is used on target's perspective.

5. Prediction Accuracy

The predicted percept $\mathrm{p} = \mathrm{r}(\mathrm{c}_1, \mathrm{c}_2, \ldots, \mathrm{c}_\mathrm{m})$ is said to be correct if the next percept $\mathrm{p}' = \mathrm{r}'(\mathrm{c}_1{}', \mathrm{c}_2{}', \ldots, \mathrm{c}_\mathrm{m}{}')$ is such that $\mathrm{p}' = \mathrm{p}, \mathrm{r}' = \mathrm{r}, \mathrm{c}_\mathrm{i}' = \mathrm{c}_\mathrm{i}$ for all $\mathrm{i} = 1,2, \ldots, \mathrm{m}$. Prediction accuracy = $\frac{c}{n}$ where n is the number of percept receive and c is the count of correct prediction. In discrete event simulation, the paths in which the watercrafts follow are a set of waypoints, which can be seen as categorical data. The maritime simulator is a deterministic model. Therefore, all watercraft on the same path will have the same start and end location. This is a reasonable assumption since watercraft usually follows the sea lines of communication that consist of traffic buoys at designated location. In a stochastic model where location and velocity are a distribution from the waypoints, we can consider the prediction is correct if the predicted location and velocity are close to the actual ones.

6. Hardware

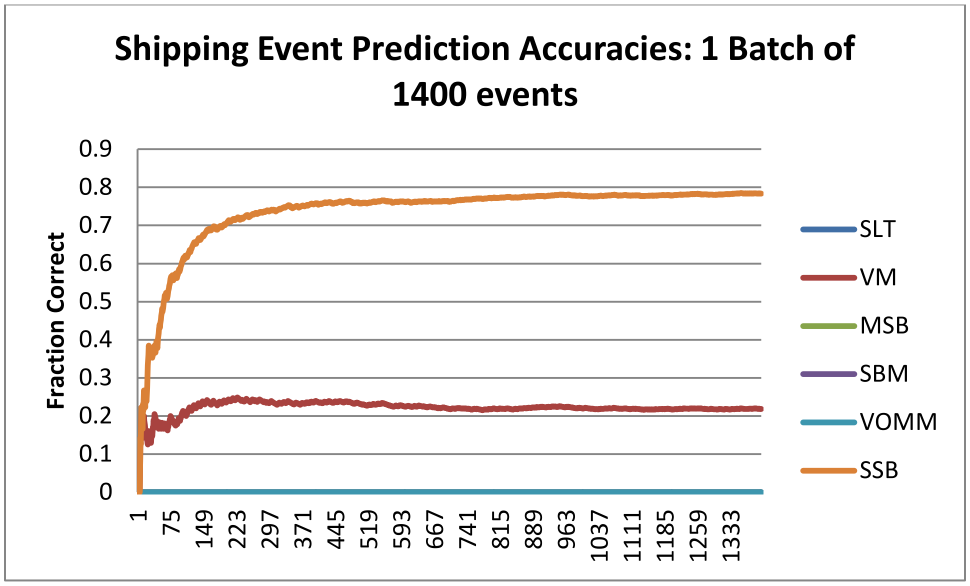
All experiments were run on a Dell XPS Laptop i7 1.87Ghz 16GB RAM with Windows 7.

7. Time Window ize

The time window used in the experiment is 0.1sec, the same time window used in the previous work experiment.

F. RESULTS

The variation of prediction accuracy over time for the first run is illustrated in Figure 56. . We observe that the single-scope blending prediction technique has the highest prediction accuracy, followed by the variable matching. The other techniques did not get any prediction correct.

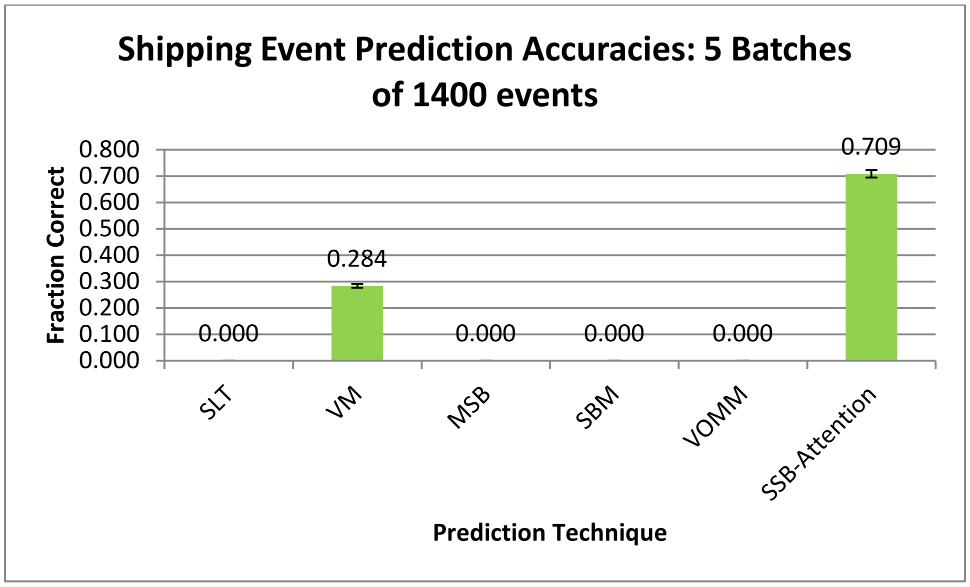


Legend: Statistical Lookup Table (SLT), Variable Matching (VM), Multiple Simple Bayesian (MSB), Simple Bayesian Mixture (SBM), Variable-Order Markov Model (VOMM), Single-scope Blending (SSB).

Figure 56. Shipping Event Prediction Accuracies: 1 Batch of 1400 events.

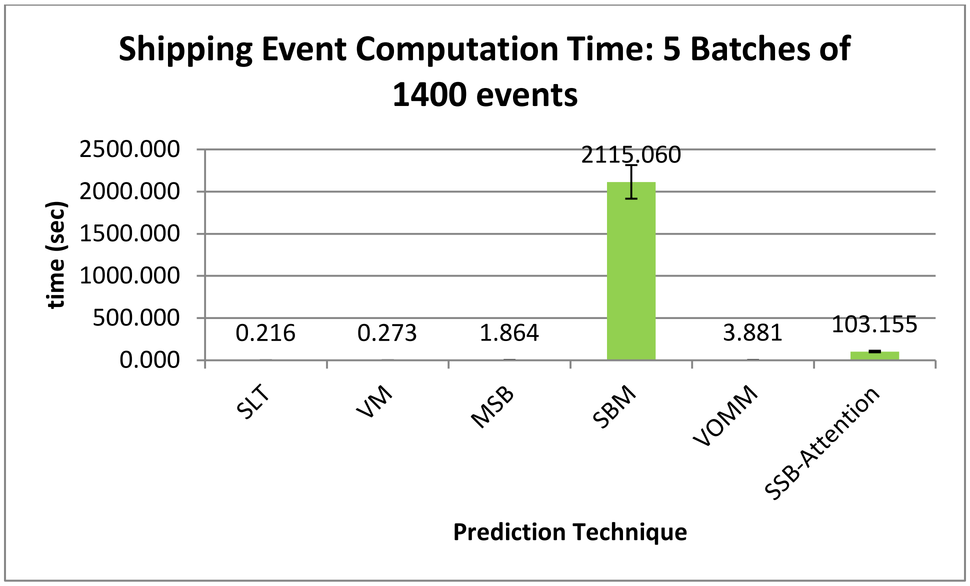
The averaged prediction accuracy with standard error over five run are shown in Figure 57. . Again, we observe that the single-scope blending prediction technique has the highest prediction accuracy, followed by the variable matching for all five runs. The averaged computation time with standard error over five run are shown in Figure 58. .

We observed that the single-scope blending is much slower than most other techniques. The p-values of student t-test of statistical significance are given in 0. Since the p-values are less than 0.05, we conclude that the results are statistically significant.



Legend: Statistical Lookup Table (SLT), Variable Matching (VM), Multiple Simple Bayesian (MSB), Simple Bayesian Mixture (SBM), Variable-Order Markov Model (VOMM), Single-scope Blending (SSB).

Figure 57. Shipping Event Prediction Accuracies: 5 Batch of 1400 events.



Legend: Statistical Lookup Table (SLT), Variable Matching (VM), Multiple Simple Bayesian (MSB), Simple Bayesian Mixture (SBM), Variable-Order Markov Model (VOMM), Single-scope Blending (SSB).

Figure 58. Shipping Event Computation Time: 5 Batch of 1400 events.

Batch	**SLT**	**VM**	**MSB**	**SBM**	**VOMM**	**VOMM-VAR**	**SSB-attention**
paired	0.00	0.00	0.00	0.00	0.00	0.00	1.00
group	0.00	0.00	0.00	0.00	0.00	0.00	1.00

Legend: Statistical Lookup Table (SLT), Variable Matching (VM), Multiple Simple Bayesian (MSB), Simple Bayesian Mixture (SBM), Variable-Order Markov Model (VOMM), Variabiized Variable-Order Markov Model (VOMM-VAR), Single-scope Blending (SSB).

Table 15 P-value for Significant Tests on Shipping Event Prediction Accuracies.

G. DISCUSSION

Other than variable matching and single-scope blending predictors, the other techniques did not get any predictions correct. Single-scope blending is significantly superior to variable matching. The reason for this phenomenon is because all ships in the scenario are distinct and each ship only passes through the path only once. Therefore, all shipping events are distinct and novel. For example, SmallBoat19 will only be at location (-13.0,160.0) and velocity [-2.343,2.477] only once. All events are new and unseen. The event that SmallBoat19 located at (-13.0,160.0) with velocity [-2.343,2.477] has never occurred in the percept history until time 38.9. As a result, Bayesian and Markov chain methods fail to achieve any correct predictions in this application. For new precept prediction, only variable matching and single-scope blending predictors are capable because of the underlying isomorphism and unification processes. As the repetitiveness increases, we expect the other techniques to catch up. Nevertheless, the purpose of this demonstration is to show how each technique performs when all percepts are unseen. From this exercise, we show that the single-scope blending is capable of achieving an average prediction accuracy of 70% without any domain knowledge or any heuristics.

H. CONCLUSION

This experiment shows that we can apply relational time-series learning and prediction on the event list in Simkit, thereby demonstrating that relational time-series learning and prediction can be applied on discrete event. We have again shown that the single-scope blending has better prediction accuracy than the other predictors.

THIS PAGE INTENTIONALLY LEFT BLANK

VII. ALGORITHMIC ANALYSIS

A. INTRODUCTION

We have introduced the single-scope blending prediction technique in chapter IV and show that it has significantly better prediction accuracy than all other prediction techniques mentioned in chapter III, tested on Pymud percept prediction, Snort alert prediction and Simkit event prediction. We have also described three variations of the single-scope blending prediction technique: backtrack, greedy best-first search and attention. While they have similar prediction accuracy, the greedy best-first search and the attention technique are not guaranteed to find the best match because of the underlying greedy approach of searching for a unification solution. The first objective is to do a more detailed completeness analysis, which compare these two techniques with the complete backtrack technique.

We have also shown that the greedy best-first search is faster than backtracking and attention based model is faster than greedy best-first search. While we have reduced the computation complexity from exponential time $O(2^n)$ to quadratic time $O(n^3)$ and $O(n^2)$ where n is he number of object constants in each situation, the exponent of greater than one is still a concern. Hence, the second objective is to do a scalability analysis to see how scalable (in the number of object constants and number of situation) the greedy best-first search and the attention model are.

B. COMPLETENESS ANALYSIS

Completeness is defined as the ability to consider all possible solutions. In our case, we want to consider all possible unifications of object constants from the previous situation and the current situation, to choose one set of unifications that give us the best similar score. The backtrack technique of searching for a common subgraph in the current and previous situations is complete because it compares the outcome of all possible unifications before arriving at the set of unification that has the best similar score. The backtrack technique uses a depth-first search, and the deepest depth is fixed because the number of object constants in the current and previous situation is fixed. The greedy best-first search only searches for the best two paths of the search tree, and is therefore incomplete. The attention is also incomplete because it only considers one path of the search tree. We want to see the effect of incompleteness through a more detailed analysis: failure and rotational analysis, which we will discuss below.

1. Failure Rate Analysis

Failure rate analysis compares the outcome of the incomplete best-first search and attention technique with the outcome of the complete backtrack technique. In all prediction events, we want to see how many of them in which the greedy best-first search and attention techniques are able to achieve exactly the same outcome as the backtrack technique in the Pymud benchmark environment. There are three possible levels in which we can explore the incompleteness. Level 1: selected previous situation, level 2: unification, level 3: similarity score. The first level compares the selected most similar previous situations by the greedy best-first search and the attention techniques with the backtrack technique. The second level compares the set of unifications chosen to bind object constants from one situation to the other, given that the selected previous situation is the same. The third level is to compare the similarity score of the selected previous situation and the current situation, among the three techniques.

We compare the outcome from the 40 batches of 100 percepts on Pymud. The results for the level 1 test are given in Table 16. Out of the situations selected by the backtrack technique, the greedy best-first search technique selects the same previous situations at about 96.375% of the time while the attention technique selects the same previous situation at about 90.55% of the time. For the level 2 test the greedy best-first search technique unification outcome is 91.3% exactly the same as the backtrack technique while the attention technique only achieve 83.9%. The average similarity score difference of greedy best-first search technique and attention when compared to Backtrack are 0.0156 and 0.0343, respectively.

Compared to Backtrack	BFS	Attention
Selected previous situation	0.96375	0.9055
Isomorphism Outcome	0.913	0.839
Average difference in similarity score	0.0156	0.0343

Table 16 Failure Analysis Outcome.

From this study, the greedy best-first search technique is closer to the backtrack than the attention technique. Despite these differences, backtrack, greedy best-first search and attention techniques are able to achieve similar prediction accuracies. This is because of the dynamic and noisy nature of the relational time series, which we discuss below.

It is possible for different set of unifications outcomes to have the same similarity score because the set of unifications may not be used to generate the prediction. For example, consider a previous situation that has two object constants, dagger1 and place1, and a percept of arity one such as commandless(). If the current situation has two object constants: goblin2 and agent2, and a percept of arity one such as commandless(). If no object constant type information is given, it does not matter if goblin2 unifies with dagger1 or place1, the only similar percept is commandless()

It is possible for different prediction techniques to select the same situation even though unifications choices are different, which result in different similarity scores. This is because situations are selected based on the relative similarity scores of other previous situation. Since the similarity score differences are small, the ideal previous situation should emerge similarly when compared to other less favorable previous situation.

It is possible to have different situations selected and yet have the same prediction outcome. This is because different situations can have the same target percept. Furthermore, different situations may be similar but are deemed as two different situations because of some trivial differences.

As a result, minor differences in unification choices and selection process can still achieve the similar prediction accuracies.

2. Rotational Sampling for Attention Technique

The attention technique pairs up all possible object constants from the previous and current situations, generates a constant-to-constant similarity score, and selects the pairs that have the highest score. Pairs that are not selected will be discarded. This is a one-time process. Rotational sampling iterates through the selection process multiple times. Each time, one pair is rotated to the top of the list, regardless of its score, so that every pair will be selected at least once. At the end of one iteration, the situation similarity score will be computed. The pairs from the iteration that has the highest situation similarity score will be used for prediction.

We will illustrate the incompleteness using an example. Figure 59. shows a list of possible bindings, sorted based on their node similarity scores. If we process from the top, we will use the bindings as shown in Figure 60. .

```
Constant1,          constant2,          node similarity
['Record1',         'Record1',          [1, 1, 1, -0.0]]
['Record2',         'Record2',          [1, 1, 1, -0.0]]
['Record1',         'Record2',          [0, 1, 1, -0.0]]
['63.205.26.73',    '69.64.58.18',      [0, 1, 1, -0.0]]
['189.250.177.224', '69.64.58.18',      [0, 1, 1, -0.0]]
['Record2',         'Record1',          [0, 1, 1, -0.0]]
['189.250.177.224', '63.205.26.77',     [0, 0, 1, -1.0]]
['189.250.177.224', '63.205.26.80',     [0, 0, 1, -1.0]]
['63.205.26.73',    '63.205.26.77',     [0, 0, 1, -1.0]]
['63.205.26.73',    '63.205.26.80',     [0, 0, 1, -1.0]]
```

Figure 59. Possible unification

```
Constant1,          constant2,          node similarity
['Record1',         'Record1',          [1, 1, 1, -0.0]]
['Record2',         'Record2',          [1, 1, 1, -0.0]]
['63.205.26.73',    '69.64.58.18',      [0, 1, 1, -0.0]]
['189.250.177.224', '63.205.26.77',     [0, 0, 1, -1.0]]
```

Figure 60. Selected unification

```
Constant1,           constant2,           node similarity
['Record1',          'Record2',           [0, 1, 1, -0.0]]
['Record2',          'Record1',           [0, 1, 1, -0.0]]
['189.250.177.224',  '63.205.26.77',      [0, 0, 1, -1.0]]
['189.250.177.224',  '63.205.26.80',      [0, 0, 1, -1.0]]
['63.205.26.73',     '63.205.26.77',      [0, 0, 1, -1.0]]
['63.205.26.73',     '63.205.26.80',      [0, 0, 1, -1.0]]
```

Figure 61. Discarded bindings

The attention model is incomplete because the unexplored unifications as shown in Figure 61. can potentially lead to a better situation similarity score, since they have the same node similarity score as some selected unifications. In the unselected unifications, ['189.250.177.224', '69.64.58.18', [0, 1, 1, -0.0]] has the same node similarity score as the accepted unification ['63.205.26.73', '69.64.58.18', [0, 1, 1, -0.0]], which is selected by chance. If ['189.250.177.224', '69.64.58.18', [0, 1, 1, -0.0]] is selected, ['63.205.26.73', '69.64.58.18', [0, 1, 1, -0.0]] will be rejected because of the clash. If ['189.250.177.224', '69.64.58.18', [0, 1, 1, -0.0]] is selected, ['189.250.177.224', '63.205.26.77', [0, 0, 1, -1.0]] will be rejected. From this example, this algorithm is not complete.

We use the rotational sampling technique to check for the effect of incompleteness. During rotational sampling, discarded unification such as ['189.250.177.224', '69.64.58.18', [0, 1, 1, -0.0]] will be rotated to the top once and be accepted as a unification.

The experiment is based on Snort alert dataset 1. We divided the 16000 alerts into 16 batches of 1000 alerts each. The prediction results are displayed in Figure 62. . The prediction accuracies are statistically insignificant. The t-tests results are shown in Table 17. The computation time for rotational attention is as shown in Figure 63. .

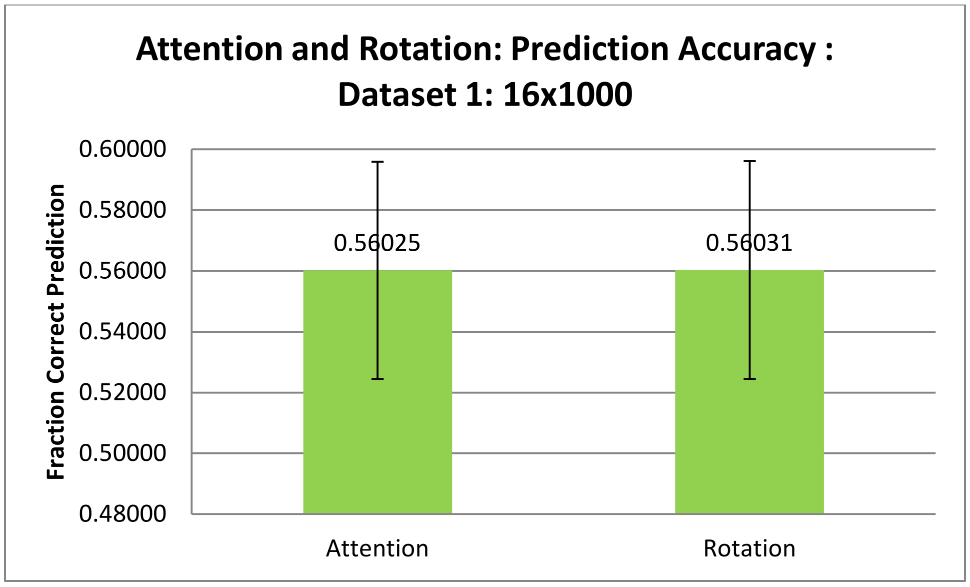


Figure 62. Attention versus Rotational Attention: Prediction Accuracy.

T-Test	p-value
paired	0.89960219
2-group	0.99902179

Table 17 Statistical t-test for Comparing PredictionAaccuracies of Attention and Rotational Attention.

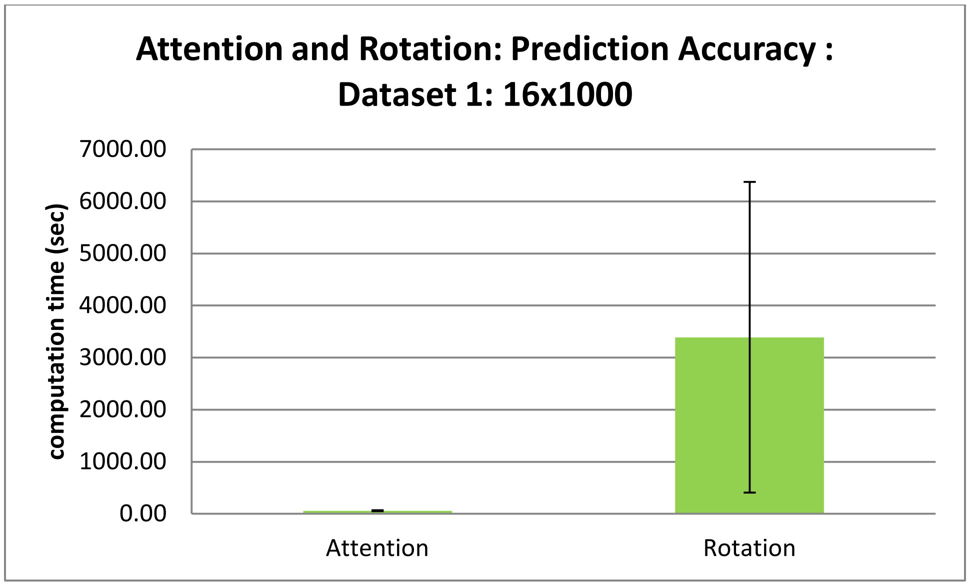


Figure 63. Attention versus Rotational Attention: Computation Time.

From this experiment, we observed that there is insignificant difference on prediction accuracy between the attention and the rotation techniques.

C. SCALABILITY ANALYSIS

In this scalability analysis, we will start with a brief discussion on the theoretical time complexity of the prediction techniques used in the three experiments. Next, we will see how the three variations of single-scope blending techniques perform on long relational time series. In section C3, we will study how these three prediction techniques scale on the number of percept count in one situation. The last section study how these three prediction techniques scale on the number of object constant count in one situation.

1. Theoretical Time Complexity

From the above description of various prediction techniques, the time complexity of statistical lookup table (SLT) for comparing two situation is O(s) where s is the number of situation in the lookup table. The worst case time complexity of variable matching (VM) is $O(s*n^2)$ where n is the number of object constants in one situation, assuming both situation has the same number of constant n, and s is the number of situation in the lookup table. There exist many heuristics to reduce the time complexity, even to O(s) time complexity at the expense of accuracy. For example, if the numbers of constants in two situations are different, there is no matching. The time complexity of multiple simple Bayesian (MSB) is $O(n^2)$ where n is the number of distinct percept. The worst case occurs when each distinct percept is a child of very other distinct percepts. Simple Bayesian mixture (SBM) has a time complexity similar to the multiple simple Bayesian. The worst case complexity of variable order Marov model is O(n) where n is the number of percept, when the entire sequence matches.

The worst case complexity of best-first search is $O(b^m)$ where b is the branching factor and m is the maximum depth of the tree. The branching factor is the possible bindings and maximum depth is the total number of pairings needed. Hence, the maximum complexity is $O(s*n^n)$ where s is the number of situation in the lookup table and n = min (n1, n2) where n1 and n2 are the number of object constants in situation 1 and situation 2. The number of object constants in the tree search that the Greedy BFS must process is mn + a(m-1)(n-1) + a(m-2)(n-2) + .. + a(m-(m-1))(n-(m-1)) where m and n are the number of object constants in situation 1 and 2, respectively, $m \leq n$ and a is the fringe size. Supposed that the number of constants are n on both situations, the greedy best-first search complexity for fringe size a is $O(s*a*n^3)$.

The attention model has to evaluate all possible pairings once. The maximum time complexity is also $O(n*m*s)$ where m and n are the number of object constants in situation 1 and 2, respectively.

2. Scalability: Long Relational Time Series

In this experiment, we run a long time-series for backtrack, greedy best-first search (BFS) and attention prediction techniques. Instead of 100, we want to see the effect of longer sequence up to 1000 percepts. The prediction and computation time results on Pymud over 1000 percepts are as shown in Figure 64. and Figure 65. , respectively. These two charts show that while the prediction accuracies are similar, the computation time for best-first search and attention increase much slower than the backtrack technique. The time complexity for best-first search and attention look linear on Figure 65. . A long time-series (5000) in Figure 66. shows the effect of the exponents after the time-series passed the 1000 percept point.

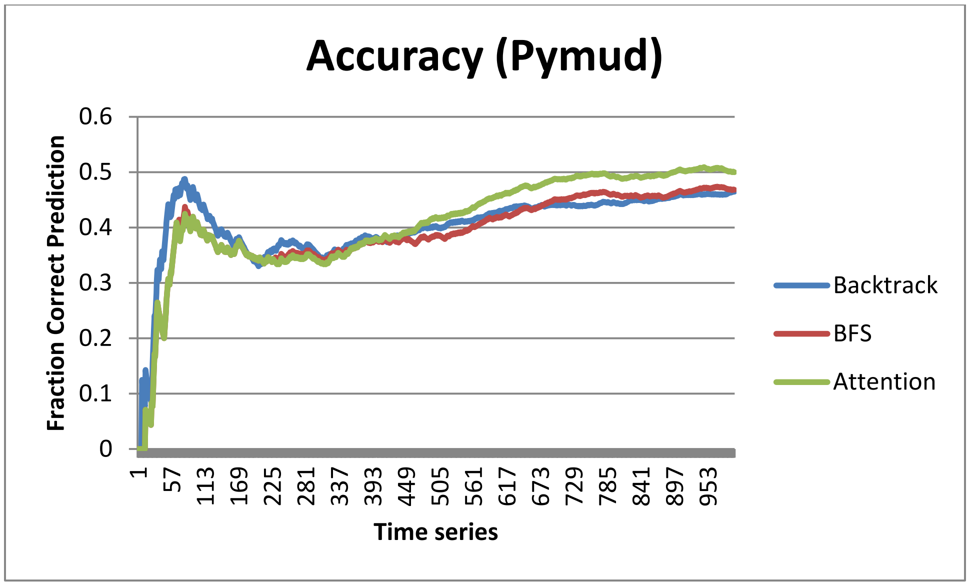


Figure 64. Pymud 1x1000: Prediction Accuracy.

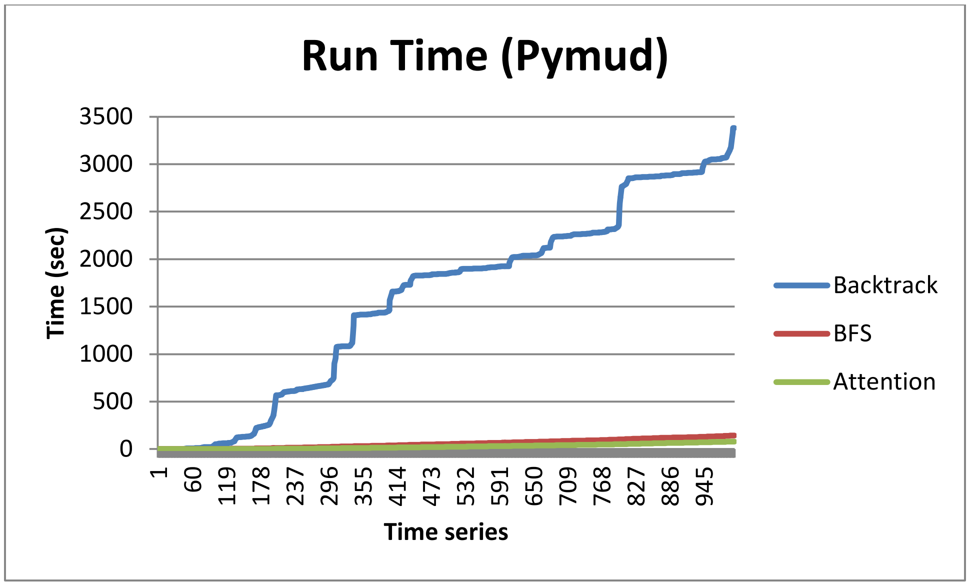


Figure 65. Pymud 1x1000: Computation Time over Time.

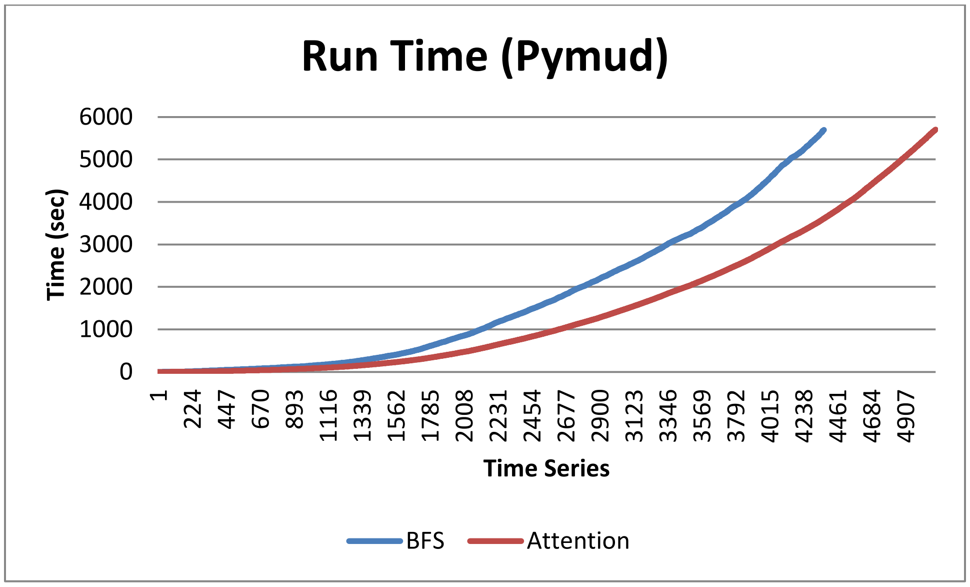


Figure 66. Pymud 1x1000: Computation Time without Backtrack over Time.

Similar observations are made on Snort alert prediction as shown in Figure 67. for prediction accuracy and Figure 68. for run time. Figure 69. shows the computation time in Figure 68. but without the backtrack result, to allow better view on the time performance of best-first search and attention. It can be observed that the run time over a longer time-series for attention increases much slower than the BFS. The run time appear to be linear because unlike in Pymud, the repeat rate is a lot higher in the Snort alert sequence

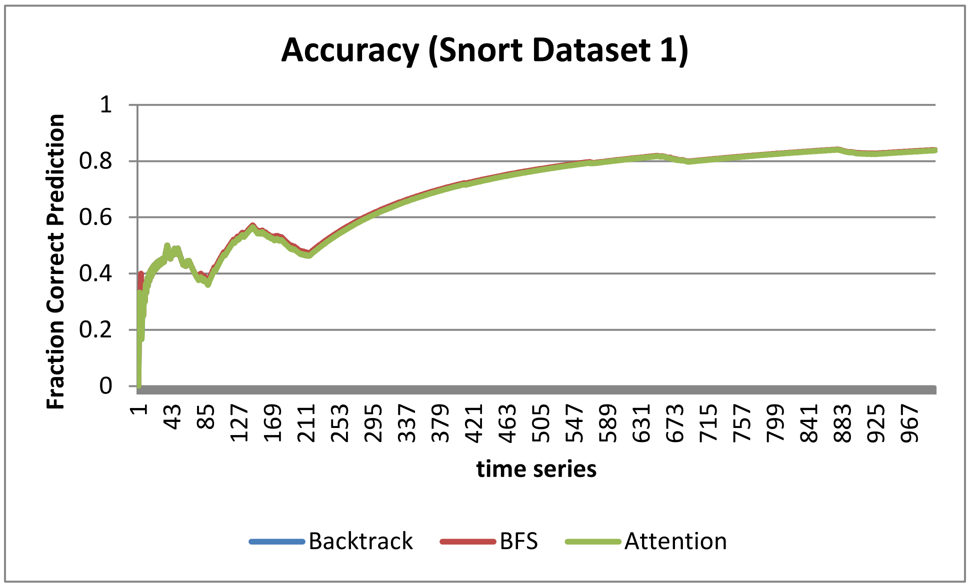


Figure 67. Snort Dataset 1 1x1000: Prediction Accuracy.

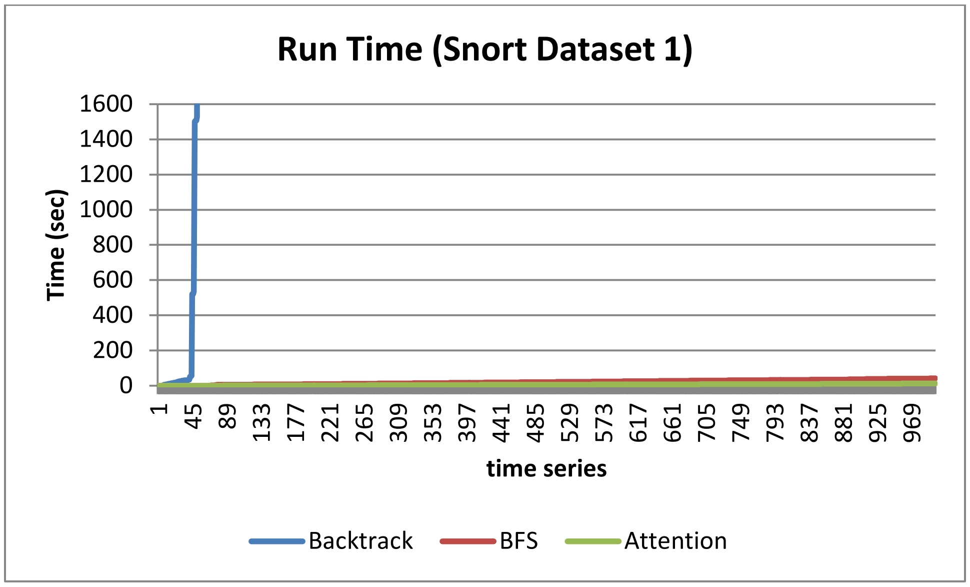


Figure 68. Snort Dataset 1 1x1000: Computation Time over Time.

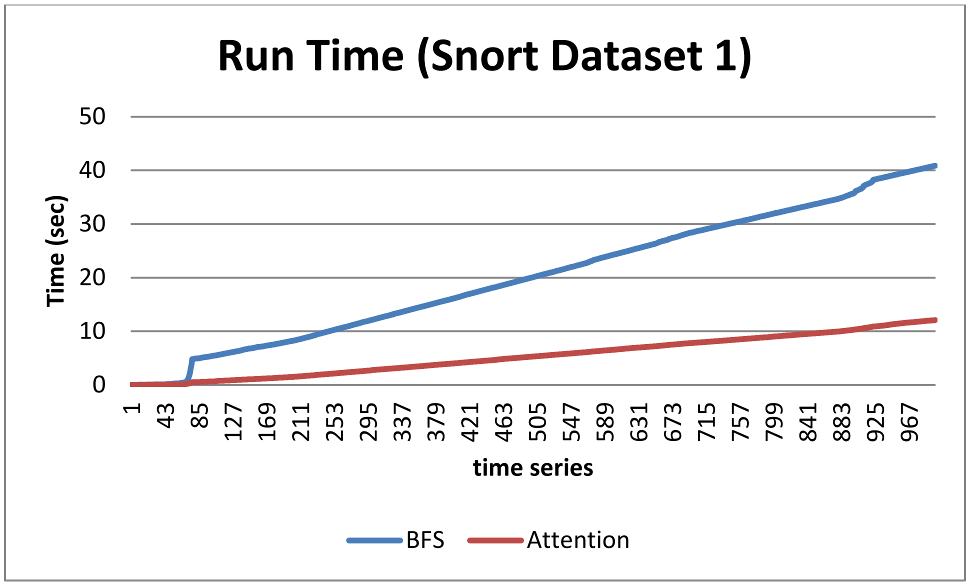


Figure 69. Snort Dataset 1 1x1000: Computation Time without Backtrack over Time.

3. Scalability: Function of Situation Size

In the next test, we evaluate the computation time as a function of number of percept in each situation. In this test, we ignore the prediction accuracies and have an infinite time window. Therefore, the situation size will keep increasing.

The run time outcomes are as shown in Figure 70. and Figure 71. . Figure 71. focus on best-first search and attention. The backtrack technique has obvious complexity issues. Both best-first search and attention run time over situation size is observed to be linear on Pymud for up to 10 percepts in a situation. For Snort alert, Figure 72. shows that the best-first search is slowly blowing up after 5 alerts n a situation in the face of higher arity. Recall that Pymud has percept of arity 0, 1 and 2. Snort alert has a constant arity of 3, which can become a set of atoms of 9 for each alert.

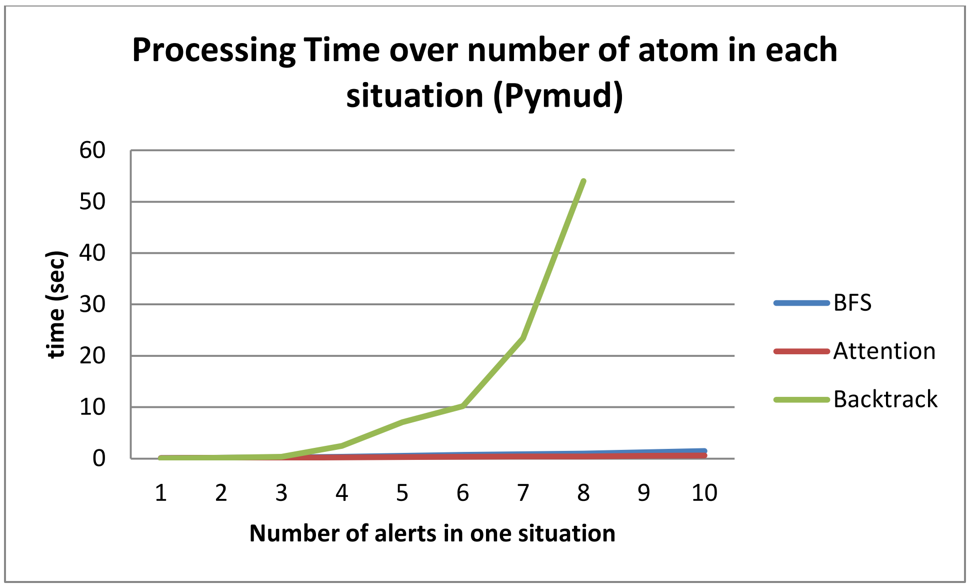


Figure 70. Pymud Computation Time over Situation Size. BFS: Greedy Best-first Search.

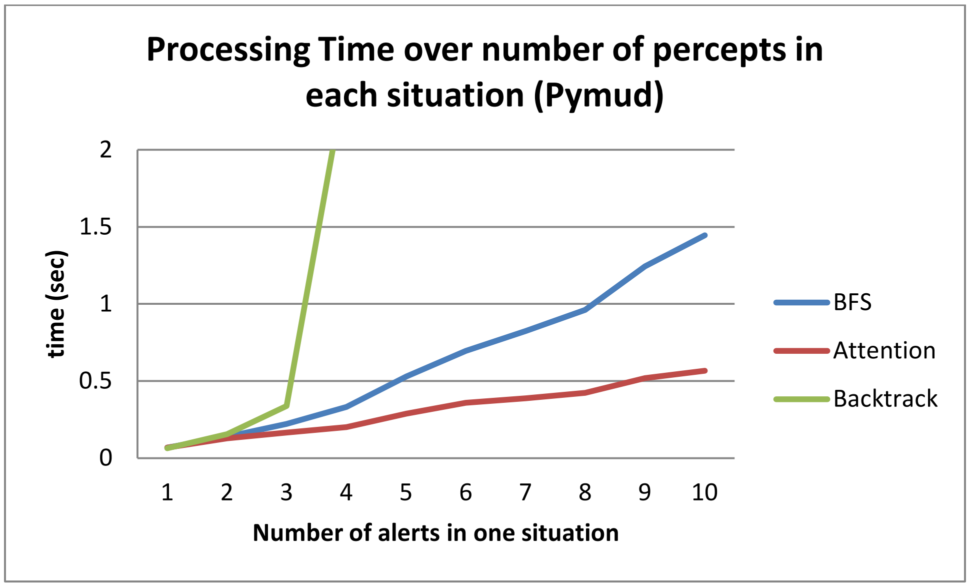


Figure 71. Pymud Computation Time over Situation Size Focusing on Greedy Best-first Search and Attention.

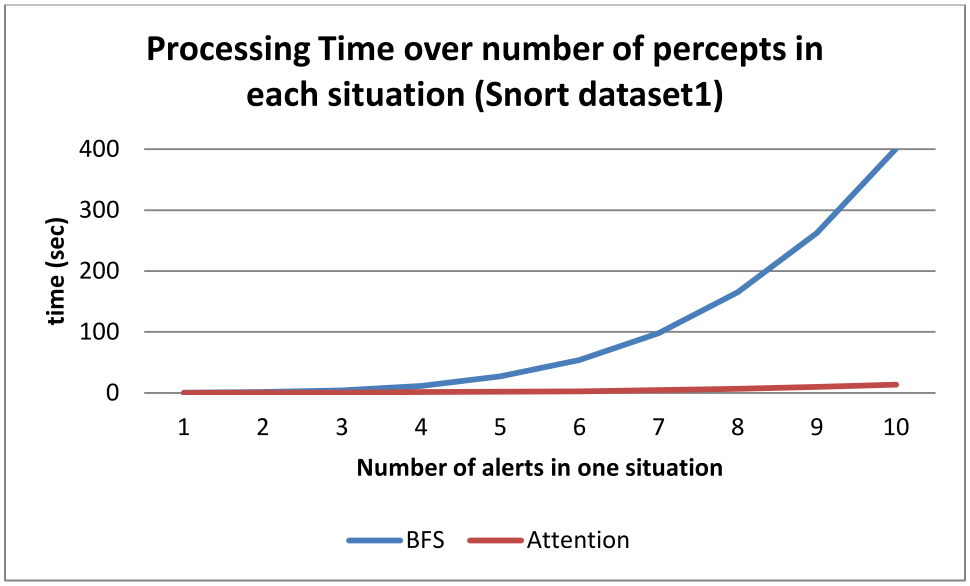


Figure 72. Snort Computation Time over Situation Size. BFS: Greedy Best-first Search.

4. Scalability: Function of Object Constant

In the next level, we evaluate the run time over number of object constants in one situation. As the number of object constant reaches 30 (Figure 73.), the greedy best-first search technique becomes intractable. The Attention model continues to be near linear even up to 250 object constants, as shown in Figure 74. .

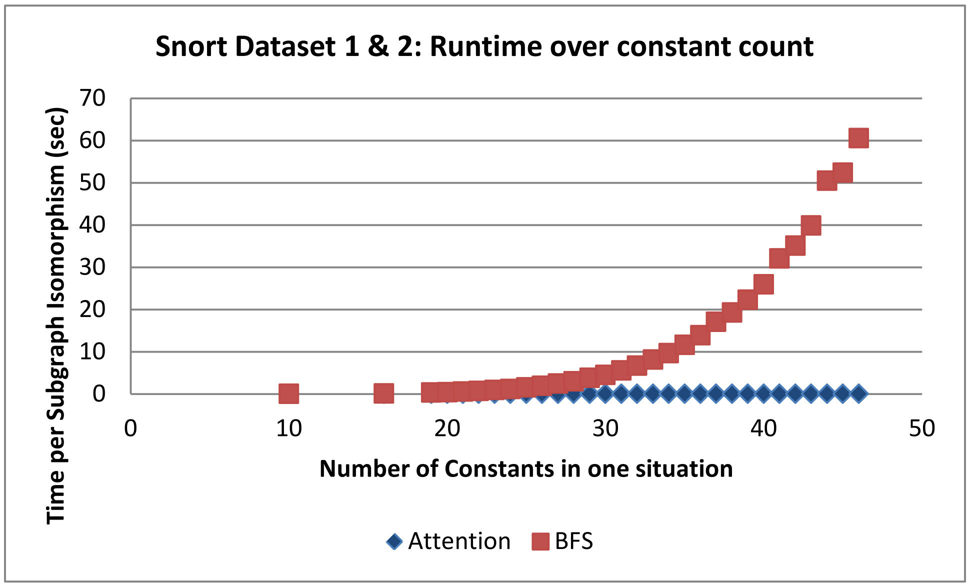


Figure 73. Snort Computation Time over Constant Size.

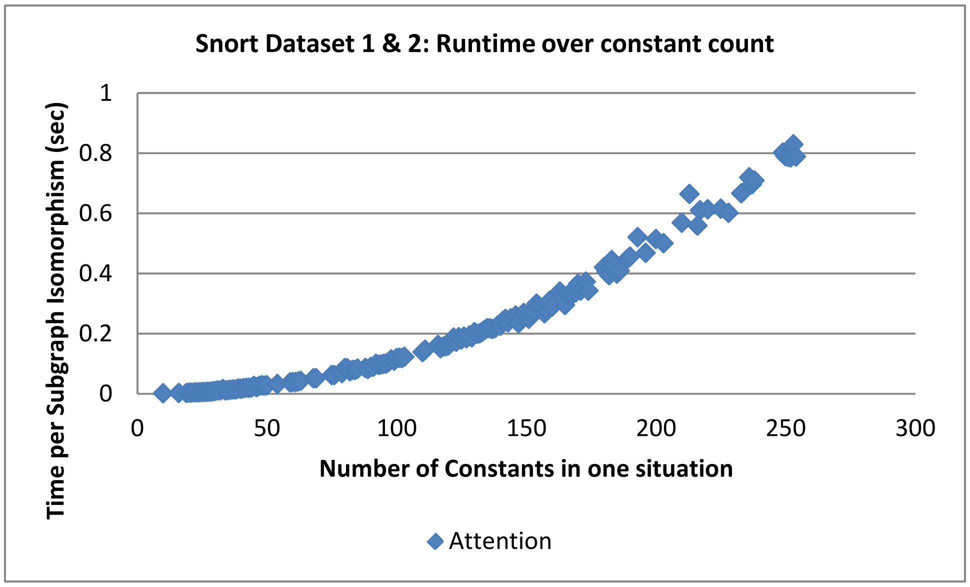


Figure 74. Snort Computation Time over Constant Size, Attention Only.

D. CONCLUSION

In this chapter, we have shown that while Greedy best-first search and attention are incomplete, both are able to achieve similar results as the complete backtrack technique at much faster speed. We have also shown that both best-first search and attention techniques appear to have linear time complexity on short time series. However, when we increase the resolution of investigation, we observed that the greedy best-first search is intractable when the number of alert in one situation is greater than 5, and if the number of constant in a situation reaches 30. The attention technique has consistently remains efficient at near linear time complexity even at constant count of 250. Nevertheless, as shown in the long time series, even the attention technique does not scale well when the number of situations increases. The nest two chapters discuss two methods on how we can improve the scalability of the attention technique.

VIII. EVENT SEGMENTATION

A. INTRODUCTION

Previously, we show that the single-scope blending predictors are desirable because they have significantly better prediction accuracy as compared to other predictors. However, single-scope blending predictors are slow, because the problem of searching for a set of unifications to maximize situation similarity is equivalent to a subgraph isomorphism problem. While the most efficient attention technique scale well in the number of object constants, it does not scale well in the face of long time-series because of the increasing number of situation being added into the lookup table.

One way to improve on the time complexity is to look at the learning aspect of the relational time series. This chapter describes a technique inspired by the event segmentation theory that can potentially improve the time complexity over long time-series with frequent new situation. The next chapter discusses another approach that improves the time complexity by eliminating unpopular and older situation from the lookup tables.

B. EVENT SEGMENTATION THEORY

Event segmentation theory [2][77][78][79][80][81][82] is a theory of how the mind/brain segments ongoing activity into meaningful events. Segmentation simplifies the ongoing activity and treats an interval of time as a single chunk [2]. This chunk is constructed and maintained as a mental representation of the current unfolding event in the working memory. This mental representation provides a basis for predicting how activity will unfold. When situation changes, prediction error increases because the current mental representation is no longer effective in predicting new events. Hence, prediction errors cause an update to the working memory: saving the previous mental representation to the long term memory and construct a new one. At event boundaries, the active memory is cleared and a new event model is from current perceptual information. This process is explained in Figure 75. .

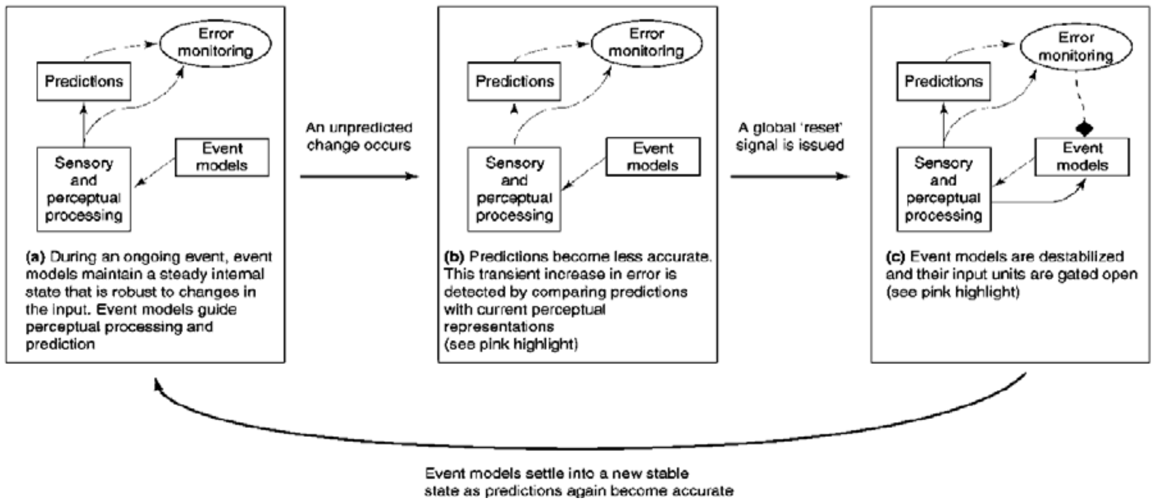

Error monitoring
Predictions
Sensory and perceptual processing
Event models
(a) During an ongoing event, event models maintain a steady internal state that is robust to changes in the input. Event models guide perceptual processing and prediction
An unpredicted change occurs
(b) Predictions become less accurate. This transient increase in error is detected by comparing predictions with current perceptual representations (see pink highlight)
A global 'reset' signal is issued
(c) Event models are destabilized and their input units are gated open (see pink highlight)
Event models settle into a new stable state as predictions again become accurate

Figure 75. Prediction and Event Segmentation Theory [2].

Kurby and Zacks [2] write that good segmentation saves on processing resources and improves comprehension. Good segmentation requires identifying the correct event features. There are two general ways to segmentation. Fine units segmentation (bottom up) [78] is based on salient physical features, distinctive sensory characteristics and movement features. Coarse units Segmentation (top-down) is identified based on abstract features related to goals, social relationships, or personality traits and Knowledge structures.

Maglianoand and Zacks [79] write that event can be identified based on three degrees of continuity (i) Continuous in space, time, and action; (ii) Discontinuous in space or time but continuous in action; and (iii)Discontinuous in action as well as space or time. Swallow et al. [80] also mentioned about identifying events based on features such as Location, Actors, Goals, Objects and Interactions with objects. Reynolds et. al. [77] also suggested that event can be identified based on people, places, things, action changes, temporal, spatial location, and causal sequence. Zacks [78] also suggested new mental models are initiated when there is a change in space, time, protagonist, objects, goals, causes, or character.

Event Model is a working memory representation of “what is happening now” [81]. All perceptual input is processed in the context of an activated Event Model. Event Model contains aspects of a situation that are consistent within an event and allow disambiguation of ambiguous sensory information and Filling-in of missing information. Zacks and Sargent [81] suggest that event model is maintained in lateral prefrontal cortex (PFC). Event model has two parts: Current perceptual information and an event schema (Similar Encountered State). Event schema contains patterns of information learned over a lifetime of experience. It resides in the long-term semantic memory and is implemented by the lateral prefrontal cortex. Schema effects on ongoing perception [82] and provide a framework for incoming information and new information. It also suggests what objects are likely to be present and what steps are likely to be performed, and in which order. It also helps to fill in missing information.

C. MOTIVATION

The current ways of managing the situation-target tuples is to place all situation-target tuples in one lookup table. During prediction, prediction techniques such as statistical lookup table, variable matching and single-scope blending must search through all situation-target tuples in the lookup table when none of the situations in the situation-target tuples matches the current situation.

Event segmentation can provide a hint on how to improve the management of the lookup table to improve search efficiency. The current implementation of having one lookup table for all situation-target tuples can be seen as having all tuples coming from the same segment. Conceptually, we can segment the entire relational time-series by some event features, thereby forming multiple isolated sequences of percepts. Percepts that fall in between two event features will be part of the same segment, identified by the first event feature. We will have one lookup table for one event feature. Segment of the same event features will go into the same lookup table. Examples of event features in Pymud are action, actor, event, and place. Instead of searching one big lookup table, we can search an appropriate smaller lookup table of the same segment event feature as the current situation, thereby, reducing processing time.

D. METHOD OF HIERARCHICAL EVENT SEGMENTATION

In hierarchical event segmentation, we segment the relational time-series by multiple event features. We will describe a way of hierarchical event segmentation by two event features. The first layer of event segmentation is by time. The second layer is by a term in the percept.

1. Percept Sequence Segmentation by a Time Window

The first layer of event segmentation is by time. This is similar to the way by which a situation-target tuple is formed as defined in definition 10, except that the situation is now timed. At this layer, the time-series is not explicitly isolated into multiple time series, but forming multiple segments with overlapping percepts. We effectively convert a percept sequence into a situation-target tuple sequence.

Timed situation: Given a relational time-series $p_1p_2\ldots p_n$ that occurs at time $t_1t_2\ldots t_n$, and a time window t_w for segmentation, a situation is formed by the set of timed percepts $\{H, p_r, p_{r+1},\ldots, p_{r+m}\}$ if $t_r \leq t_{r+1} \leq \ldots \leq t_{r+m}$, $(t_{r+m} - t_r) \leq t_w$ where H is a set of interval timed simplified percepts from $p_1p_2\ldots p_{r-1}$ that has not encountered the corresponding '-' percept, and that $p_r, p_{r+1},\ldots, p_{r+m}$ cannot include contradictory percepts, and the most recent percept will remove earlier contradictory percepts. $p_r, p_{r+1},\ldots, p_{r+m}$ cannot contain percept of type '-' and the corresponding interval percept must be removed. The time of situation is the time of current timed percept.

Timed situation-target tuple: A timed situation-target tuple is defined as $st_i = (s_i, t_i)$ where s_i is a timed situation and t_i is a time percepts that is the next simplified percept of s_i. The percepts t_i is known as target percepts. We will call the (situation, next-percept-target) tuple as timed situation-target in short. The subscript 'i' is ordered by time.

Time-series of timed situation-target tuple: A time-series of situation-target tuple is a sequence of situation-target tuple: $st_1\ st_2\ldots\ st_n$. If a_i is the time of timed percept p_i, the following holds: $a_{i-1} \leq a_i \leq a_{i+1}$.

Segmentation by time: Segmentation by time is the first layer of segmentation that converts a relational time-series into a timed situation-target tuple sequence:

$$\text{Segmentation}_{time}(p_1 p_2 \ldots p_n) = st_1\ st_2 \ldots\ st_n.$$

2. Percept Sequence Segmentation by a Term in Percept

Event feature: Let event feature be $f_e = v_0(v_1, v_2, \ldots, v_n)$ where each variable v_i corresponds to one term c_i in a timed percept $p = c_0(c_1, c_2, \ldots, c_n)$ such that c_i is a value of variable v_i. Each variable has a binary state: on and off. A variable is said to be off if the variable is not used as part of the event feature. We use ϕ to indicate a variable v_i is turned off.

For example, an event feature based on action in Pymud is described as $f_e = \phi(v_1,\phi, \ldots, \phi, v_n=a)$. An event feature based on actor in Pymud is described as $f_e = \phi(v_1=troll|dragon|green\_goblin|red\_goblin,\phi, \ldots, \phi, v_n=a)$. An even feature based on rule ID in Snort alert is described as $f_e = \phi(v_1,\phi, \ldots, \phi, \phi)$.

Segment: A segment in the relational time-series $r = p_1p_2\ldots p_n$ is comprised of the percept subsequence $[p_a p_{a+1} p_{a+2} \ldots p_{a+m} p_b)$ such that p_a is the first percept in the subsequence and one that contains the event feature and p_{a+m} is the last percept in the subsequence before p_b, which mark the start of another segment.

Segment-tag situation and situation-target tuple: We use the event feature f_e as the name of a segment. All timed situation-target-tuples are tagged with a segment name based on the event feature. These situation-target tuples are called Segment-tag situation-target tuple, respectively. When tagged, all timed percepts in the timed situation-target-tuple are converted to simplified percepts through homomorphism.

3. Learning for Event Segmentation

Given a sequence of segment-tagged situation-target tuples $st_1, st_2 \ldots st_m$ with their associated segment tag sequence $t_1, t_2 \ldots t_m$, a new lookup table is created when a new segment tag is encountered. Given a situation-target tuples and its segment tag, we invoke the learning algorithm in section C.1.a in chapter IV using the container C associated with the segment-tag.

4. Prediction for Event Segmentation

Given a set of containers of lookup table associated with a segment tag, a current segment-tagged situation, we invoked the prediction algorithms in section C.1.b in chapter IV with container associated with the segment tag.

5. Example of Event Segmentation

An example of how the segmentation might work is described in Figure 76. . When a percept contains an event feature, that feature becomes the name of the new segment. Given a sequence of incoming percepts, we tag the resultant situation-target tuples by the segment where they belong to. We can store the situation-target tuples in their respective segment lookup table.

Incoming Percepts
*look(agent4,0.0,a)
place(Loc3,0.0,+)
loc(fork74,Loc3,0.0,+)
fork(fork74,0.0,+)
 	
*get(fork74,agent4,2.75,a)
get(agent4,fork74,2.75,e)
loc(fork74, Loc3,2.75,-)
loc(fork74,agent4,2.75,+)
 	
*w(agent4,5.5,A)

Situation	Target	Tag
[]	look(spock8,a)	[]
look(spock8,a)	place(Loc3,+)	look

look(spock8,a)
place(Loc3,+)
	loc(fork74,Loc3,+)	look

look(spock8,a)
place(Loc3,+)
loc(fork74,Loc3,+)
	fork(fork74,+)	look

look(spock8,a)
place(Loc3,+)
loc(fork74,Loc3,+)
fork(fork74,+)
	get(fork74,agent4,a)	look

place(Loc3,+)
loc(fork74,Loc3,+)
fork(fork74,+)
get(fork74,agent4,a)
	get(agent4,fork7,e)	get

place(Loc3,+)
loc(fork74,Loc3,+)
fork(fork74,+)
get(fork74,agent4,a)
get(agent4,fork7,e)
	loc(fork74, Loc3,-)	get

place(Loc3,+)
fork(fork74,+)
get(fork74,agent4,a)
get(agent4,fork7,e)
	loc(fork74,agent4,+)	get

place(Loc3,+)
fork(fork74,+)
get(fork74,agent4,a)
get(agent4,fork7,e)
loc(fork74,agent4,+)
	w(agent4,5.5,A)	get

Figure 76. Example of Event Segmentation by Action.

E. EXPERIMENT ON PYMUD

The goal of the experiment is to see if we can use event segmentation to significantly reduce the computation time.

1. Event Feature

When a percept that contains an event feature arrives, the subsequent situations and the resultant situation-target tuples will be tagged based on the new event feature. The relational time-series of Pymud offer several event features for segmentation: Actor, Action, Event and Place.

A troll actor percept Troll(troll84,0.0,+) contains a feature Troll(ϕ,ϕ,ϕ). Since there are other actors, we have to defined a few event feature such as Dragon(ϕ,ϕ,ϕ), Green_goblin(ϕ,ϕ,ϕ), and Red_goblin(ϕ,ϕ,ϕ). Note that such explicit iteration of actor is not possible in unknown environment. We need other meta information to indicate if a percept is defining an actor.

A look action percept `look(spock84,0.0,a)` and any other action percepts contain the event feature `ϕ(ϕ,ϕ,a)`.

A place percept `place(Paperville3,0.0,+)` and any other place percepts contain the event feature `place(ϕ,ϕ,ϕ)`.

A event percept `get(spock84,pitchfork74,2.75,e)` and any other event percepts contain the event feature `ϕ (ϕ…ϕ, e)`.

2. Experiment Methodology

The experiment in chapter IV is repeated with event segmentation by action, actor, event and place individually. The first experiment studies the effect of event segmentation on short time series. We continue to use 40 batches of 100 percepts. The second experiment studies the effect of event segmentation on longer time series of 10,000 percepts. We repeat the experiment 20 times. We also ran one experiment of a 50,000 percept sequence. The attention predictor is used in this experiment.

3. Results

The prediction performance and time performance for short time-series are described in Figure 77. and Figure 78. , respectively. The prediction performance and time performance for a longer time-series of 10,000 percepts are described in Figure 79. and Figure 80. , respectively. The prediction performance and time performance for a time-series of 50,000 percepts are described in Figure 81. and Figure 82. , respectively.

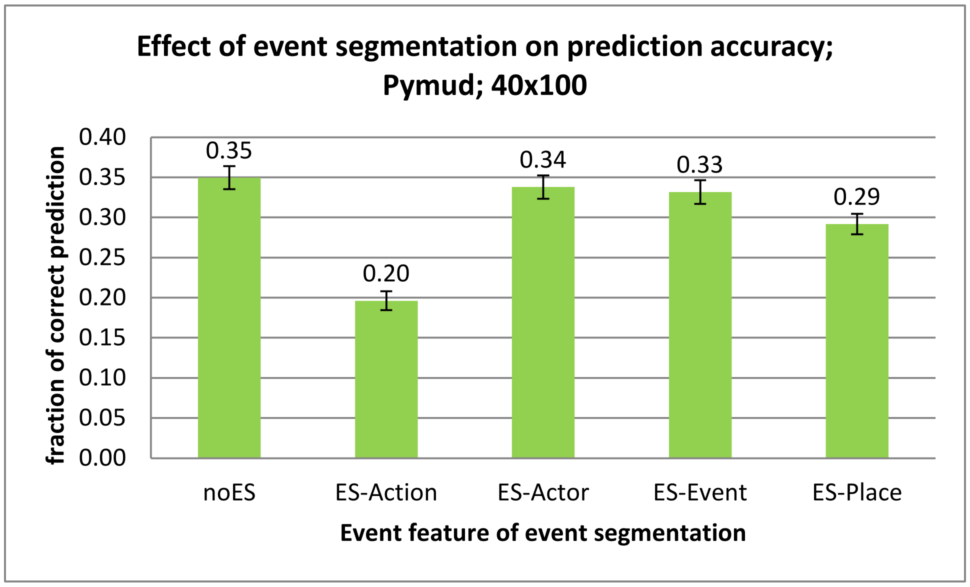


Figure 77. Effect of event segment on Prediction Accuracy on Pymud short time series, 40x100. noEST: no event segmentation. EST Place: Segmentation by Place. EST Action: Segmentation by Action.

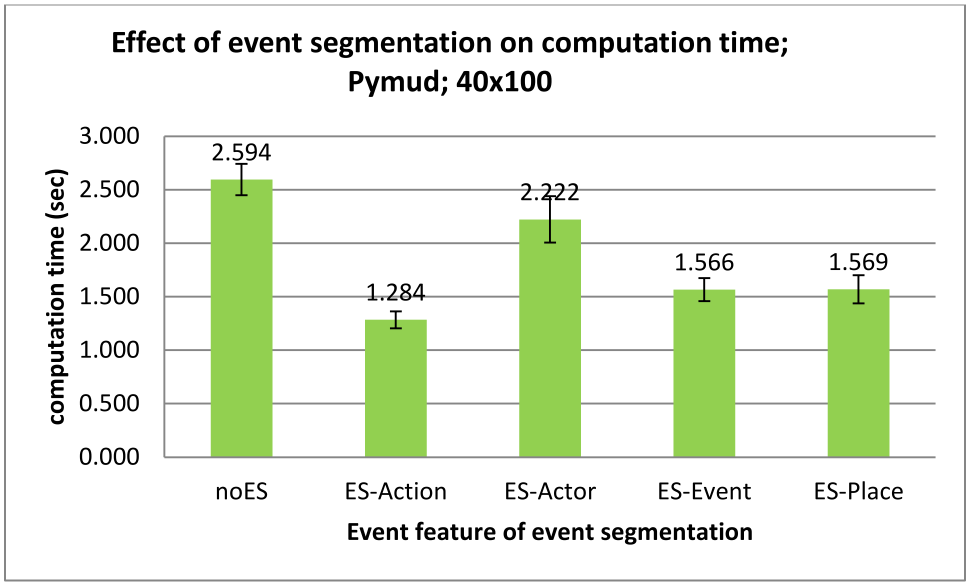


Figure 78. Effect of Event Segmentation on Computation Time on Pymud Short Time Series, 40x100.

The prediction performance and time performance for the longer time-series are described in Figure 79. and Figure 80. , respectively.

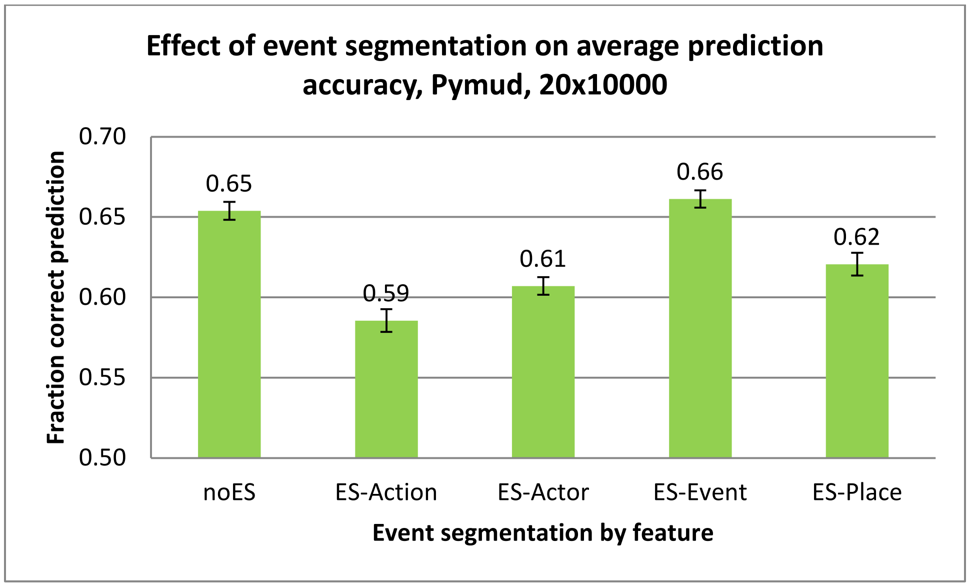


Figure 79. Effect of event segment on Prediction Accuracy on Pymud short time series, 20x10000.

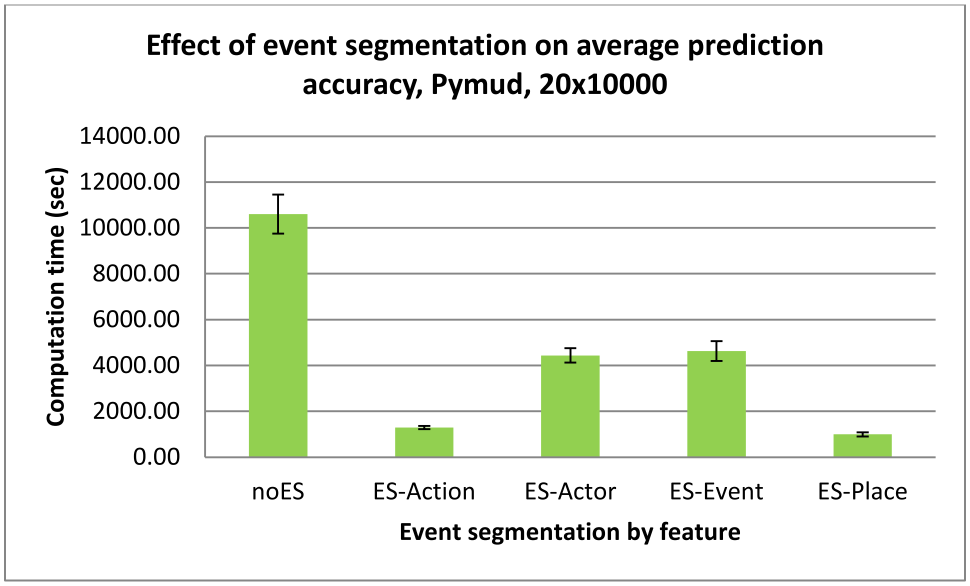


Figure 80. Effect of Event Segmentation on Computation Time on Pymud Short Time Series, 20x10000.

The prediction performance and time performance for the 50,000 time-series are described in Figure 81. and Figure 82. , respectively.

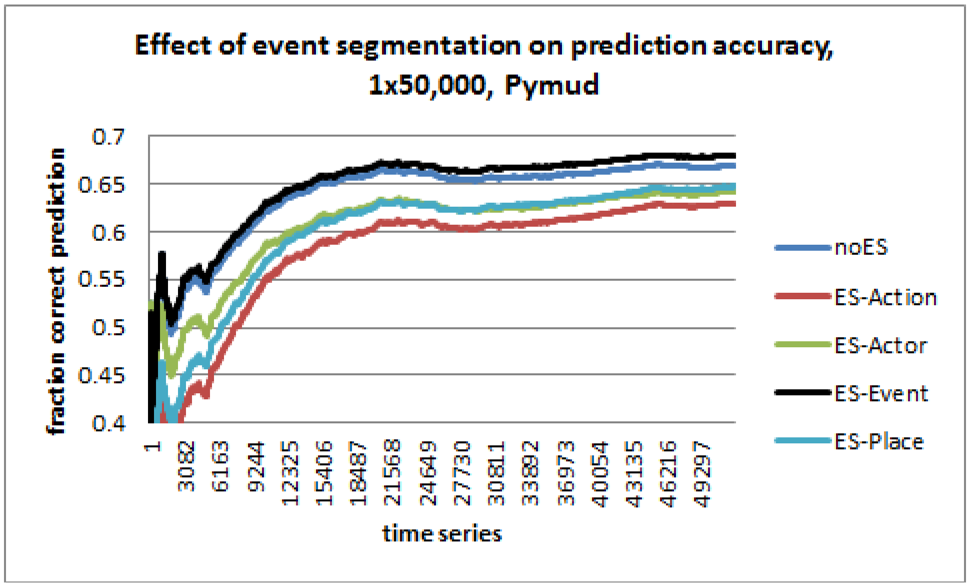


Figure 81. Effect of Event Segment on Prediction Accuracy on Pymud Short Time series, 1x50000.

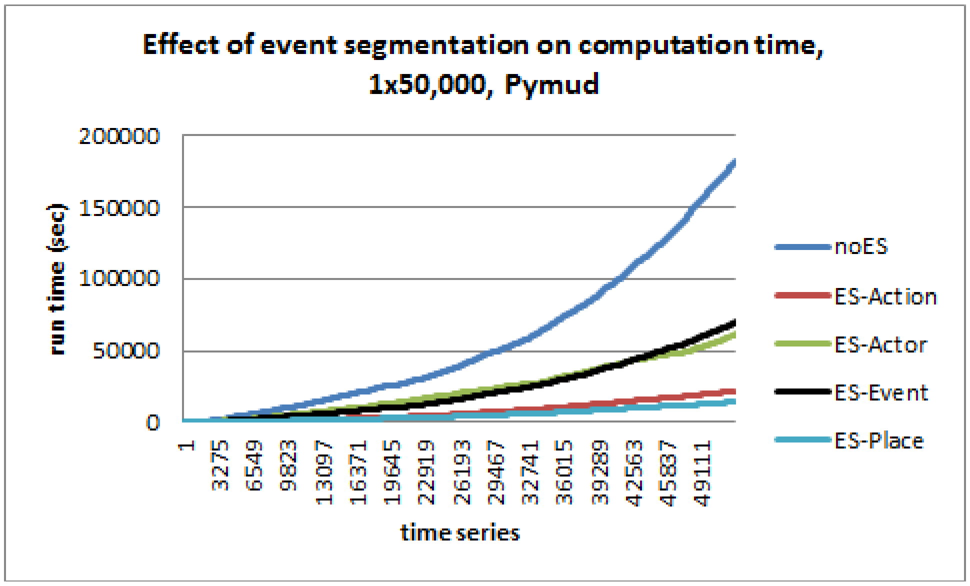


Figure 82. Effect of Event Segmentation on Computation Time on Pymud Short Time Series, 20x10000.

4. Discussion

On short time series, we observe from Figure 77. that, the prediction accuracy of segmentation by actor and event decreased insignificantly while the prediction accuracy of segmentation by action and place deceased significantly. From Figure 78. , the run time of all event segmentation experiment register significant decrease. This result is encouraging because segmentation by actor and event register significant decreased on computation time while achieving similar prediction accuracy.

On longer time series, we observe that the segmentation by event continue to achieve similar prediction accuracy (Figure 79.) but has a much lower computation time (Figure 80.). This result is expanded on Figure 81. and Figure 82. that segmentation by event appear to improve on prediction accuracy over a longer time with the run time growing much slowly.

Choosing an appropriate feature for event segmentation is therefore crucial.

F. EXPERIMENT ON INTRUSION ALERTS

A second experiment studied the effect of event segmentation on network intrusion detection alerts prediction.

1. Event Feature

The relational time-series of Snort alerts also allows several event features such as rule ID and protocol. There are more features if we consider more data from the Snort alert. In this experiment, we will only look at rule ID and protocol.

Note that segmentation by ID will not degenerate into a first order Markov model when the ID is constantly changing. Recall that in the hierarchical segmentation, we segment by time first followed by an event feature. We effectively convert a sequence of alerts into a sequence of situation-target tuples. The second layer of segmentation provides a tag in each situation-target tuple. Each situation will still contain a set of alerts. A constantly changing ID means that the tag for the situation-target tuple keeps changing.

2. Experiment Methodology

The experiment in chapter V was repeated with event segmentation added. We have 16 replications of 1000 alerts each and one run on the entire 16,000 alerts.

3. Results

The prediction accuracy and time performance for 16x1000 are given in Figure 83. and Figure 84. , respectively. The instantaneous results for prediction accuracy and time performance are given in Figure 85. and Figure 86. , respectively. The instantaneous results for prediction accuracy and time performance of 1x16101 are given in Figure 87. and Figure 88. , respectively.

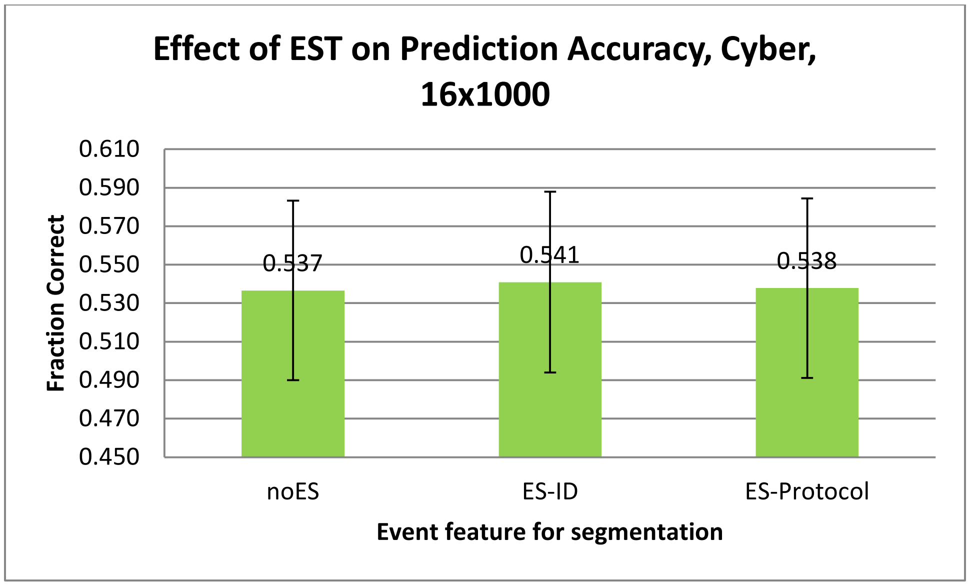


Figure 83. Effect of Event Segmentation on Prediction Accuracy, Cyber, 16x1000.

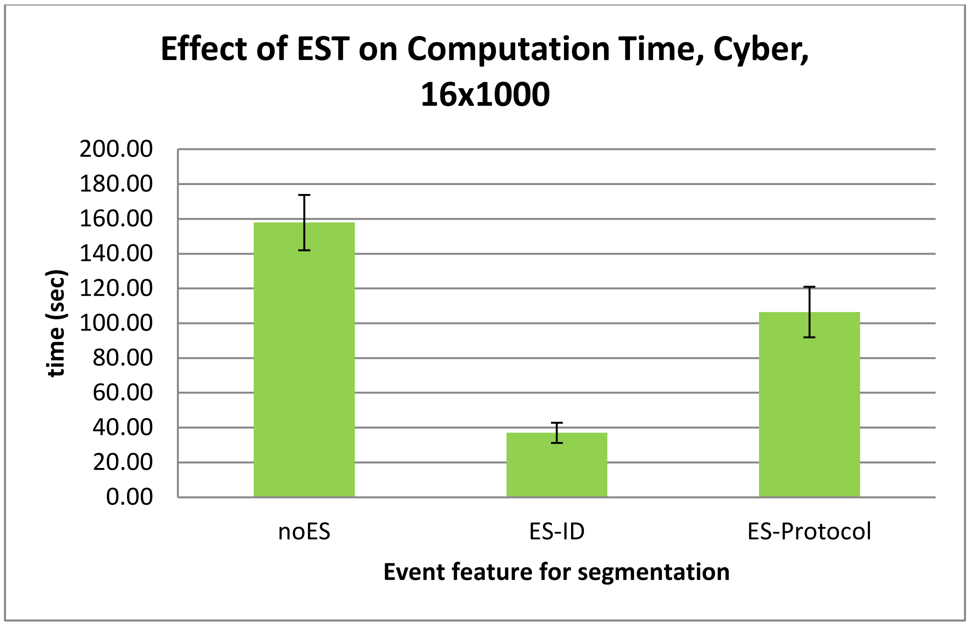


Figure 84. Effect of Event Segmentation on Computation Time, Cyber, 16x1000.

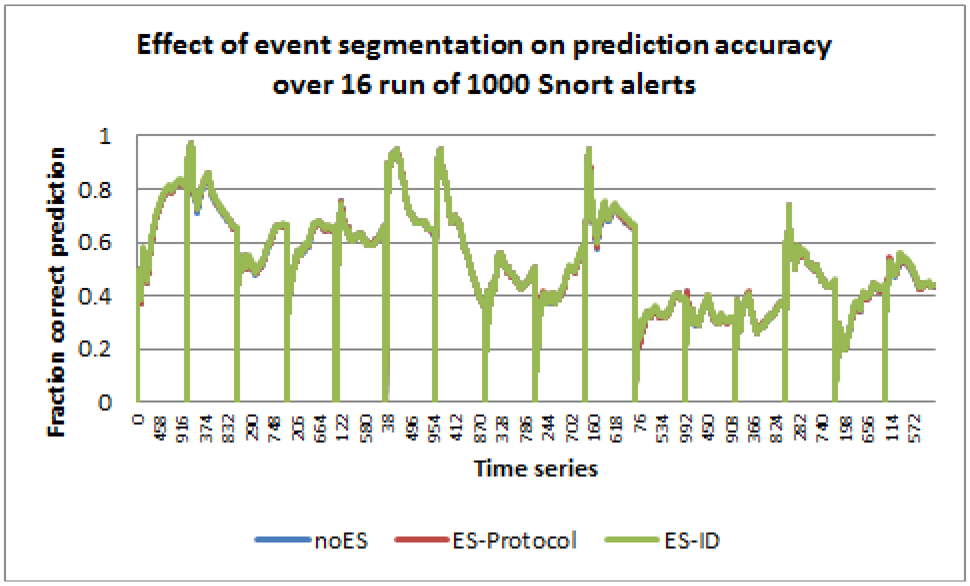


Figure 85. Instantaneous Result of the Effect of Event Segmentation on Prediction Accuracy, 16x1000, cyber.

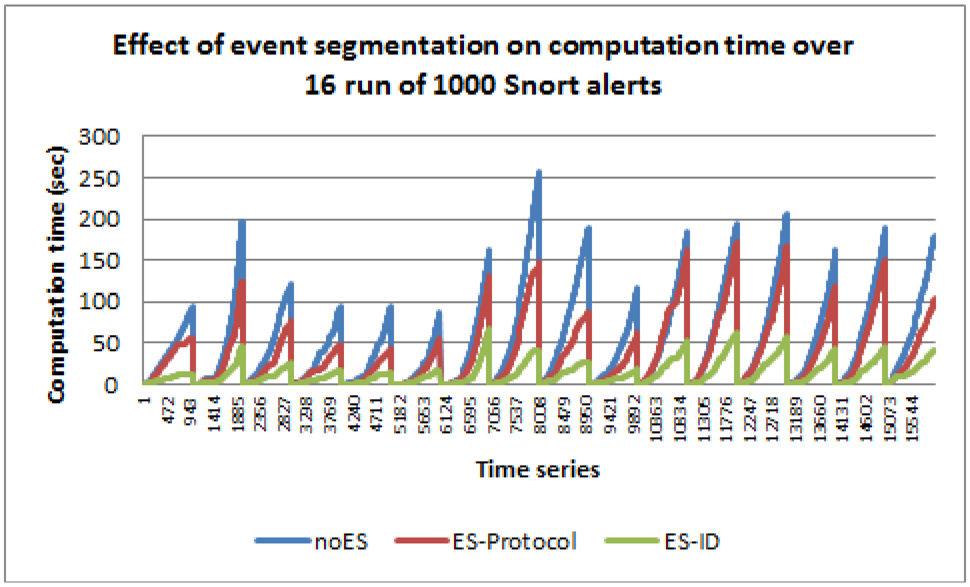


Figure 86. Instantaneous Result of the Effect of Event Segmentation on Computation time, 16x1000, cyber.

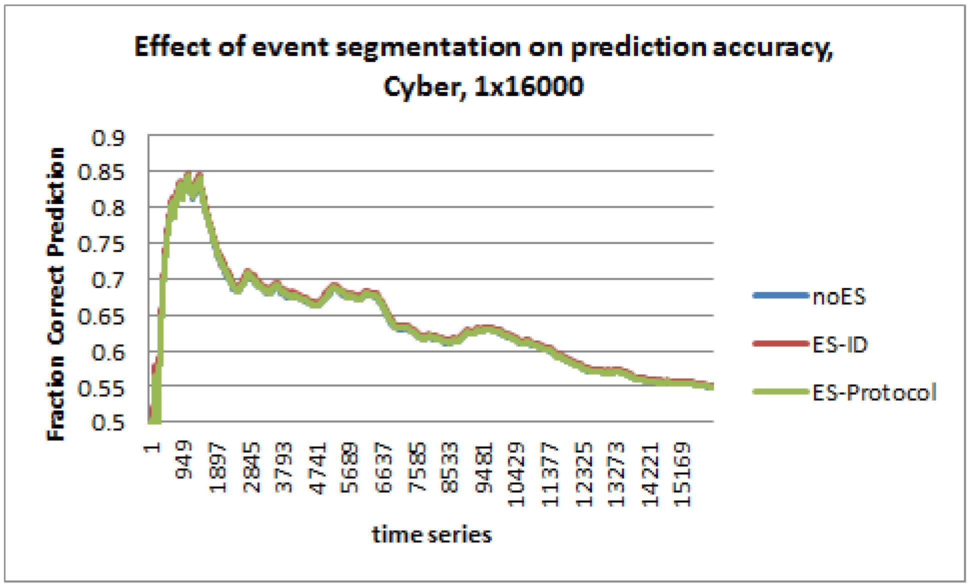


Figure 87. Effect of Event Segmentation on Prediction accuracy, Cyber Dataset 1 and 2, 1x 16000.

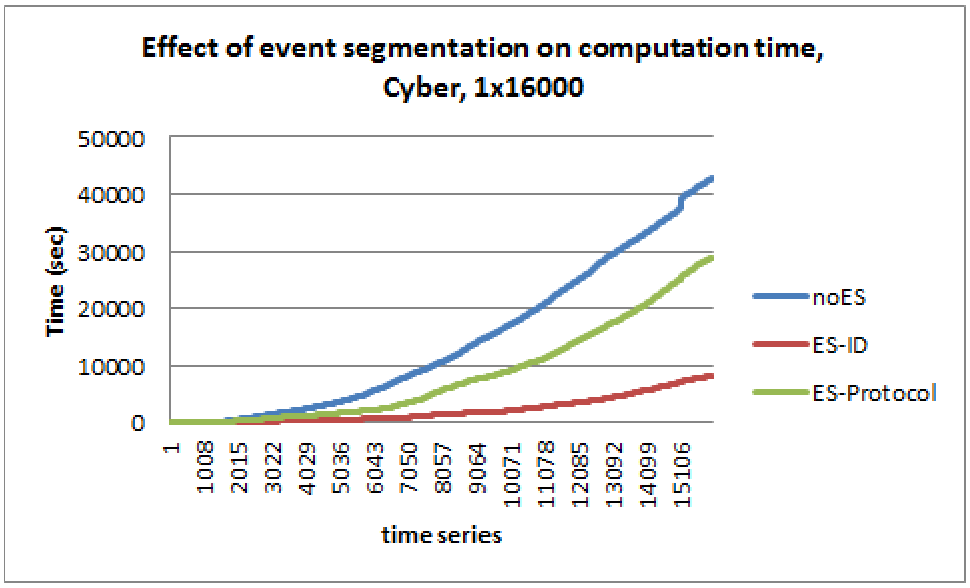


Figure 88. Effect of Event Segmentation on Computation Time, Cyber Dataset 1 and 2, 1x 16000.

4. Discussion

We observe that the prediction accuracies are similar for event segmentation and for no event segmentation. Event segmentation by Snort ID achieves the most saving on computation time. From the instantaneous results, we can see the computation time for segmentation by ID increase much slower than the other two. The time performance for protocol, though is significantly faster than the noES, is poorer than the ID because there are only three type of protocol: TCP, UDP and ICMP. There are only three lookup tables while the ID one has a lot more because of the variety of ID.

G. CONCLUSION

In this chapter, we show that we can use the event segmentation to improve the current state of the art in relational time-series learning and prediction. We show that event segmentation by event and rule ID can help to reduce processing time for Pymud and cyber domain, respectively.

For future works, there are many possible way of improving the event segmentation. Kurby and Zacks [2] mentioned that the percept stream can be segmented by multiple features and be arranged in hierarchical order. Our current implementation is a two layer system in which, the top layer is segment by a feature and the lower layer is segment by a time window. A more complicated hierarchical system can be introduced that account for more features, such as place and actor. In addition, the current mean of combination is to switch when the accuracy of event segmentation surpasses the one without event segmentation. One possible improvement is the switch at the point with the difference in prediction accuracy begins to narrow.

THIS PAGE INTENTIONALLY LEFT BLANK

IX. SITUATION ELIMINATION

A. INTRODUCTION

The situation learning approach learns a set of situation-target tuples from the relational time series. The number of situation-target tuples increases over time whenever new situations are encountered. Figure 89. shows that the cumulative prediction time increases exponentially (top chart) when the number of situation increases (bottom chart) linearly. The number of situations is the number of entries in the lookup table.

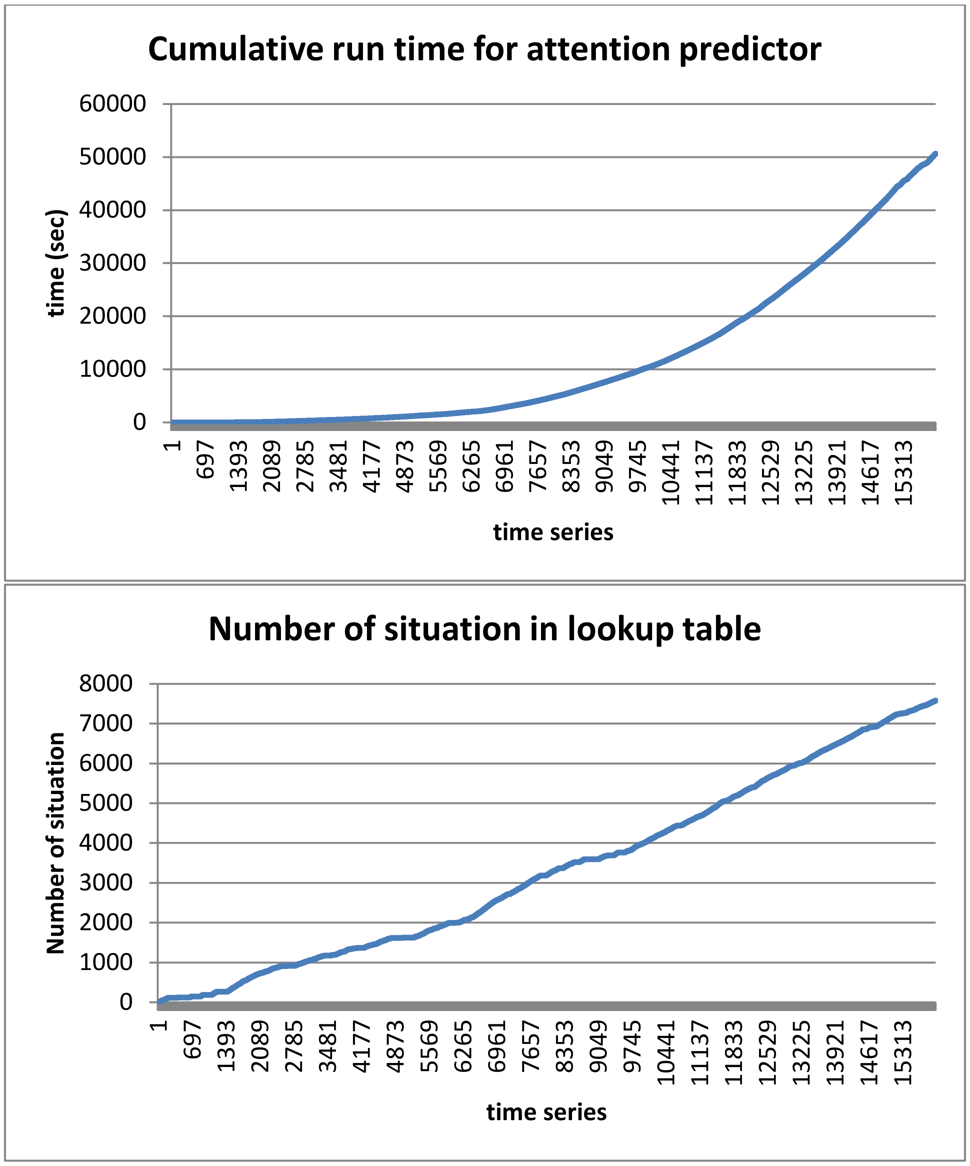


Figure 89. Effect of Increasing Number of Situation on the Run Time.

In the lookup table, not all previous situations are useful for prediction. Some situations are there because of noise. Some situations might be too old to be useful. Furthermore, in a relational time series, there is a notion of moving context. Situations encountered a long time ago may not be relevant in the current context. These un-useful situations could require unnecessary computation time. The objective in this chapter is to find a changing minimal set of situations to be kept in the lookup table that corresponds to the unknown moving context such that the prediction accuracy will not suffer significantly but will significantly improves processing time.

In the next few sections, we will first discuss how to rank the situations in the lookup table. When the situations are ranked, we can eliminate the situation that has the least ranking. Next we will discuss several ways to determine when to eliminate situation of the lowest rank.

B. SITUATION RANKING

To find a minimal set of relevant situations in the lookup table, we have to eliminate the less relevant ones. We need a mean to sort the situation by relevancy so that we can eliminate situation with low level of relevancy. We can define relevancy by count or time or both.

1. Count

In the count method, the situations are ranked by their number of occurrence. Situations of higher count have a higher probability of occurring, and may therefore be more valuable, since a situation that rarely occurs may just be a noise. A tie can be broken by using the time. There is a problem that newer situations that occur recently may get eliminated because they are among the lowest count.

2. Time

In the time method, the situations are ranked by their time of occurrence, or update. A situation that occurred recently is more valuable than one that occurred a long time ago. A tie can be broken by using the count. There is a problem that the oldest situation but high occurrence situation may get eliminated

3. Count and Time

Each situation has two scores, one from time-rank and the other from count-rank. These two scores are averaged to form the combined ranking. Situations that have high ranking mean that they occur frequently and recently.

The combined method is used to generate a relevancy ranking for each situation.

C. ELIMINATION TECHNIQUES

When the situations are ranked, we can eliminate the situations that have the least rankings. We will now discuss several ways on how to eliminate those low-rank situations.

1. Fixing Memory Size

The simplest way to eliminate situations is to set a threshold or maximum memory size, which is the maximum number of situation we want in our table. When the maximum memory size is reached after we added a new situation, we will eliminate the one with the lowest rank. We fixed the memory size by a fraction of the length of each time series.

2 Fixing a Fraction of the Cumulative Memory Size of No Elimination

Instead of fixing the maximum memory size on some constant figure, it might be better to use online information to decide whether we should increase or decrease the maximum memory size. Hence, the maximum memory size can be based on a fraction of the cumulative memory size of the original memory size when no situation elimination is used. For example, we can fix the memory size based on 10%, 20%,…, 100% of the running number of situation of no elimination.

3. Consecutive Success (Bit)

We can also vary the maximum memory size based on the past few predictions performances. If we record one correct prediction as 1, and incorrect prediction as 0, we have a bit string [b_1, b_2, …, b_n] that describes the prediction results, where n is the number of prediction events so far. A new memory size learning method can be based on the number of past consecutive success. If we can achieve a fixed number of consecutive successes, we will lower the maximum memory capacity. Let a bit length define the number of consecutive success required. Let the number of past consecutive success be c. If c == bit length, decrease memory size.

We introduce bit length learning (BitL), a mode where we attempt to learn a bit length instead of fixing it. Bit length learning is described in following algorithms. Essentially, if we have successes, decrease memory size, increase bit length.

Algorithm 10: Consecutive success

```
Bit_Length ← 1
At each prediction event:
     If Bit_Length == c:
          Bit_Length = Bit_Length + 1
     else:
          Bit_Length =  max(1, Bit_Length -1)
```

4. Fraction Learning

Instead of fixing a fraction of the original number of situation, we can let the fraction vary according to the prediction performance. If we are getting good performance, we can reduce the factor. We use the consecutive method to determine when to reduce or increase. For example, if we have c consecutive correct prediction, we reduce the factor by a rate r. The algorithm is given below:

Algorithm 11: Factor Learning

```
If Bit_Length == c:
     Factor f =  max(0, f – r)
Else
     Factor f =  min(1, f + r)
```

5. Gradient of Past Performance

A measure of efficiency can be computed by dividing the current prediction accuracy by the number of situation: $e = a/s$ where e is the efficiency, a is the average accuracy after each prediction event, and s is the number of situation after that prediction event. Suppose that e_n is the current efficiency, e_{n-m} is the efficiency of the prediction event occurred m events before n. A gradient g can be computed between e_n and e_{n-m}: $g(m) = e_n - e_{n-m}$. If g is negative, we can reduce memory size.

D. EXPERIMENTAL SETUP

We test the elimination techniques on two domains: intrusion-alert prediction and Pymud percept prediction. For intrusion-alert prediction, we used the two alert sequences (dataset 1 and dataset 2) described in chapter 5. Dataset I and 2 are combined to form a single dataset. These two dataset contains more than 16,000 of alerts, over 3.5 months of network intrusion alerts. The alerts are then broken down into 16 series of 1,000 alerts each, numbered 1,2,…,16. A second experiment was run based on 8x2000 percepts. This is to test the robustness of the results determined by the first experiment.

For Pymud, we run the experiments on the relational percept sequence obtained from Pymud as described in chapter 4.

E. RESULTS AND DISCUSSION

1. Fixing Memory Size

The prediction accuracy and situation count as a function of fixed memory size are as shown in Figure 90. and Figure 91. . The statistical method of the “student group t-test” shows that maintaining a memory size of 100 is enough to have similar accuracy for 1000 long alert sequence. The next set of results (Figure 92. , Figure 93.) is obtained from the 8x2000 experiment. Statistical student group t-test shows that maintaining a memory size of 200 is enough to have similar accuracy for 2000 long alert sequence. From these two tests, we know that there exists a solution that has fewer situation counts but similar accuracy.

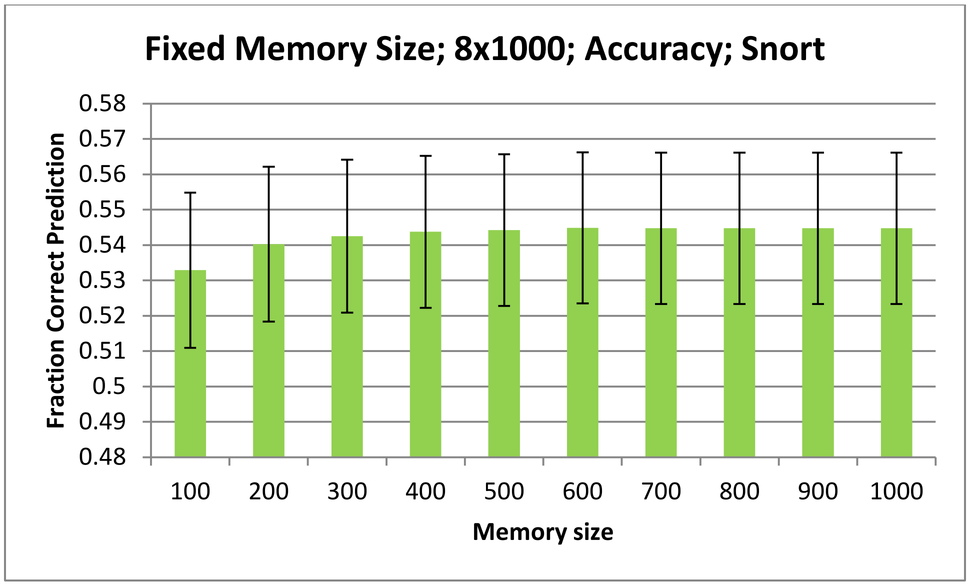


Figure 90. Effect of Fixing Memory Size on Snort Alert Prediction, 8x1000.

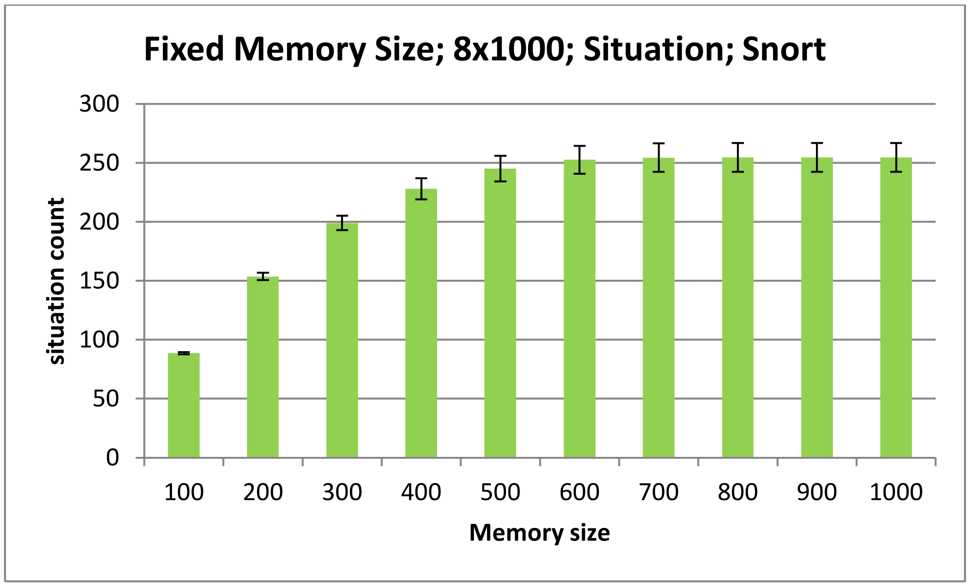


Figure 91. Effect of Fixing Memory Size on Situation Count, 8x1000.

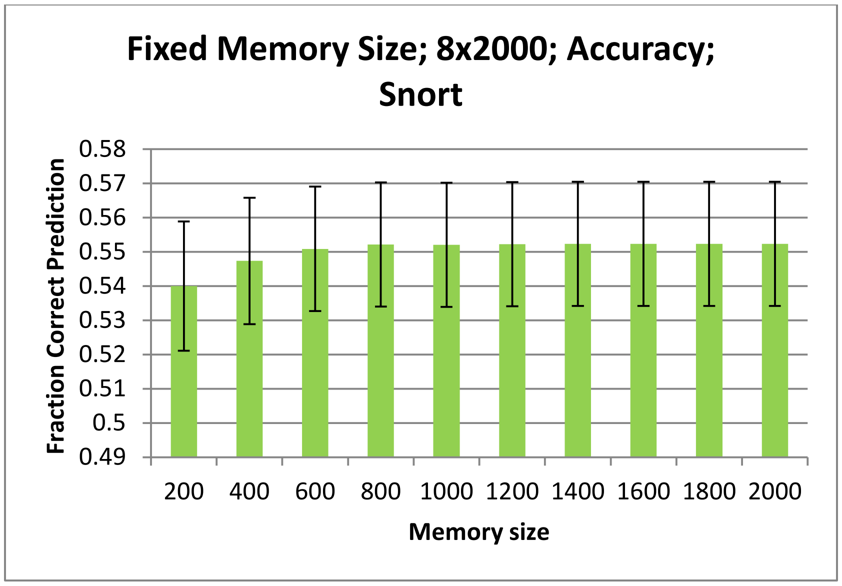


Figure 92. Effect of Fixing Memory Size on Snort Alert Prediction, 8x2000.

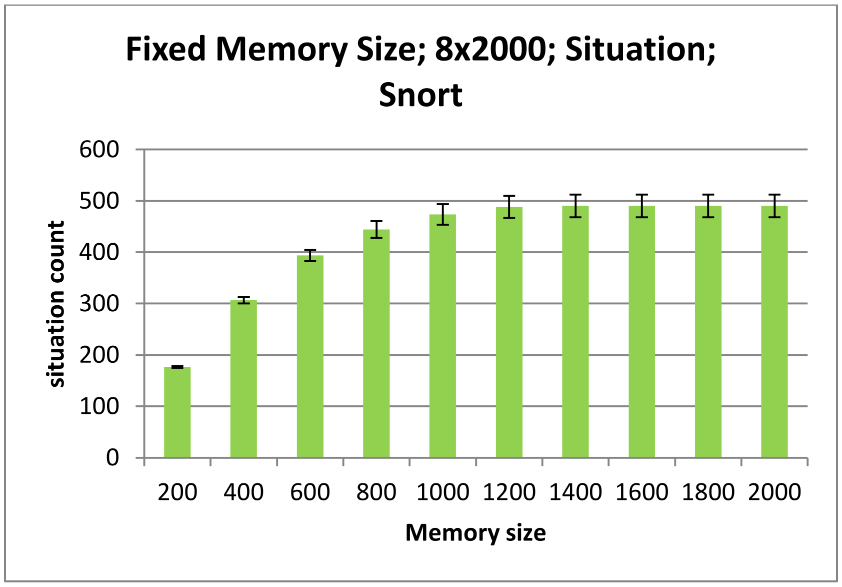


Figure 93. Effect of Fixing Memory Size on Situation Count, 8x2000.

2. Fixing a Fraction of the Cumulative Memory Size

The prediction accuracy and situation count as a function of fixing the memory size at a fraction of the original un-eliminated memory size for 8x1000 are listed in Figure 94. and Figure 95. , respectively. The student group t-test shows that maintaining a memory size of 10% of the original memory is enough to have similar accuracy for

1000 long alert sequence. The results for 8x2000 are listed Figure 96. and Figure 97. . Statistical student group t-test also shows that maintaining a memory size of 10% of the original memory is enough to have similar accuracy for a 2000 alert sequence.

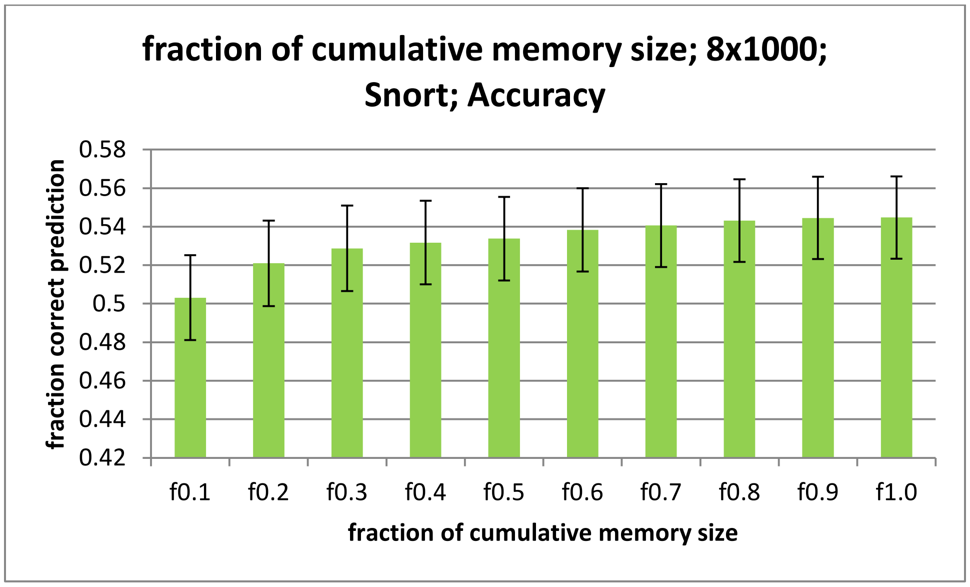


Figure 94. Effect of Fixing a Fraction of Memory Size on Snort Alert Prediction, 8x1000. F0.1 Means 10% of the Original Memory Count.

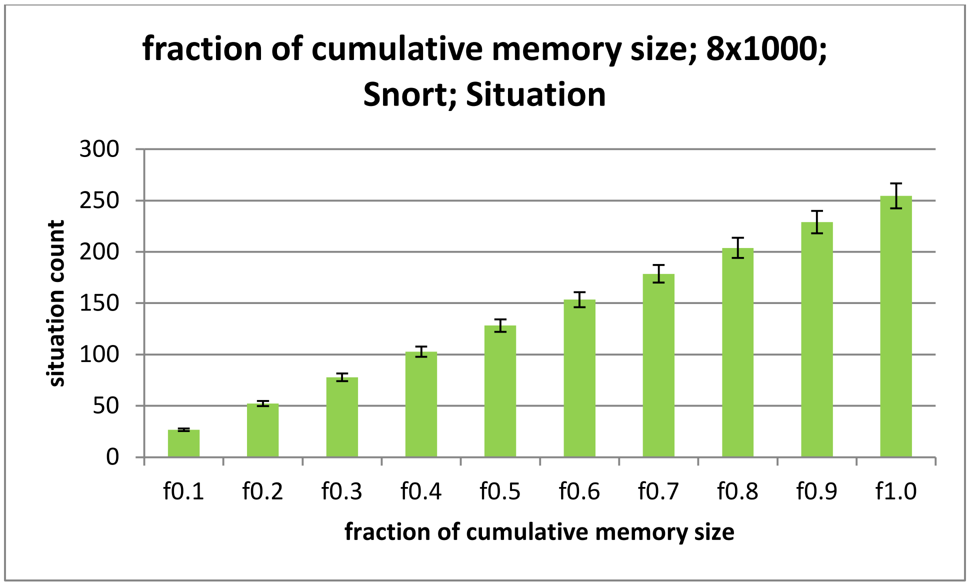


Figure 95. Effect of Fixing a fraction of Memory Size on Situation Count, 8x1000. F0.1 Means 10% of the Original Memory Count.

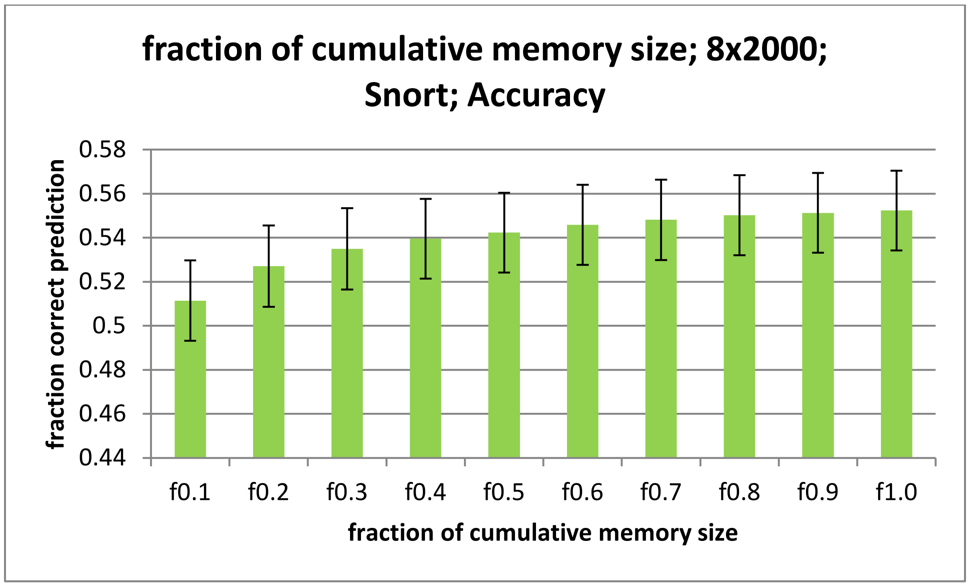


Figure 96. Effect of Fixing a factor of memory size on Snort Alert Prediction, 8x2000. F0.1 means 10% of the original memory count

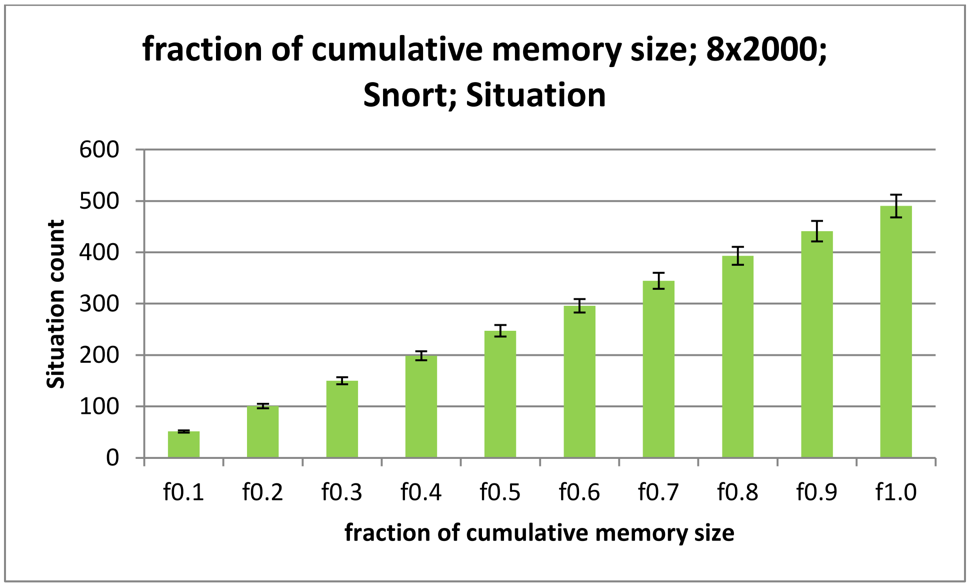


Figure 97. Effect of Fixing a Factor of Memory Size on Situation Count, 8x1000. F0.1 Means 10% of the Original Memory Count.

3. Consecutive Success

The prediction accuracy and situation count as a function of number of consecutive success to eliminate situation for 8x1000 are listed in Figure 98. and Figure 99. while the results for 8x2000 are listed in Figure 100. and Figure 101. . The student group t-test shows reducing the memory size based on the previous prediction outcome is enough to have similar accuracy for both 1000 and 2000 long alert sequence. The bit learning method appears to achieve the highest number of saving on situation count while maintaining similar accuracy.

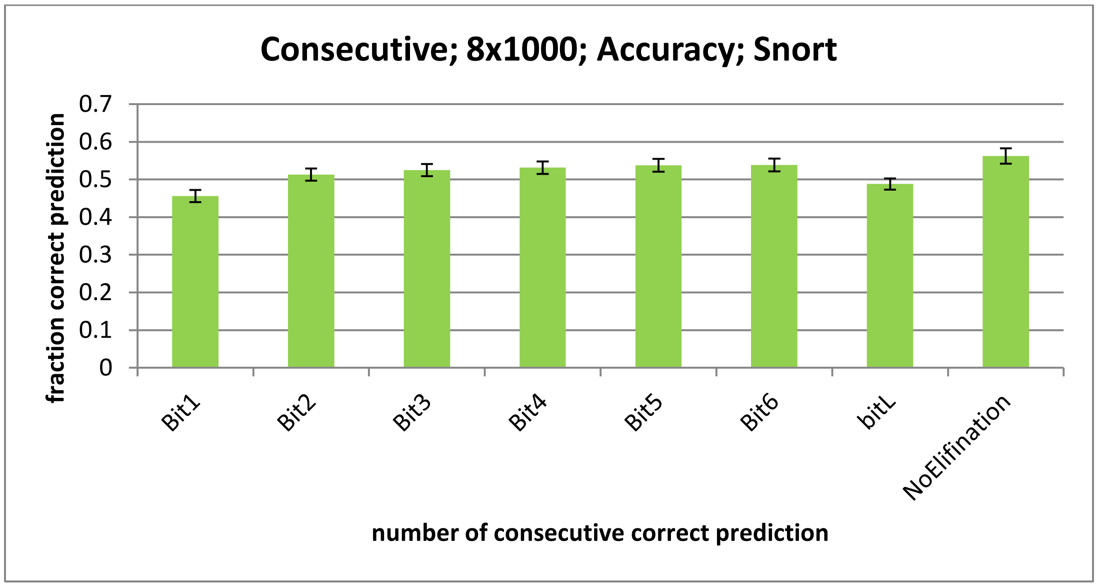


Figure 98. Effect of Consecutive success on Snort Alert Prediction, 8x1000. BitX Means X Consecutive Correct Prediction. BitL Is Variation of Consecutive Success Requirement.

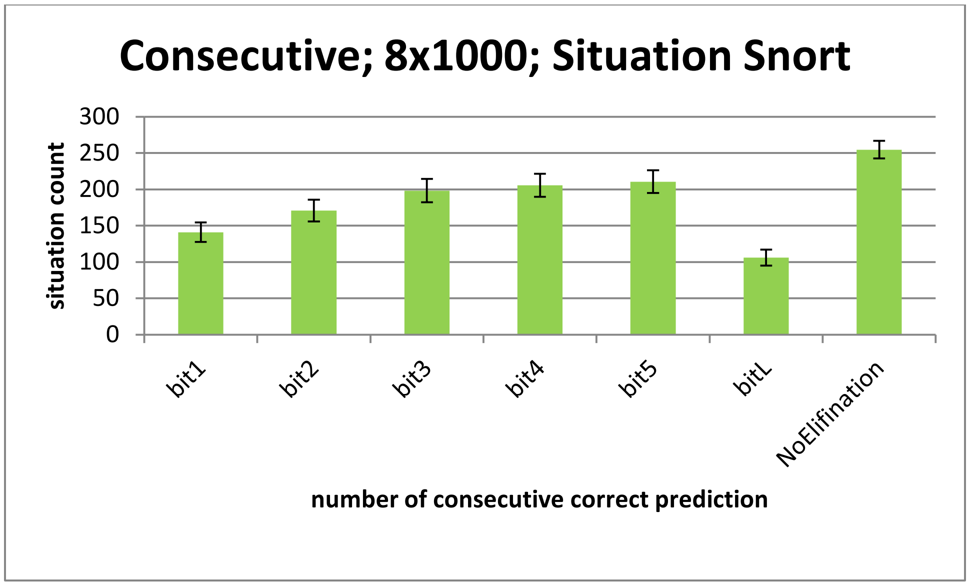


Figure 99. Effect of Consecutive Success on Situation Count, 8x1000. BitX Means X Consecutive Correct Prediction. BitL is Variation of Consecutive Success Requirement.

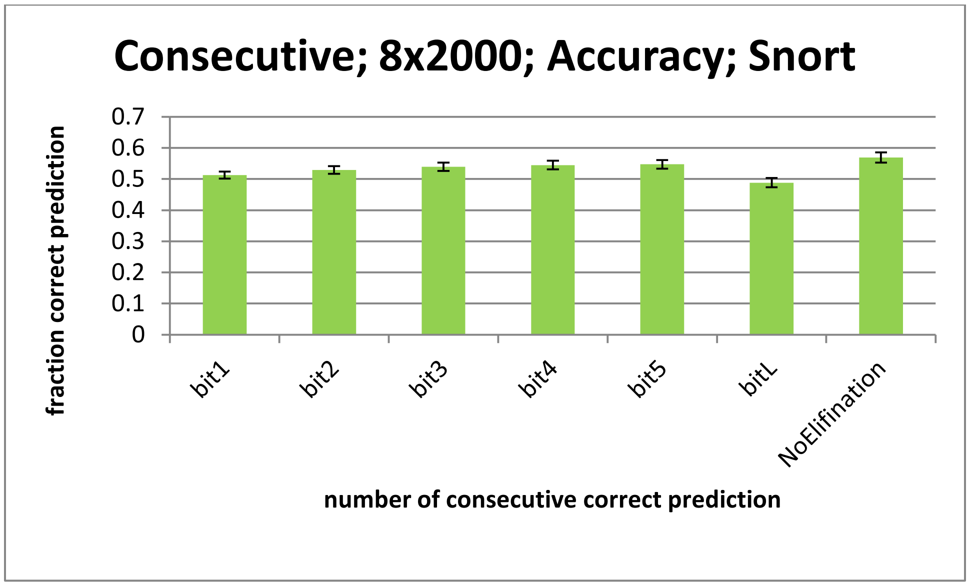


Figure 100. Effect of Consecutive success on Snort Alert Prediction, 8x2000. BitX Means X Consecutive Correct Prediction. BitL is Variation of Consecutive Success Requirement.

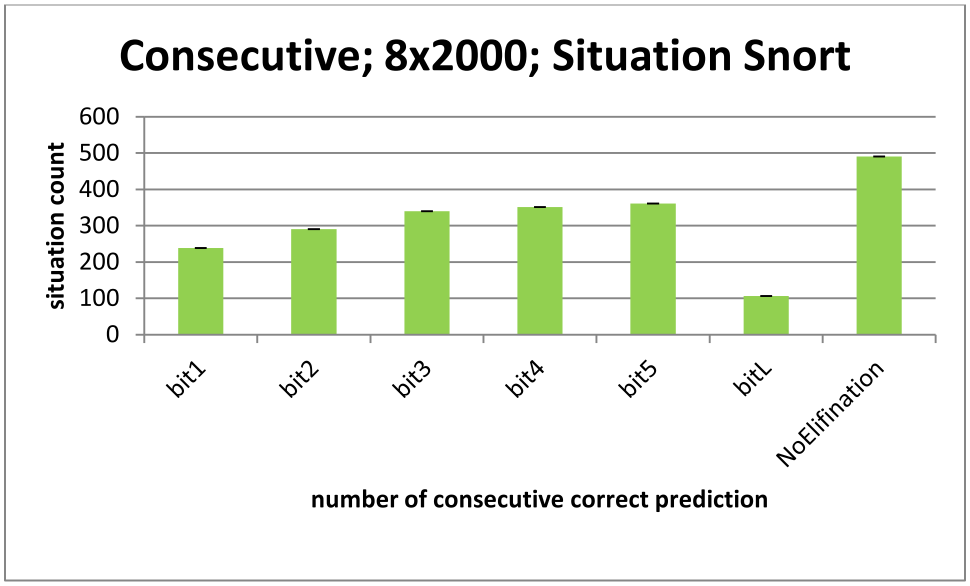


Figure 101. Effect of Consecutive success on Situation Count, 8x2000. BitX Means X Consecutive Correct Prediction. BitL Is Variation of Consecutive Success Requirement.

4. Fraction Learning

In this experiment, we varied the rate at which the fraction of the original situation count is increased or decreased. 1 consecutive success is used to reduce the fraction by the rate. The prediction accuracy and situation count for 8x1000 are listed in Figure 102. and Figure 103. while the results for 8x2000 are listed in Figure 104. and Figure 105. . The student group t-test shows varying the fraction of original memory by the rate of 0.05 is able to reduce the situation count significantly while having similar accuracy for both 1000 and 2000-alert sequence.

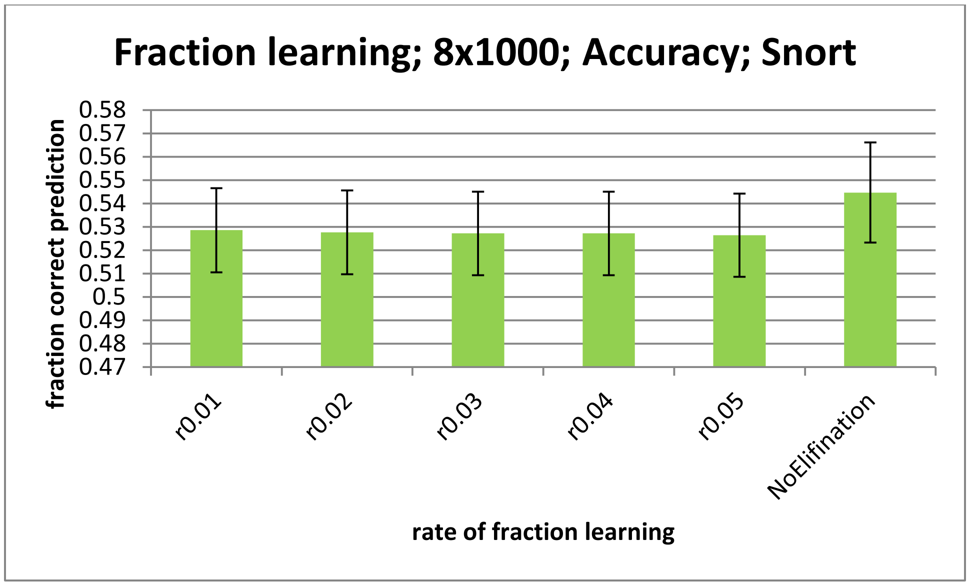


Figure 102. Effect of Factor Learning on Snort Alert Prediction, 8x1000. R Is the Rate of Learning.

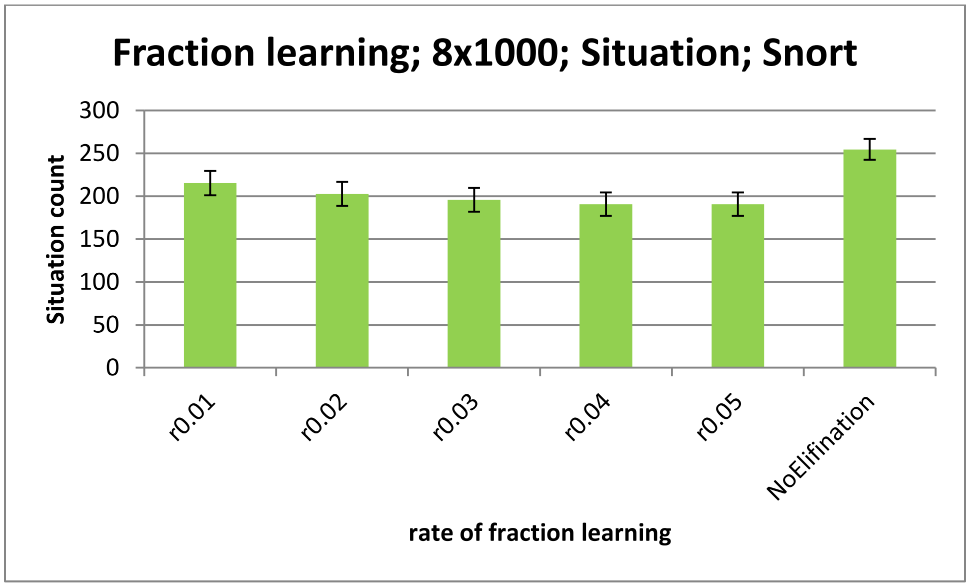


Figure 103. Effect of Factor Learning on Snort Alert Situation Count, 8x1000. R Is the Rate of Learning.

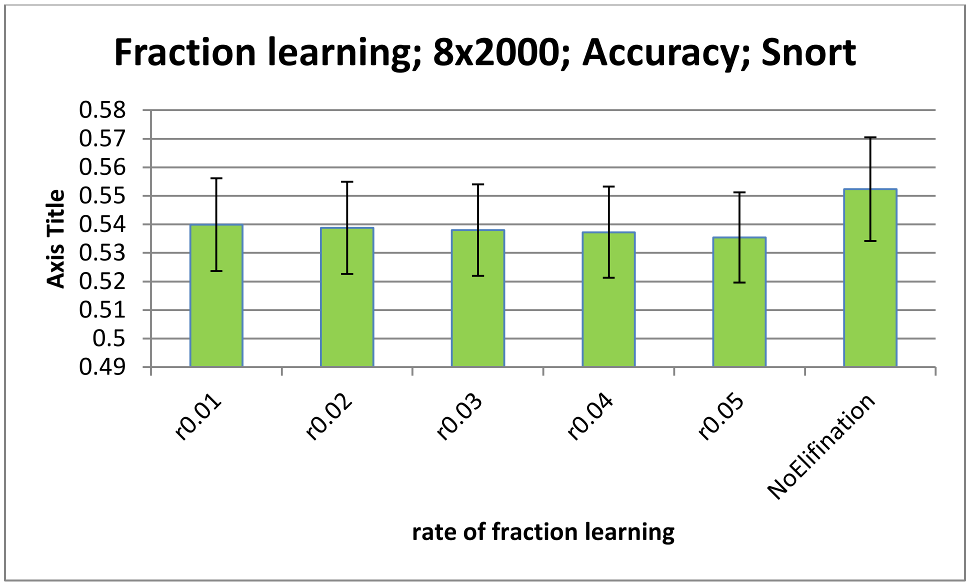


Figure 104. Effect of Factor Learning on Snort Alert Prediction, 8x2000. R Is the Rate of Learning.

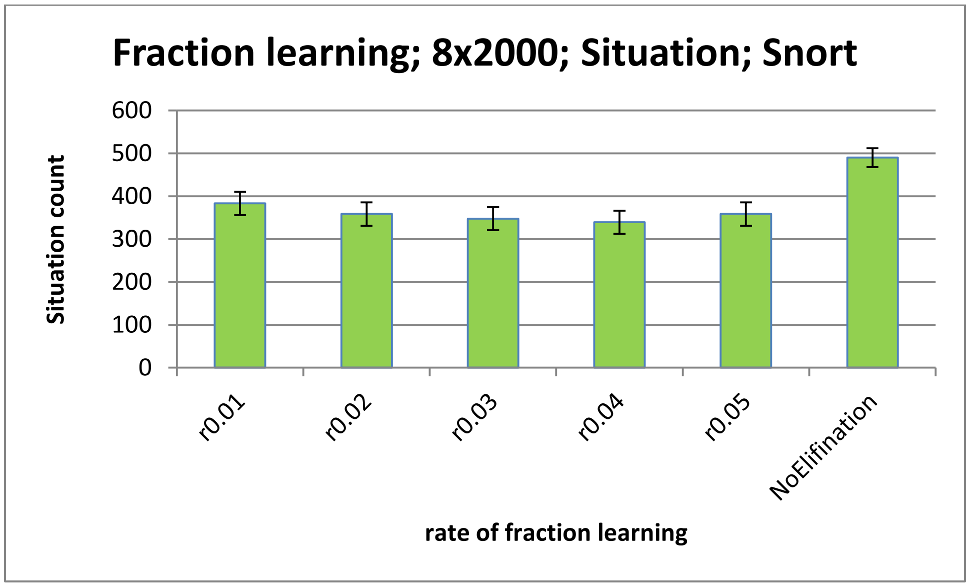


Figure 105. Effect of Factor Learning on Snort Alert Situation Count, 8x2000. R is the rate of learning

5. Learning Memory Size: Based on Gradient of Past Performance

In this experiment, we varied the length of the gradient from the current prediction event to the previous one prediction event (g2), two prediction event (g3), … nine prediction event g(10). gInf is the gradient computed from the current prediction event to the first prediction event. gSuccess is the gradient computed from the current prediction event to the past few prediction event where there is consecutive prediction success. The results are given in Figure 106. and Figure 107. for 1000 long sequence and Figure 108. and Figure 109. for 2000 long sequence. The student group t-test shows that gInf and gSuccess are able to reduce the situation count significantly while having similar accuracy for both 1000 and 2000 long alert sequence.

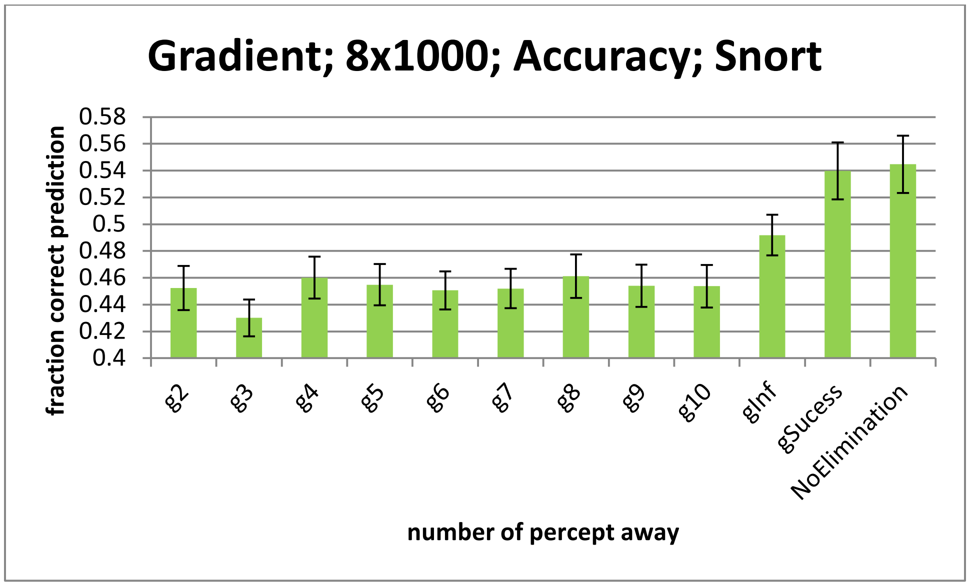


Figure 106. Effect of Gradient Difference on Snort Alert Prediction, 8x1000.

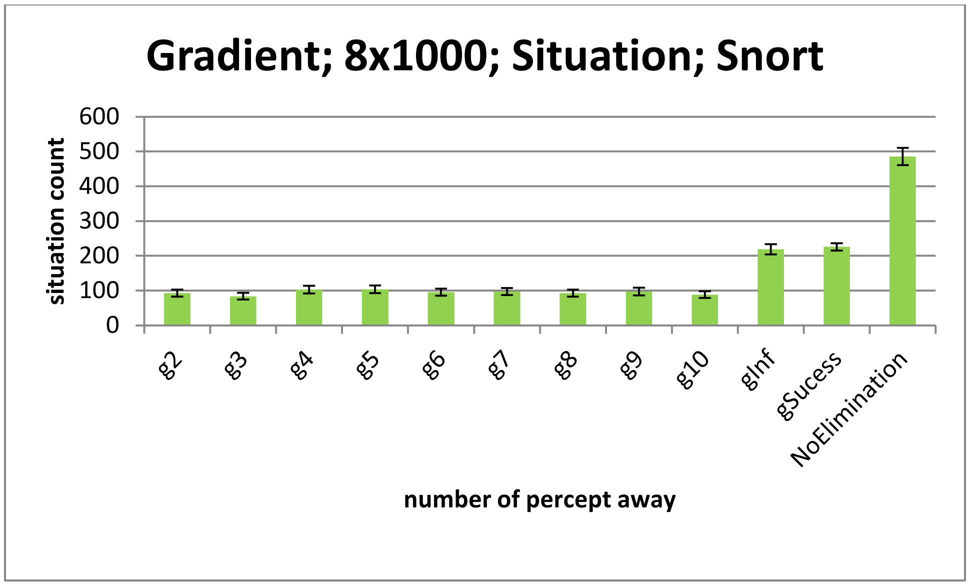


Figure 107. Effect of Gradient Difference on Snort Alert Situation Count, 8x1000.

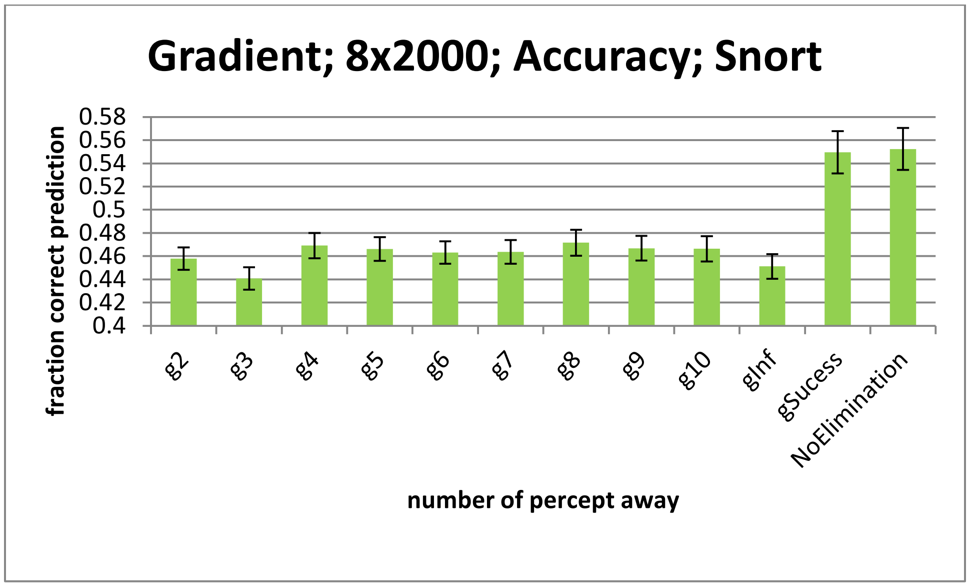


Figure 108. Effect of Gradient Difference on Snort Alert Prediction, 8x2000.

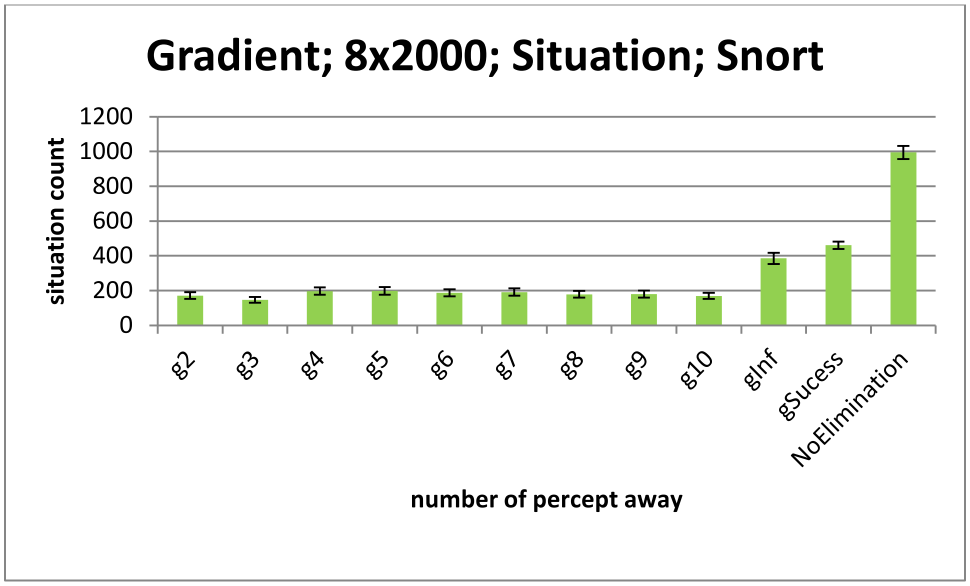


Figure 109. Effect of Gradient Difference on Snort Alert Situation Count, 8x2000.

7. Experiment on Longer Time Series.

Using some of the promising situation elimination methods above, we conducted experiments on a longer time series of 16,000 Snorts alerts. This dataset is combined from datasets 1 and 2. The chosen methods are:

Bit1 – 1 consecutive prediction success

Bit2 - 2 consecutive prediction success

BitLearning – varying consecutive prediction success

g8 – gradient of efficient difference between current and 8 prediction events ago

gInf – gradient of efficient difference between current and first prediction events

g2 - gradient of efficient difference between current and 2 prediction events ago

f0.1 – 10% of the original memory size

fLearning – varying fraction of the original memory size

The results are as shown in the four charts below (Figure 110. , Figure 111. , Figure 112. and Figure 113.). These elimination techniques appear to have their accuracy converge to the no-elimination one. From Figure 113. and Figure 111. , f0.1 and fLearning are the best methods for achieving the best accuracy while maintain least number of situation.

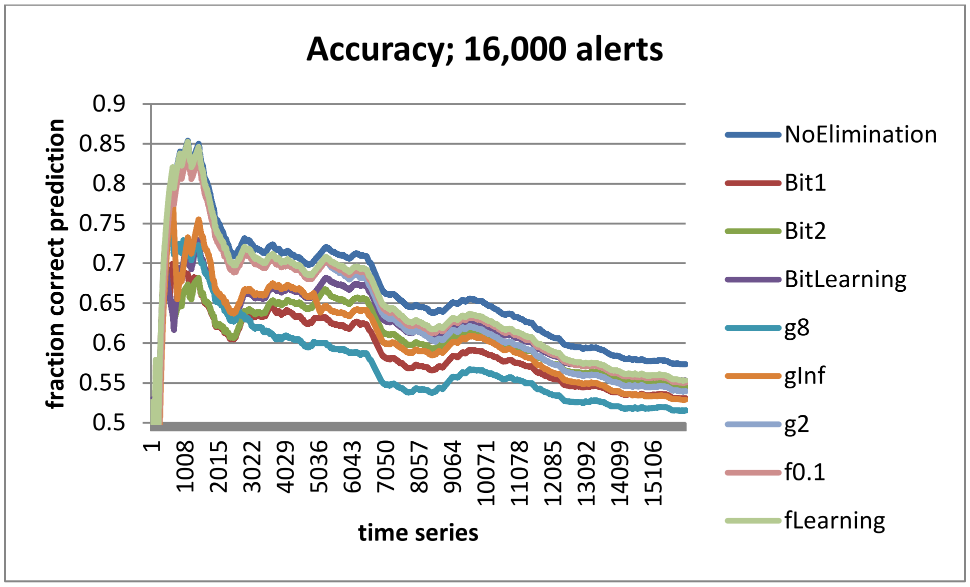


Figure 110. Comparison of different Situation Elimination Techniques on Prediction over Time.

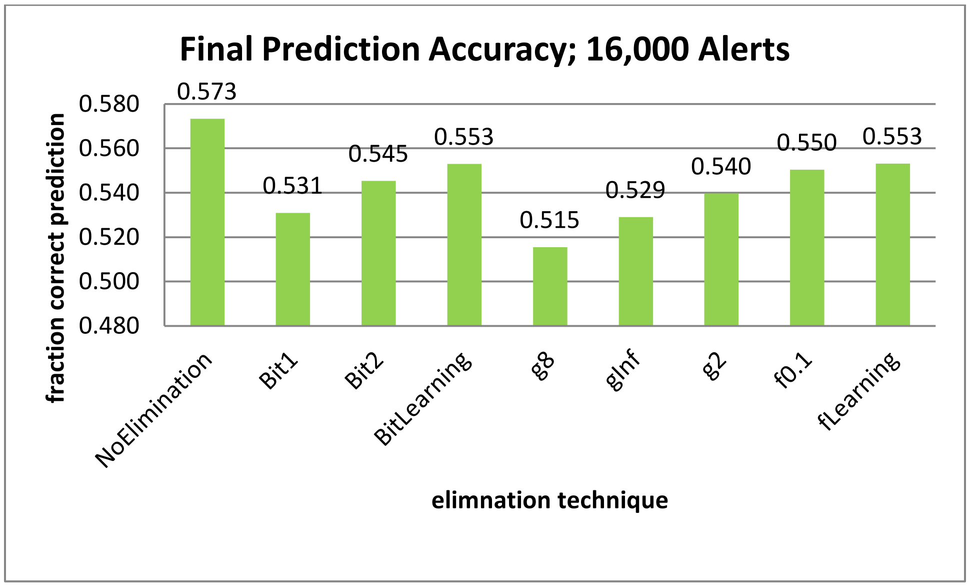


Figure 111. Comparison of different Situation Elimination Techniques on Final Prediction accuracies.

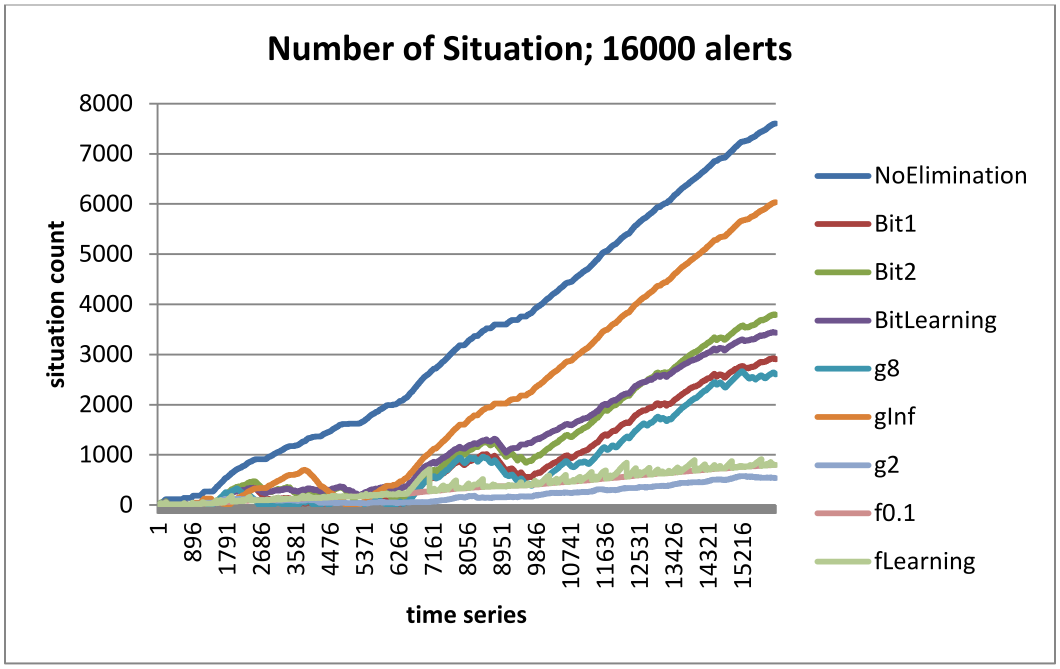


Figure 112. Comparison of different Situation Elimination Techniques on situation Count over Time.

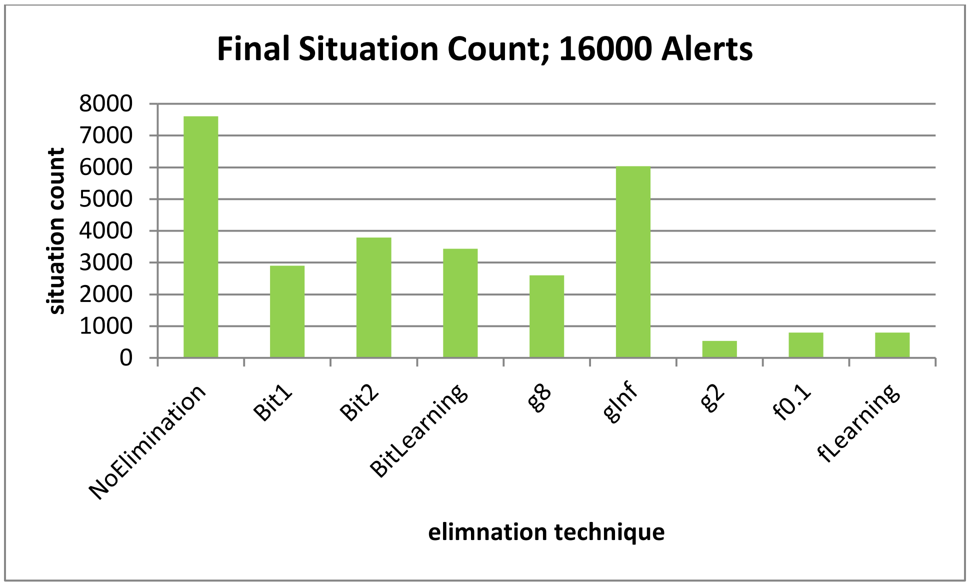


Figure 113. Comparison of Different Situation Elimination Techniques on Final Situation Count.

The next set of results below (Figure 114. and Figure 115.) compares the effect of fLearning situation elimination with the No-Elimination. The prediction accuracies are similar but the computation time has significantly been reduced.

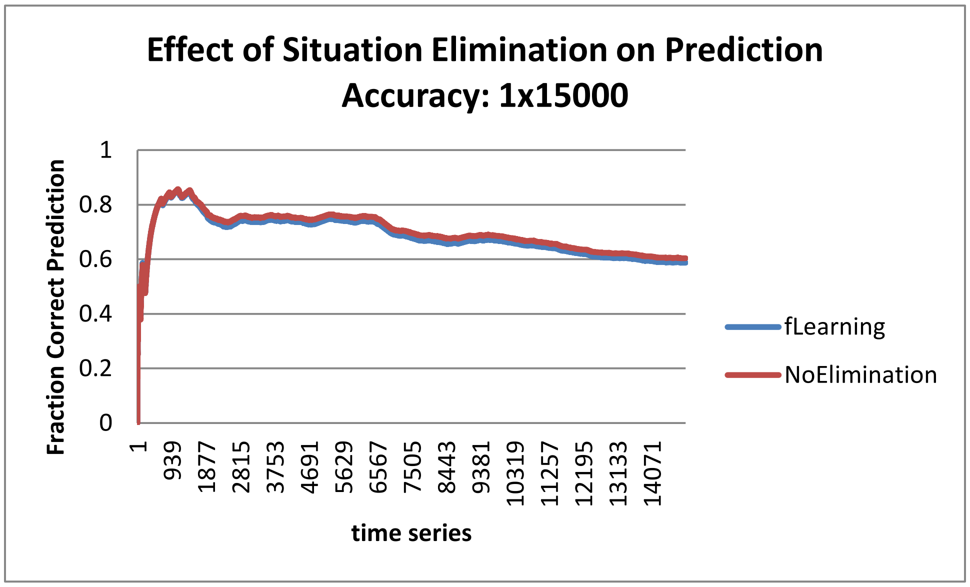


Figure 114. Effect of Situation Elimination on Prediction Accuracy: 1x15000.

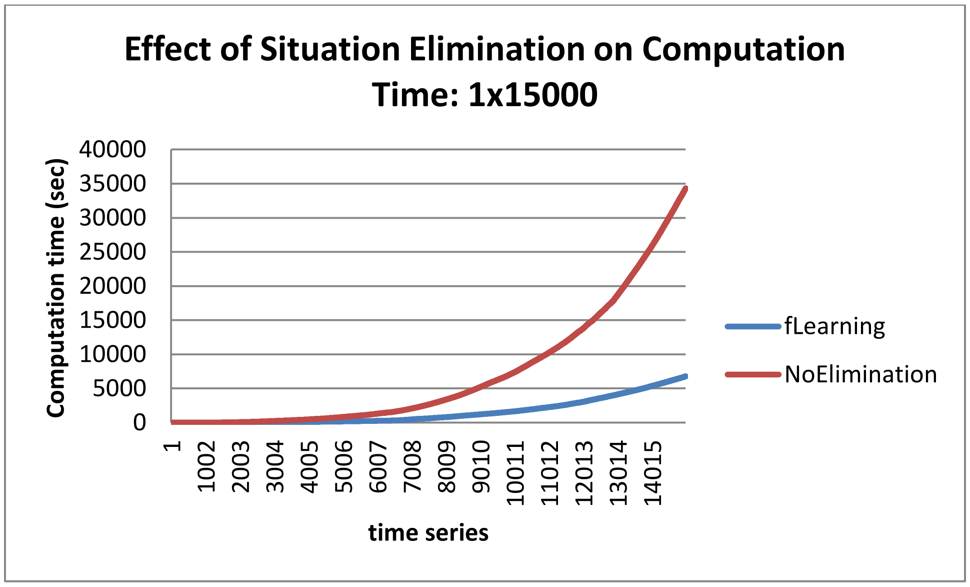


Figure 115. Effect of Situation Elimination on Computation Time: 1x15000.

8. Effect of Situation Elimination on Prediction Accuracy on Pymud

The virtual environment of the Pymud environment used for this experiment was a small one that only has 19 rooms. As a result, the Pymud is a more stationary domain. The probability of returning back to the situation is high. Hence, situation elimination may not work well. We repeat the above exercise on Pymud but found that none of the above elimination methods can provide significant situation reduction while maintaining similar accuracy. A selection of some of the better situation elimination techniques on Pymud and their associated results are described in Figure 116. and Figure 117. . From these two charts, we can see that as we reduce the situation count, the prediction accuracies also reduces accordingly and significantly. Nevertheless, the bit learning (black line) method appears to be a good compromise between prediction accuracies and situation count. The final prediction accuracies is 12% lower but the situation count is reduced by 80%.

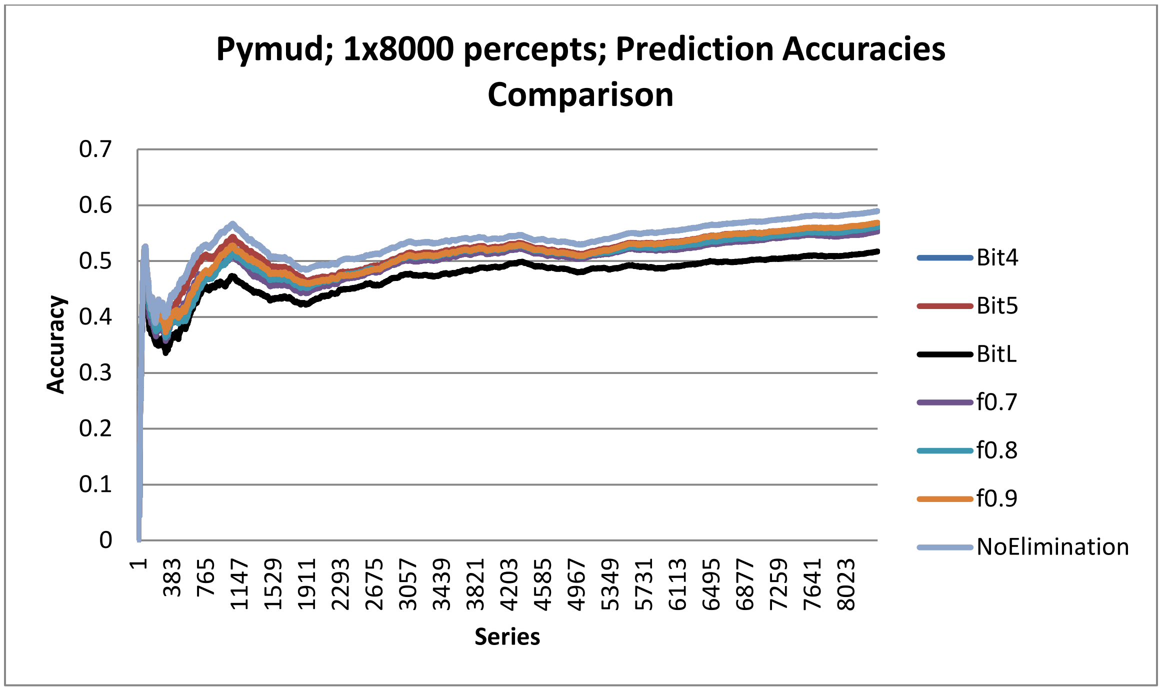


Figure 116. Effect of Situation Elimination on Prediction Accuracy for Pymud: 1x8000

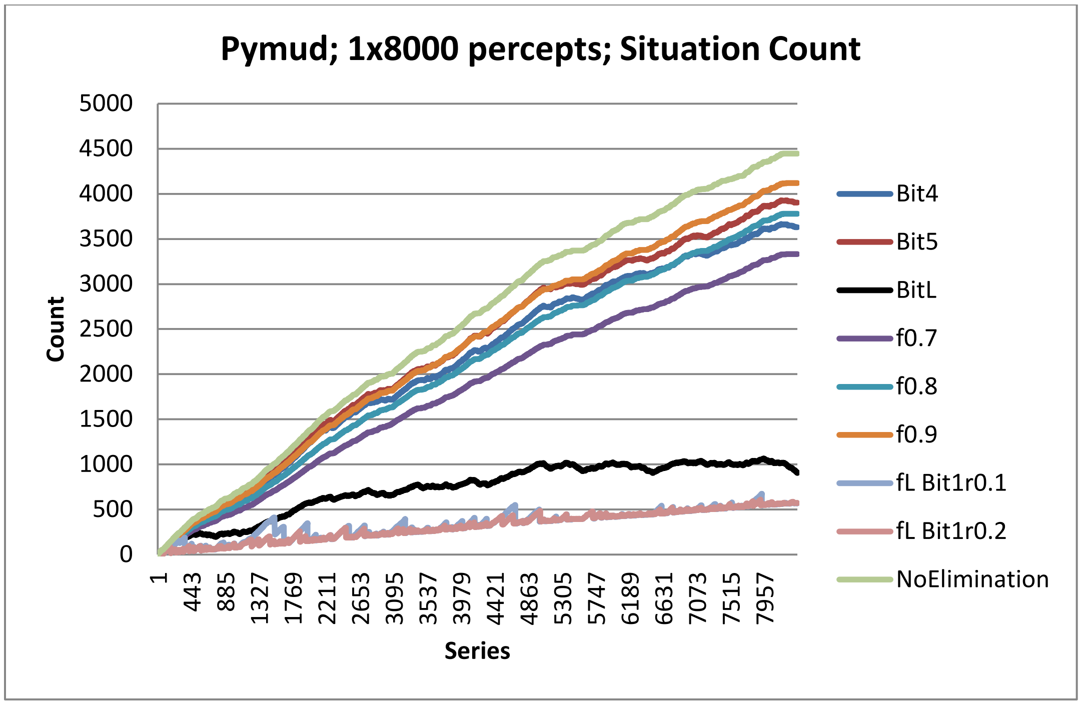


Figure 117. Effect of Situation Elimination on Situation Count for Pymud: 1x8000

F. CONCLUSION

In this exercise, we have shown that we can improve the efficiency of time-series learning and prediction on network intrusion-alert predictions by reducing the situation and yet, still able to maintain similar prediction accuracy. There are many methods of learning the memory size. In the methods that we tried so far, changing the maximum memory size by a varying fraction of the original memory size is the most optimal one on network intrusion alert predictions.

Situation elimination is more applicable to a domain that is not stationary. The Snort alert prediction is a non-stationary domain because the alerts generated, though highly repetitive, only exist in certain time duration and almost never return. This is not true in the Pymud domain because the agent can return to the same encounter in which previous situations are still relevant no matter how old or how infrequent it can be. Nevertheless, we have also shown that situation elimination using bit learning works partially for Pymud domain if minor reduction in prediction accuracies can be accommodate for faster prediction time.

THIS PAGE INTENTIONALLY LEFT BLANK

X. DOUBLE-SCOPE BLENDING

A. INTRODUCTION

There are four types of blending (simple, mirror, single-scope and double-scope), depending on the structure of the input and blended spaces. The structures are defined by a set of relations and object constant types. The simple network is one which one input space contains the structure while the other input space does not have a structure but only object constants and types (role). Mirror-scope is one which all spaces have the same structure. In single-scope, both input spaces have different structures. The structure of one space (previous situation) is used for the blended situation. In double-scope blending, all spaces have different structures. A new structure is generated for the blended situation.

The central idea for double-scope blending is to create a new structure, one that we have not seen before, that is useful to reason about a current situation that we have not seen before. An example is given in Figure 9. and shown again in Figure 118. . In the figure, we have a current situation that describes a new situation (a Pegasus) that we have not seen before. In order to predict its capability, we have to find something similar. However, if we could only find a horse and a bird in our previous situation, we can create a new structure (or creature) that combines the horse and the bird structures. The new structure may allow a better understanding of the new situation. However, the resultant meaning of the new structure depends on the parts of the old structures added to the new structure. It is possible and common that many structures generated are nonsensical.

Fauconnier and Turner [42] describe that the Microsoft Windows desktop is a double scope blend of an office environment and the world of computer science. Goguen and Harrell [48] use double-scope blending for machine poetry generation. Pereira [49] generates creative animation characters by blending known characters. Tan and Kwok [50] use double-scope blending to generate creative scenarios of maritime terrorism from known scenarios.

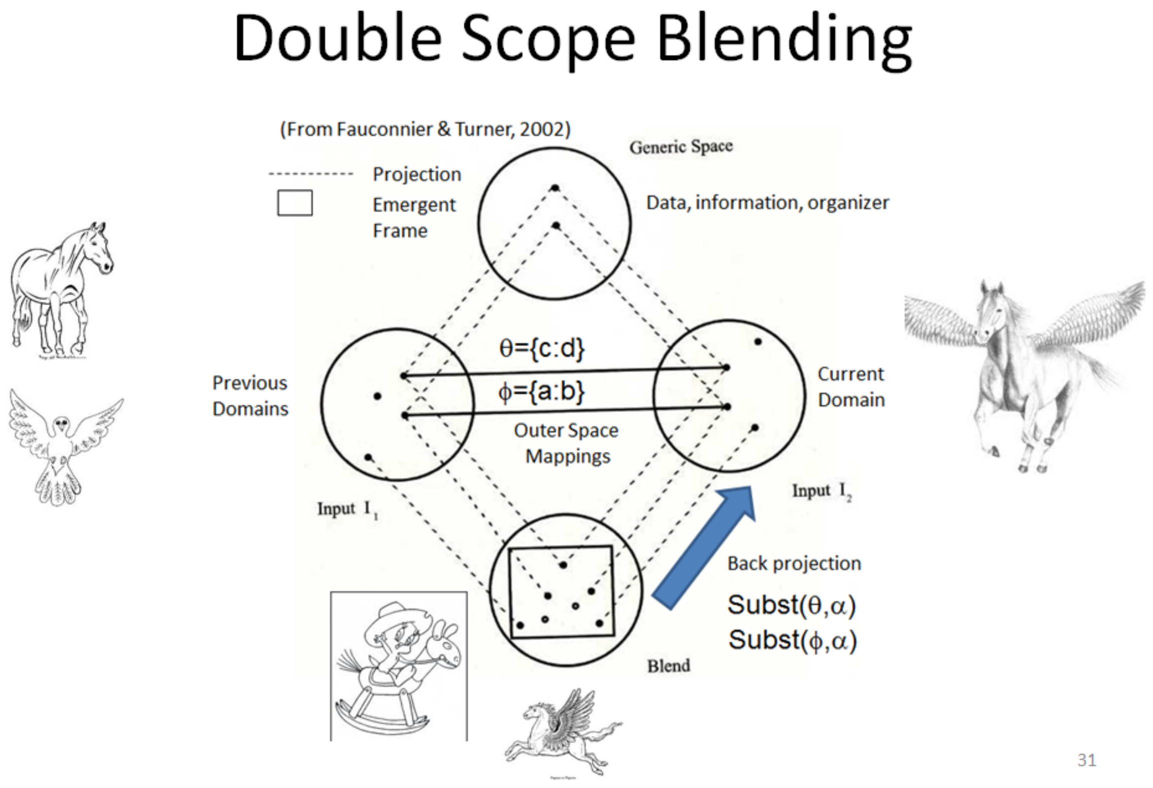


Figure 118. A Cartoon Example for Double-scope Blending [42]

Our initial assessment of using double-scope blending on relational time series prediction was discouraging because double-scope blending can generate too many structures and possibilities such that there is no way to evaluate these structures to determine which one should we use to make a prediction. The number of possible new structures are in the order of $(|n_1|+|e_{1|})^{(|n2|+|e2|)}$ where n_1 and n_2 refer to the number of constant nodes in situation 1 and 2, respectively, and e_1 and e_2 are the number of relations in situation 1 and 2, respectively. Nevertheless, we could use a very simple form of double-scope blending to avoid exponential explosion of possible structures by deviating minimally from the most similar situation. We will describe two simple double-scope blending: blending of current and previous situation, and blending of two previous situations.

B. BLENDING OF CURRENT AND PREVIOUS SITUATION

1. Description

In our single-scope blending algorithm described in algorithm 5 and 6, constants that are of different type will not be bound. Binding of constants that are not the same type will violate the single-scope blending because the structure of the selected previous situation will be changed. This is true since some bindings of different type are nonsensical. For example, IP address should not be bound to a Protocol. A pitchfork should not be bound to a dragon. However, some dissimilar type bindings are acceptable such as pitchfork and dagger because they share a common higher level: weapon.

To allow dissimilar type bindings, we need to generate a new structure, a blend of the current situation with the previous situation, by changing the type of the constant in the previous situation. To avoid nonsensical dissimilar types, we created a classification of types to allow dissimilar types to be classified into the same class so that dissimilar type from the same class can be bind. The classifications are described in the table. Such classification knowledge may be found in dictionary or knowledgebase such as Princeton University's WordNet or MIT's Semantic Net. An illustration is given in Figure 119. . The previous situation has an object constant of type troll while the current situation has a goblin. Since both troll and goblin belong to the class "monster", we allow such binding and change the structure of the previous situation to have goblin instead of troll.

Class	**type**		
Weapon	Pitchfork	Dagger	Sword
Monster	Goblin	Troll	Dragon

Table 18 Classification of Types by knowledge.

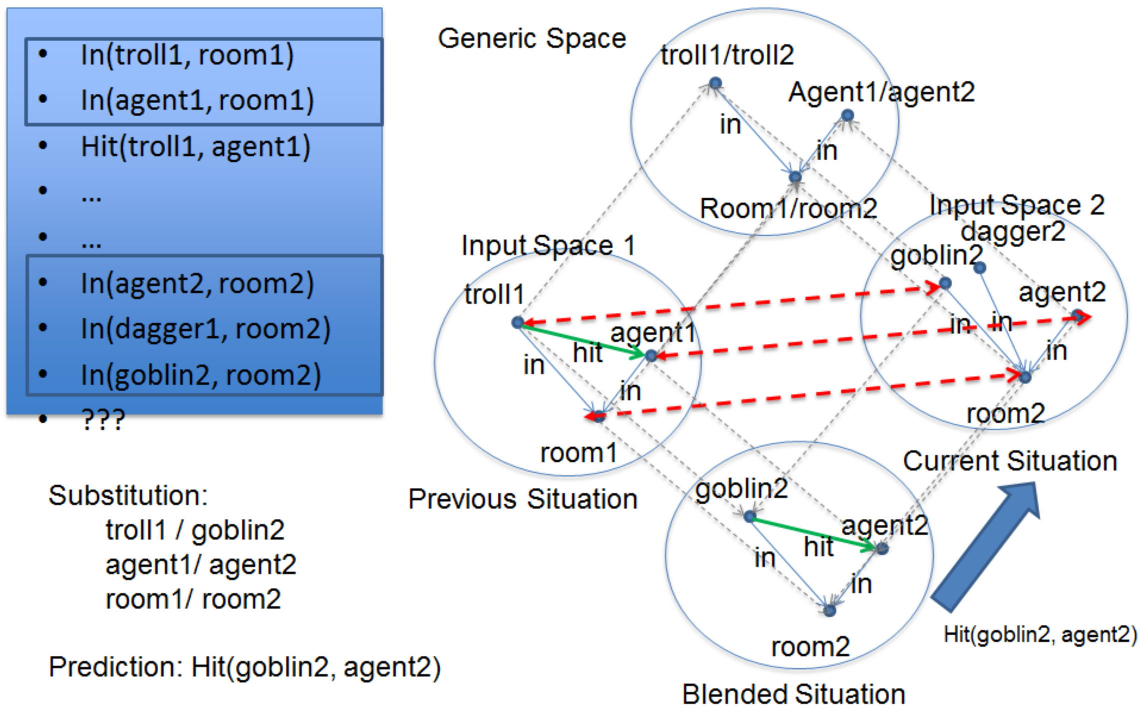


Figure 119. Illustration of Double-scope Blending of Previous and Current Situation Situations by Types.

2. Experiment

We repeated the experiments described in chapter IV on Pymud. Everything else remained the same except that constants that were of different types, but belonged to the same class, were allowed to be unified. The objective is to see if we can get better prediction accuracy on this type of double-scope blending. This type of double-scope blending is not suitable for the intrusion-alert prediction because different constant types cannot be further classified for unification.

3. Result

The prediction accuracies for double-scope blending by type and single-scope blending for two different relational time-series length are given in Figure 120. . The significant test results are given in Table 19. For length 100, there is no statistical significant for both paired and group t-test. This is likely due to no encounter of different type constant within that short time series. For 1000 length, paired t-test indicates that both double-scope blending by type and single-scope blending are different. While group t-test shows that there is no significance difference, it does show a trend of significant difference as time-series lengthen. This is likely due to the fact that constant of dissimilar types are frequently encountered as time-series lengthened.

	SSB	DSB (Type)
Paired	0.4982	0.0125
Group	0.9708	0.8484

Table 19 Significant Test Comparing Double-scope Blending by Type to Single-scope Blending.

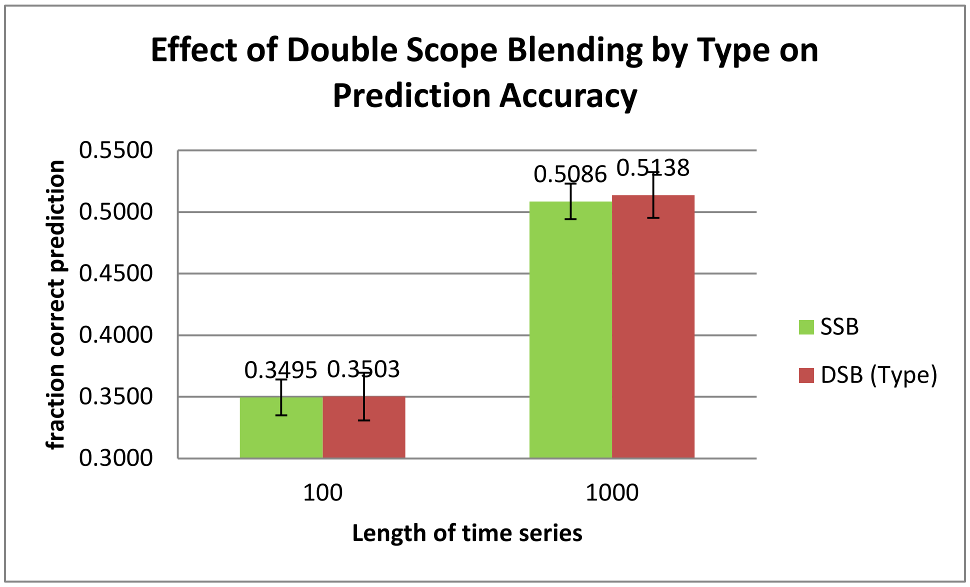


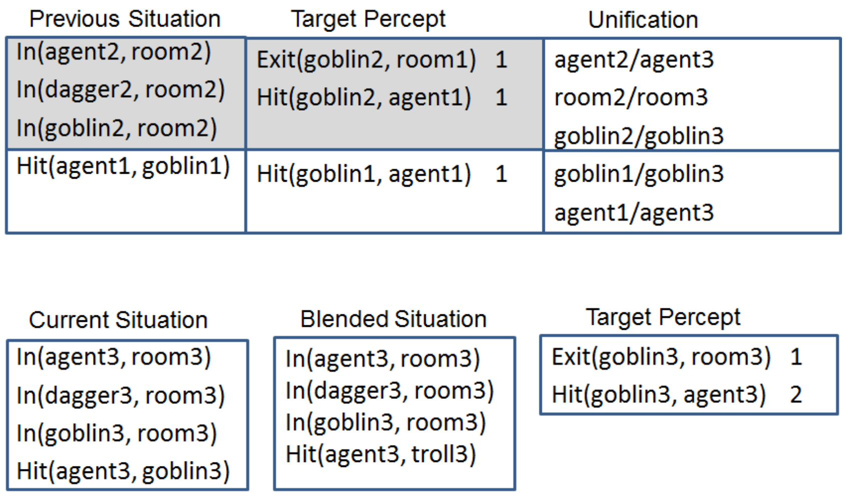
Figure 120. Effect of Double-scope Blending by Type on Prediction Accuracies.

C. BLENDING OF TWO PREVIOUS SITUATIONS

1. Description

In our single-scope blending, the most similar previous situation is used to make prediction. Frequently, not all percepts in the current situation are found in the selected previous situation. The research question here is to see if we are able to find another previous situation that serves to supplement the selected previous situation such that the combined situation will be more similar than the current situation. The algorithm is given in Algorithm 12.

An illustration is given in Figure 121. . Suppose we have identified the most similar situation as shown in the left shaded box, with its associated target percepts. The similarity score is ¾. Both target percepts have the same number of occurrence of 1. If the desired prediction is the second percept, we would be wrong because we would have chosen the first one based on the default tie breaking rule. We will look for another situation such that when added, will improve the similarity score. We found another situation, list at the second row, such that when added into the most similar previous situation, and after unification, results in a better score of 4/4. Note that during the initial search process for the most similar situation, all previous situations constants would already been unified with the current situation. Note that in the final outcome, the second target percept "Hit()" is chosen for prediction because it now has a higher occurrence count than the target percept Exit().



Previous Situation	Target Percept	Unification

In(agent2, room2)
In(dagger2, room2)
In(goblin2, room2)
	Exit(goblin2, room1) 1
	Hit(goblin2, agent1) 1
		agent2/agent3
		room2/room3
		goblin2/goblin3

Hit(agent1, goblin1)	Hit(goblin1, agent1) 1	goblin1/goblin3 agent1/agent3

Current Situation	Blended Situation	Target Percept

In(agent3, room3)
In(dagger3, room3)
In(goblin3, room3)
Hit(agent3, goblin3)
	In(agent3, room3)
	In(dagger3, room3)
	In(goblin3, room3)
	Hit(agent3, troll3)
		Exit(goblin3, room3) 1
		Hit(goblin3, agent3) 2

Figure 121. Illustration of Double-scope Blending of Two Previous Situation and Current Situation.

Algorithm 12: Blending Two Previous Situation

```
$s_{current}$        ← current situation
$s_{match}$          ← previous situation that best match current situation
$\theta_1$={a/b}     ← constant bindings of $s_{match}$ and $s_{current}$
                         such that SUBST($\theta_1$, $s_{match}$) = $s_{current}$
$\phi_1$ {a/b}       ← percept bindings of SUBST($s_{match}$) and $s_{current}$
                         such that $s_{match}$.a =  $s_{current}$.b
$\Omega_1$           ← $s_{current}$ - SUBST($s_{match}$)
$S_{previous}$       ← previous situations - $s_{match}$
$n_{unmapped}$       ← |$\Omega_1$|
$s_2$                ← none
For $s_p$ in $S_{previous}$
      $\theta_2$={a/b}     ← constant bindings of $s_p$ and $s_{current}$
                         such that SUBST($\theta_2$, $s_p$) = $s_{current}$
      $\phi_2$={a/b}       ← percept bindings of SUBST($s_p$) and $s_{current}$
                         such that $s_{match}$.a =  $s_{current}$.b
      $\Omega_2$           ← $\Omega_1$ – ($\Omega_1 \cap$ SUBST($s_{match}$))
      If |$\Omega_2$|< $n_{unmapped}$:
            $s_2$ = $s_p$
Ranges = $s_{match}$.ranges + $s_2$.ranges
If $s_{match}$.ranges.$r_1$ == $s_2$.ranges.$r_2$
      P = $r_1$.p + $r_2$.p - $r_1$.p * $r_2$.p
If tie, increase p by w1 and w2: p += (1-p)*w1*w2
w1 = n/len(range) where n is the number of terms in range found in
common percept
w2 = prior probability of range
```

2. Experiment

We repeated the experiments described in chapter IV (Pymud) and V (Snort alert). After the most similar situation was found, we looked for a second situation that could best supplement the most similar situation and result in a better situation match. When found, the newly blended situation was used for the prediction. Otherwise, single-scope blending is used. The objective is to see if we can get better prediction accuracy on this type of double-scope blending when compared to single scope blending.

3. Result

The results of the experiment on Pymud are given in Figure 122. and the significant test results for comparing the difference between double scope and single scope blending are given in Table 20. From the group t-test, there is no significant difference in all three sequence length. However, the paired t-tests show that there is a significant different for length 100 and 10,000. We notice that there is a trend that the double-scope blending prediction accuracy gets better progressively.

	p-value for comparing double and single scope blending		
	100	1000	10000
paired	0.00	0.32	0.01
group	0.52	0.95	0.74

Table 20 Significant Test Comparing Single-scope Blending and Double-scope Blending by Blending Two Previous Situations.

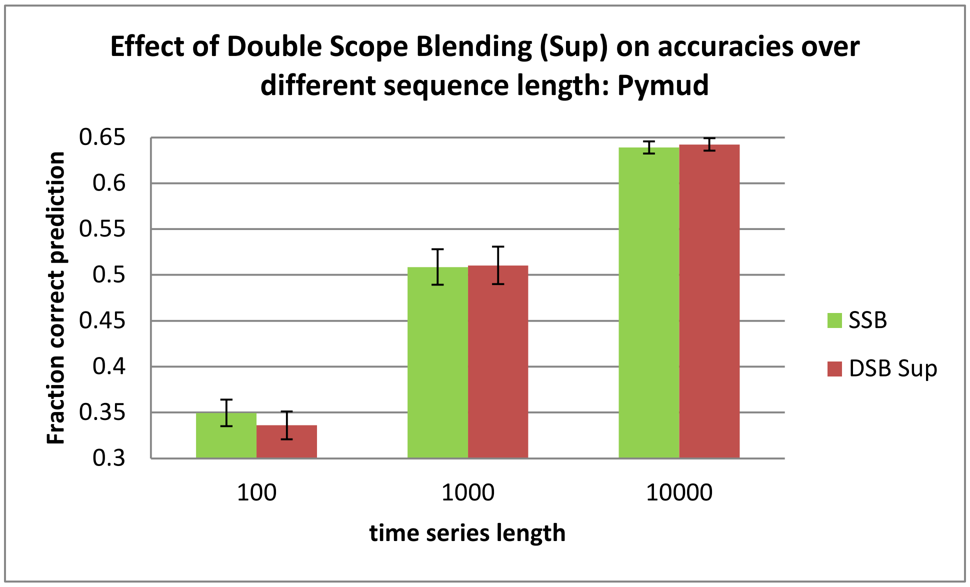


Figure 122. Effect of Double-scope Blending (Sup) on Accuracies over Different Sequence Length

To further analyze this trend, we ran a longer time series on Pymud for 100,000 percepts. The prediction accuracies comparison is given in Figure 123. . The differences are plotted in Figure 124. . We observe a trend that the double-scope blending prediction accuracies is increasingly better until around 5000 percepts and begins to narrow and converge beyond that point. Figure 125. provides some insight that explains the pattern in Figure 124. and describes the rate at which new situation are created as each percept arrives. New situation encounter rate is very high initially and maintain at more than 60% until around 5000 percepts point. The rate eventually reduces quickly and goes below 20%. This trends coincide with the prediction accuracies difference trend in that, when new situation is frequently encountered, double-scope blending tends to perform better, However, as the rate of new situation encounter reduces, the prediction-accuracy differences eventually converge.

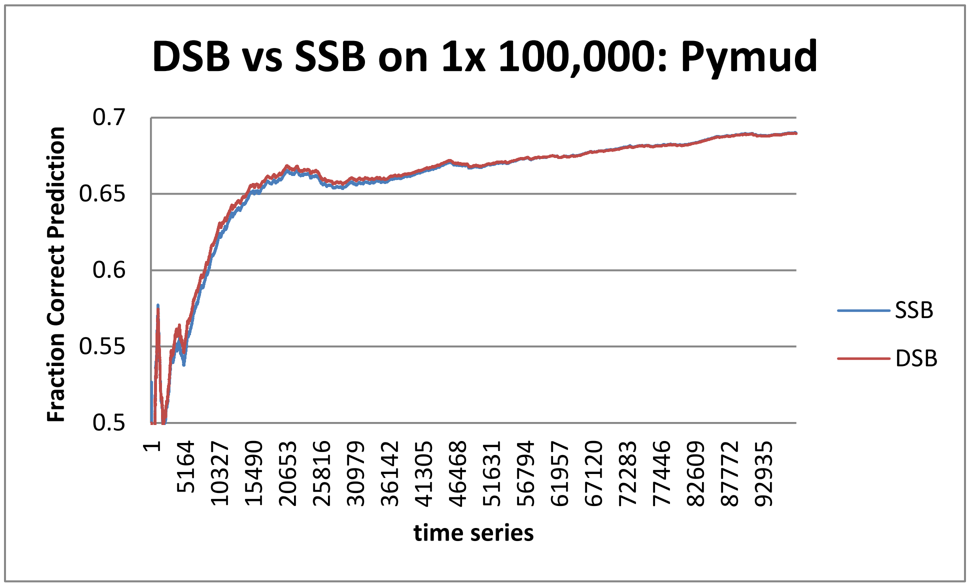


Figure 123. Effect of Double-scope Blending (Sup) on Accuracies over 100,000 Sequences on Pymud. Double-scope blending (DSB). Single-scope blending (SSB).

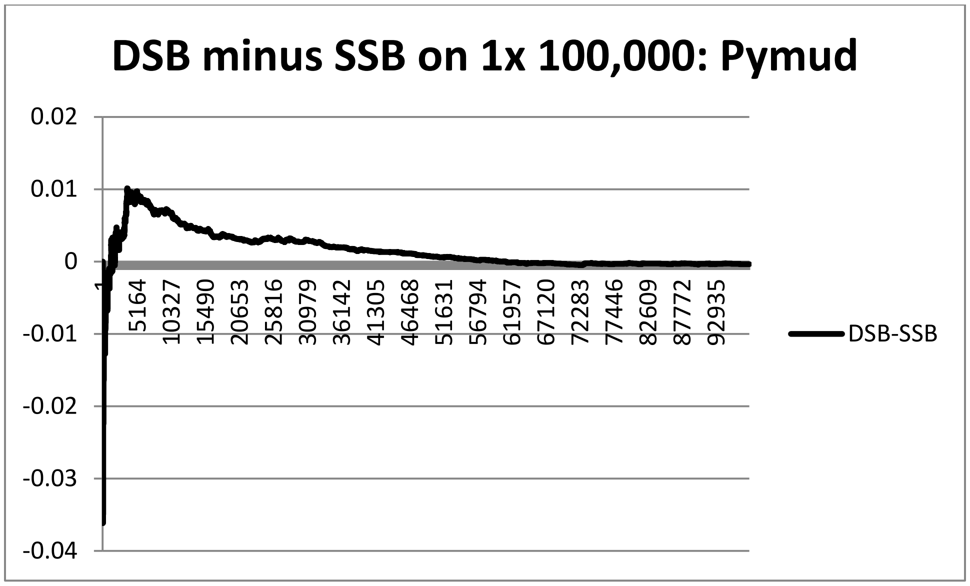


Figure 124. Prediction Accuracies Difference over Time

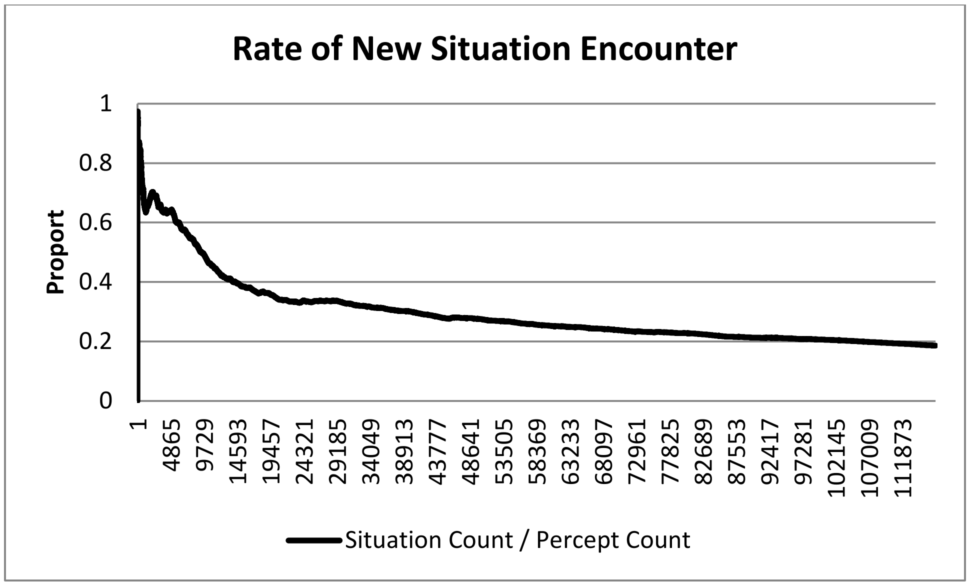


Figure 125. Rate of New Situation Encounter.

Similar experiment was conducted on the Snort alert datasets 1 and 2. The prediction-accuracy comparison between double scope and single-scope blending for 160x100 and 16x1000 are given in Figure 126. . For shorter time series, double-scope blending appears to have better prediction accuracy, since the rate of new situation is high. However, at sequence of length 1000, there is no significant difference in prediction accuracies.

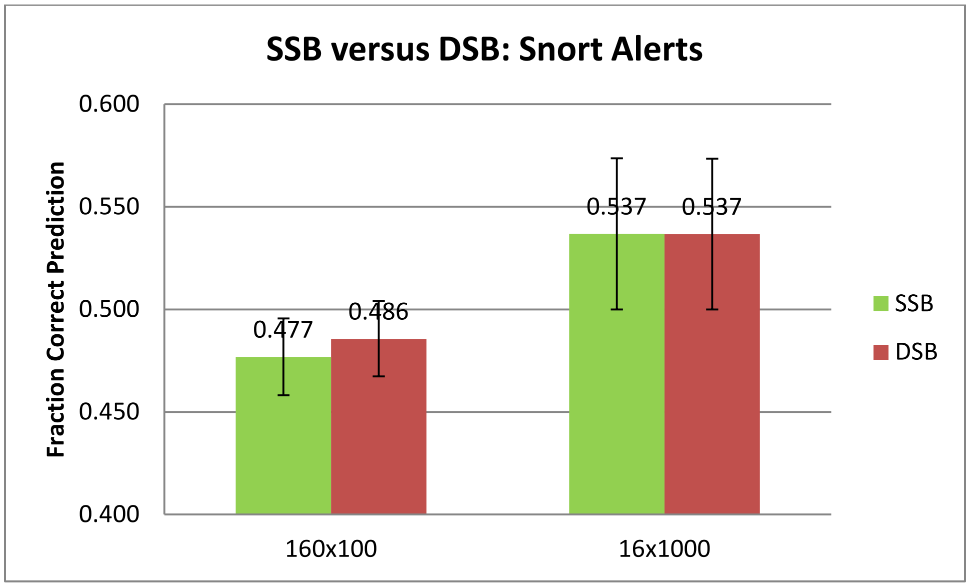


Figure 126. Comparison of AAccuracies for SSB versus DSB on Snort Alerts.

D. CONCLUSIONS

We have demonstrated two ways of doing double-scope blending. The first is a blend of the selected situation and the current situation. This type of blending allows us to bind constant of dissimilar type. Improvement on prediction accuracies have been observed on Pymud. We are unable to show this on the Snort alert dataset because the constant types cannot be classified into the same class.

The second type of double-scope blending is the blending of two previous situations. We observed that for 100 long sequences, SSB is slightly better, probably by chance. For 1,000 long sequences, the performances are the same. For 10,000 long sequences, DSB is slightly better. We also show that double-scope blending tends to be better when the rate of new situation is higher.

THIS PAGE INTENTIONALLY LEFT BLANK

XI. SENSITIVITY ANALYSIS

A. INTRODUCTION

The problem of time-series learning and prediction has many parts. Its success depends on several other variables such as the time window, time of prediction occurrence, tie breaking, etc. We will attempt to explore some issues in this chapter.

B. SITUATION TIME WINDOW

In all previous experiments described in chapters IV, V and VI, the time window has been set to 0.1 sec, which is the window used in Darken (2005). We did some sensitivity studies by looking at the prediction outcome with smaller and larger time windows.

1. Experiment

The experiments described in chapter IV and V were rerun by varying the time window at sizes such as 1-percept, 0.01sec, 0.1sec, 1sec, 2sec, 3sec and 4sec.

2. Results on Pymud and Discussion

The effect of time window on single-scope blending prediction accuracy on Pymud is described in Figure 127. . The statistical significance is described in Table 21. We observe that as the time window increases, the prediction accuracy decreases. The reason for this could be that the next percept generally depends on closer historical percepts. When we reduce the situation size to 1, which represents the shortest possible time window, the prediction accuracies is actually higher than the time window of 0.1sec. This shows that the occurrence of percepts in the Pymud depends on the immediate previous percept. As the time window gets bigger, situation gets bigger, and it becomes more difficult to encounter the same situation. The similarity score will generally decline.

The computation time also increases as shown in Figure 128. . This is because as the situation becomes bigger, the number of constant in each situation increases. Recall that the time complexity of the attention model is O(n^2) where n is the number of constant in each situation.

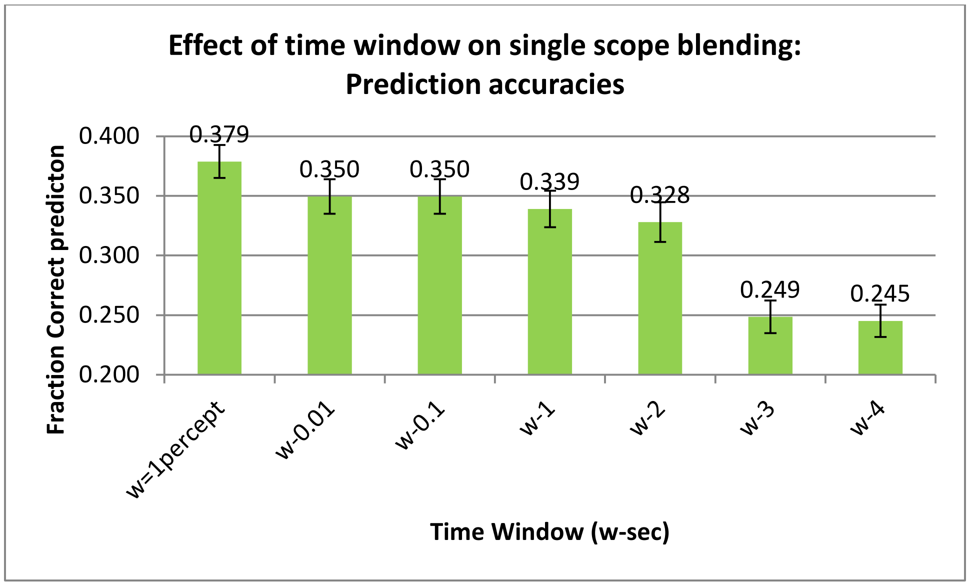


Figure 127. Effect of Time Window on Single-scope Blending Prediction Accuracies in Pymud.

	w-1percept	w-0.01	w-0.1	w-1	w-2	w-3	w-4
paired	0.000	1.00	1.00	0.00	0.00	0.00	0.00
group	0.147	1.00	1.00	0.62	0.33	0.00	0.00

Table 21 Significant Test for Effect of Time Window on Single-scope blending Prediction Accuracies in Pymud

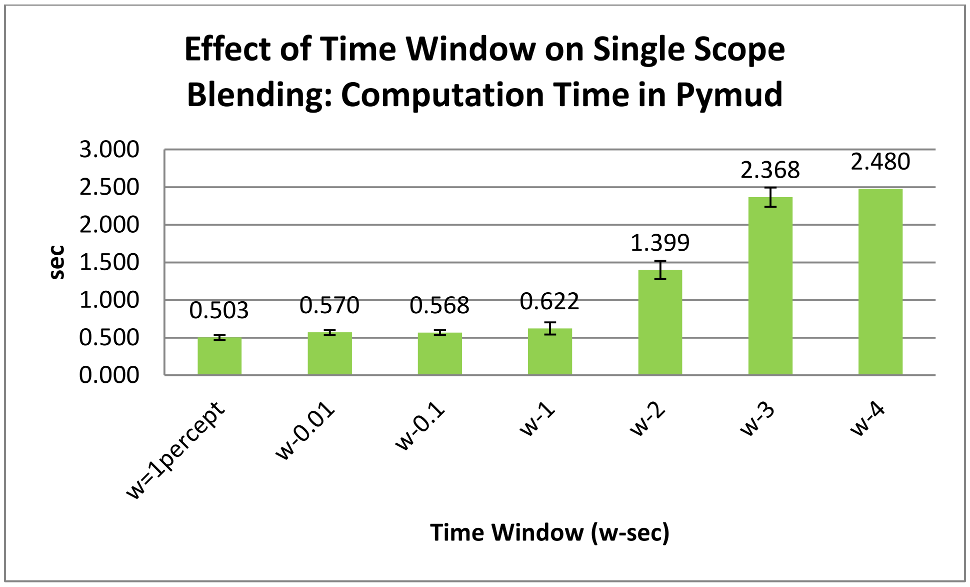


Figure 128. Effect of Time Window on Single-scope Blending Computation Time in Pymud.

3. Results on Intrusion Alerts and Discussion

A similar experiment was conducted to study the time window effect on intrusion-alert prediction. The prediction accuracy as a function of time window is as shown in Figure 129. , which shows a trend of decreasing prediction accuracy as time window increases. The significant for various cases as compared to w-0.1 are as shown in Table 22. The computation time also increases as a function of time window as shown in Figure 130. . It is interesting to note that the window of 0.001 has a negative effect on prediction accuracy as compared to w-0.01 and w-0.1. This is actually due to the inability to account for larger contexture situation when the time window becomes too small.

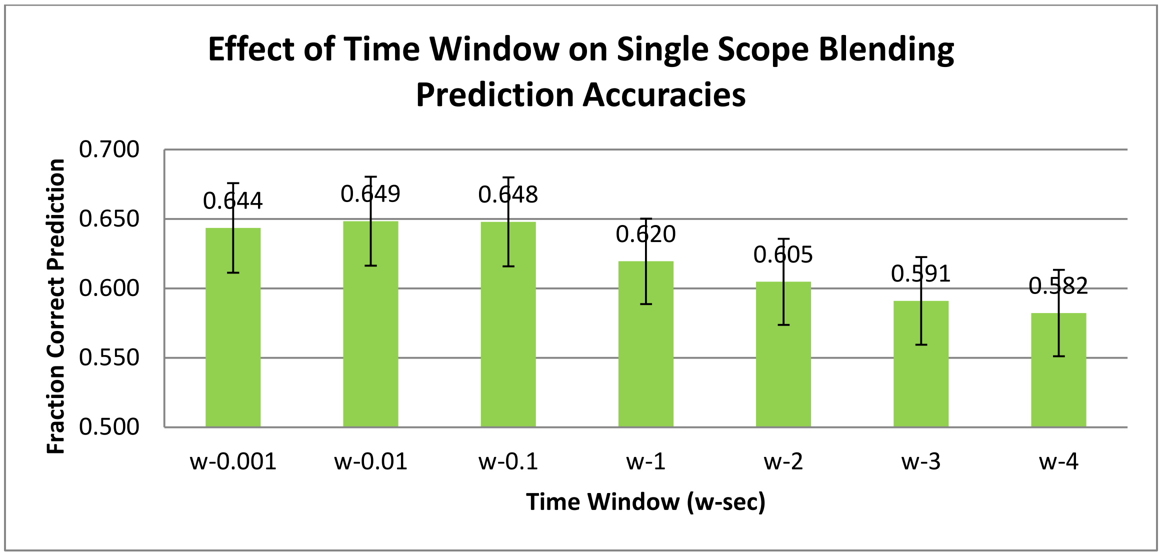


Figure 129. Effect of Time Window on Single-scope Blending Prediction Accuracies in Cyber.

	w-0.001	w-0.01	w-0.1	w-1	w-2	w-3	w-4
paired	0.922	0.991	1.000	0.003	0.001	0.000	0.000
group	0.922	0.991	1.000	0.524	0.336	0.210	0.145

Table 22 Significant Test for Effect of Time Window on Single-scope Blending Prediction Accuracies in Cyber.

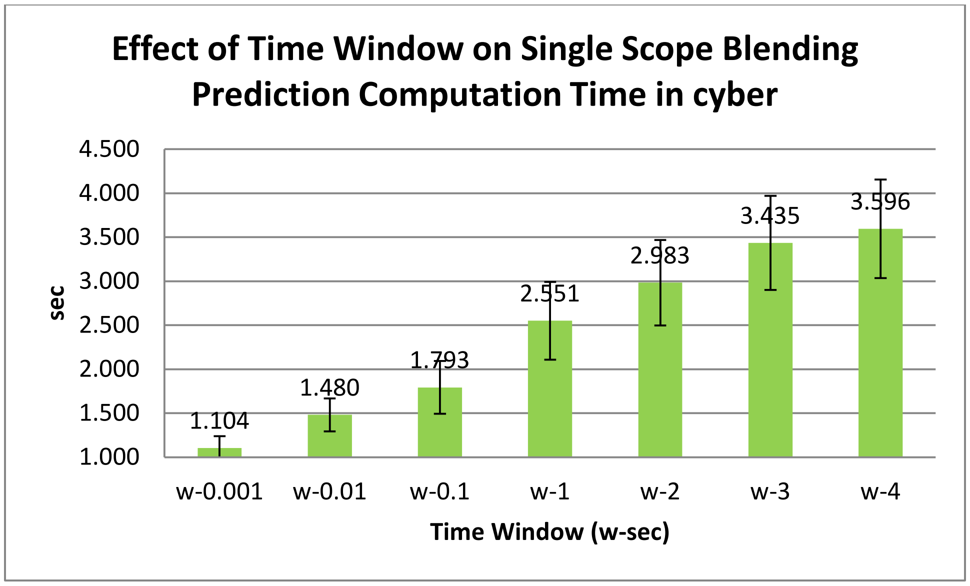


Figure 130. Effect of Time Window on Single-scope Blending Computation Time in Cyber.

4. Discussion

Each situation is a set of percepts that are related in time. If the time window is too big, irrelevant percepts may be added into a situation and provide the effect of noise. A smaller time window is preferred. However, if the time window is too small, we may not be able to account for a longer situation, which may cause the prediction accuracy to drop. From Figure 129. , while the difference is insignificant between time window 0.001 sec and 0.01 sec, there is a possibility that time window that is too small may be detrimental. Furthermore, a longer time window will affect the computation time for the conceptual blending predictors. Hence, we want a small time window but not too small. Ideally, we may want to have varying time window that is self-adjustable to maximize prediction and minimize computation time. However, a varying time window may cause more situations to be added, and increase search time. In the current state of research, we assume that this time window will be determined offline and updated periodically. With each offline determination, a different set of situations will be used in the future learning and prediction.

C. TIE BREAKING

It is possible that several previous situations may share the same similarity score when compared to the current situation. Previously, we only used the earliest situation found. In this section, we will look at various ways to break ties and to study their effect on the prediction accuracy.

The modes of tie-breakers used in this study are given in the table below

Modes	**w-0.001**
Earliest	Choose the earliest situation
Latest	Choose the latest situation
Highest Count & Earliest	Choose the situation with highest count of occurrence. Break tie by choosing the Earliest situation
Highest Count & Latest	Choose the situation with highest count of occurrence. Break tie by choosing the latest situation
Lowest Count & Latest	Choose the situation with lowest count of occurrence. Break tie by choosing the Latest situation
Lowest Count & Earliest	Choose the situation with lowest count of occurrence. Break tie by choosing the Earliest situation
Highest Target Percept Count	Gather all target percepts from all situations that have the highest similarity score. Choose the prediction that has the highest probability of occurrence

Table 23 Description of Tie-breaking Modes.

1. Experiment

The experiments described in chapter IV and V are rerun by varying the tie breaking mode as describe in Table 23.

2. Results and Discussions

The effects of different tie breaking modes for Pymud and intrusion alerts are given in Figure 131. and Figure 132. respectively. In Pymud, tie-breakers that choose the earliest, lowest earliest or highest range achieve the best prediction accuracy. For intrusion alerts, choosing the highest earliest and highest range achieve the best prediction accuracy. It is interesting to note that Pymud favors the situation with lower occurrence count while intrusion alerts favor the highest occurrence count. The highest target percept count tie breaker appears to be consistently good in both domains.

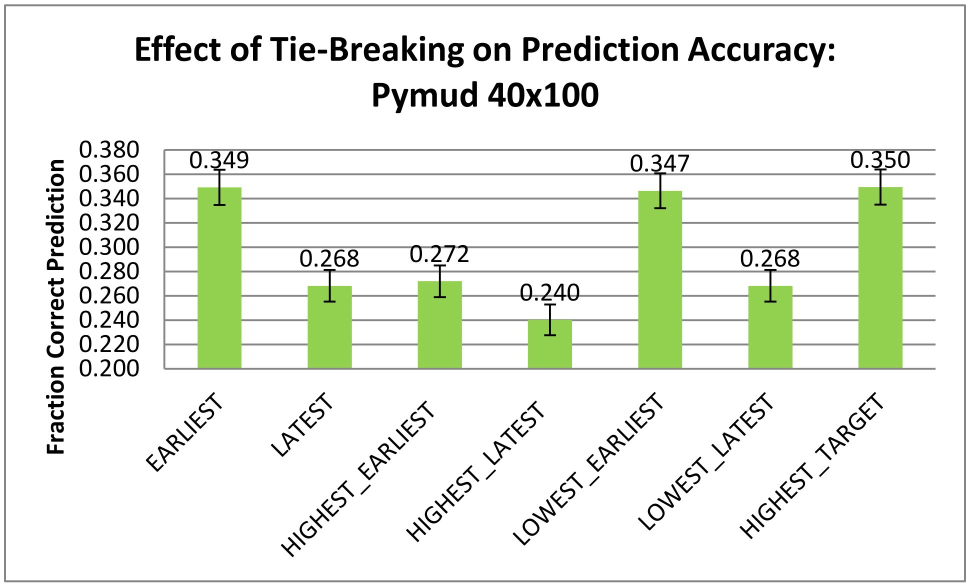


Figure 131. Effect of Tie-Breaking on Prediction Accuracy: Pymud 40x100.

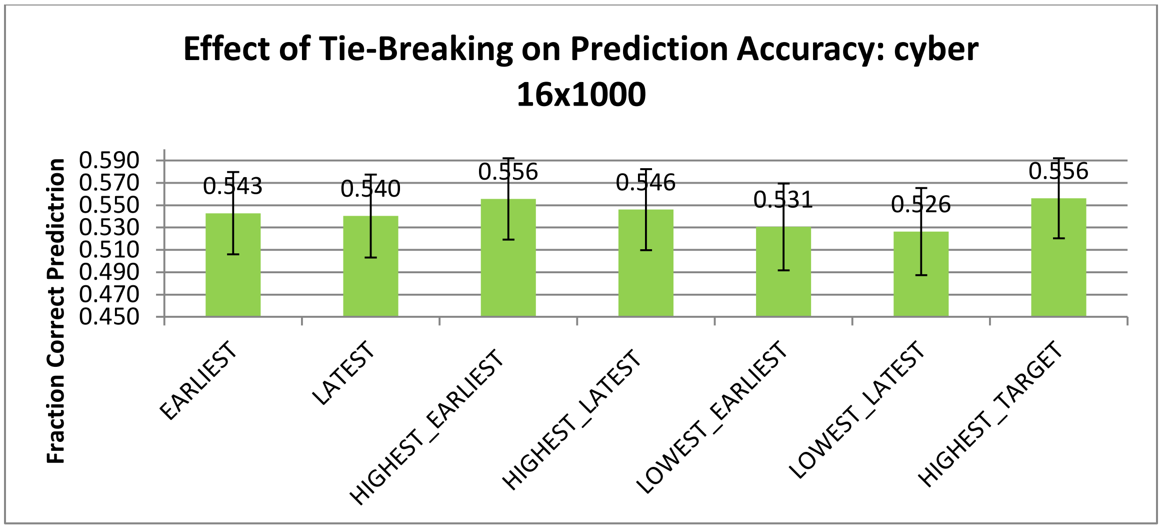


Figure 132. Effect of Tie-Breaking on Prediction Accuracy: Cyber 16x1000.

D. PREDICTION WITH TIME

The criterion for evaluating prediction accuracy in the previous experiments was very stringent. If the next percept received is not the same as the prediction, we say the prediction is wrong. In this sensitivity analysis, we relax the criterion in which, if there exists a future percept p_i and a prediction p_p such that $p_p = p_i$ and $(p_p.mt-2st) < p_i.time < (p_p.mt+2st)$ where mt and st refer to the meant time and standard deviation respectively of the time of prediction occurrence. This mean that if the predicted percept occurs at a designated range of time in the future, we say the prediction is correct even though it is not the next percept received. The range of time is computed based on the expected mean time of occurrence with two standard deviations from the expected time.

1. Experiment

The experiments described in chapter IV and V are rerun but with the new criteria of deciding if a prediction is correct.

2. Results and Discussions

The results that compare the prediction accuracies for with and without time prediction on Pymud are given in Figure 133. . With time prediction, the prediction accuracies improved for all predictors on Pymud, especially the multiple simple Bayesian network.

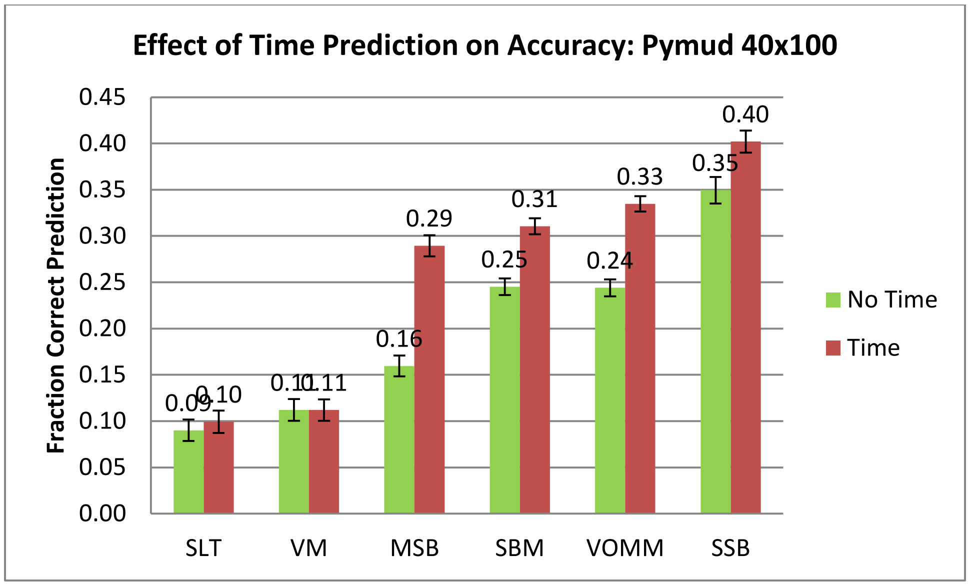


Figure 133. Effect of Time Prediction on Accuracy: Pymud 40x100.

Prediction accuracy for prediction with time on network intrusion-alert prediction is worse off as shown in Figure 134. . This could be due to the time of occurrence hardly occurs within the two standard deviations from the predicted occurrence time. If we relax the requirement further by saying that prediction is correct if it occurs anytime from prediction time to the predicted mean time of occurrence plus 2 standard deviations from the mean, the results are shown in Figure 135. . This is a reasonable assumption since if a high priority alert is predicted, we may expect it to occur almost immediately instead of waiting till the predicted time of occurrence. Nevertheless, even with more relaxation, the prediction accuracy is still not doing better than the next percept prediction.

Further investigation shows that the non-stationary nature of network intrusion alerts causes many situations learnt to have only single occurrence. As such, the standard deviation is usually zero. When standard deviation is zero, the time range becomes zero and the next percept must occur at the exact time of time prediction. This explains the poorer prediction accuracy.

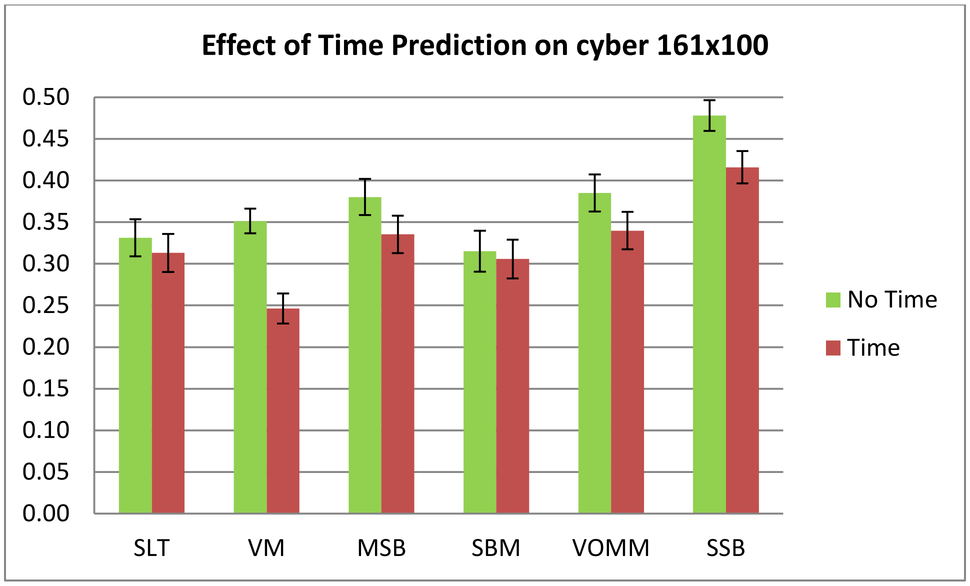


Figure 134. Effect of Time Prediction on Accuracy: Cyber 161x100.

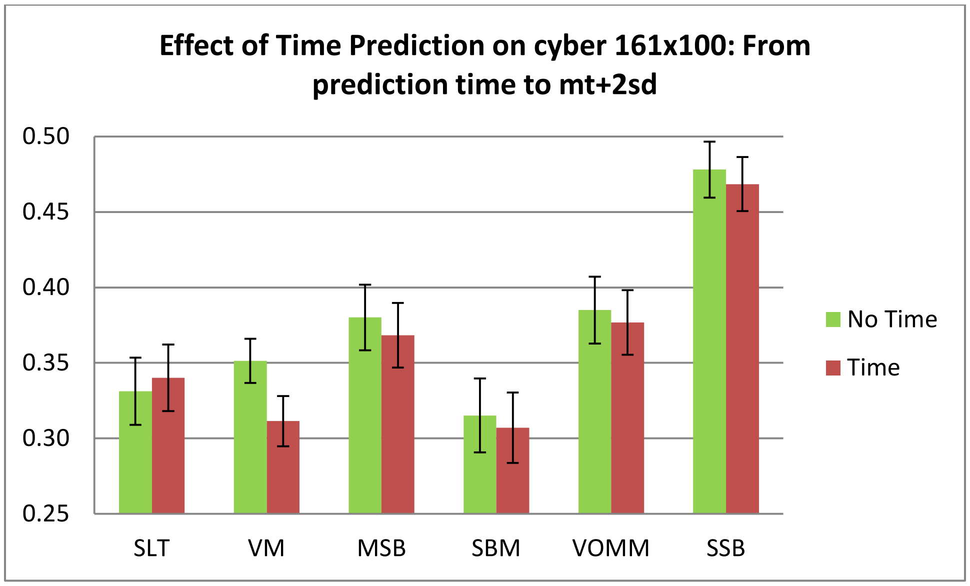


Figure 135. Effect of Time Prediction on cyber 161x100: From Prediction Time to mt+2sd Where mt Is the Mean Time of Predicted Occurrence Time and sd Is the Standard Deviation.

E. CONCLUSIONS

In this chapter, we have shown some sensitivity tests. Firstly, we explored the effect of different time window size. We found out that the window size must be kept small for both Pymud and cyber in order to maximize prediction accuracy. However, with intrusion alerts, when we reduced the time window to have just one alert, the prediction accuracy decreased. We will need to do some pre-testing to determine an appropriate window size.

The second sensitivity study looked at the effect of breaking a tie if more than two situations have the same similarity score when compared to the current situation. We show that the highest target count tie-breaker works best for both Pymud and cyber.

We have also explored the effect of time prediction. If we relax the current criterion of measuring accuracy, by allowing the predicted percepts to occur in a small time period, the prediction accuracy increases on Pymud. The prediction accuracy decrease on cyber alert prediction is due to the often novel new situation encountered and results in zero standard deviation for the expected time of arrival variation. We need a better way to collect the aggregate the data in order to have a more realistic expected time of arrival with an interval.

XII. CONCLUSION

A. CONCLUSIONS

This dissertation addresses the problem of predicting percepts that have not been experienced before, by developing algorithms and computational models inspired from recent cognitive science theories: conceptual blending theory and event segmentation theory. The parts in this dissertation can be summarized in a framework as shown in Figure 136. .

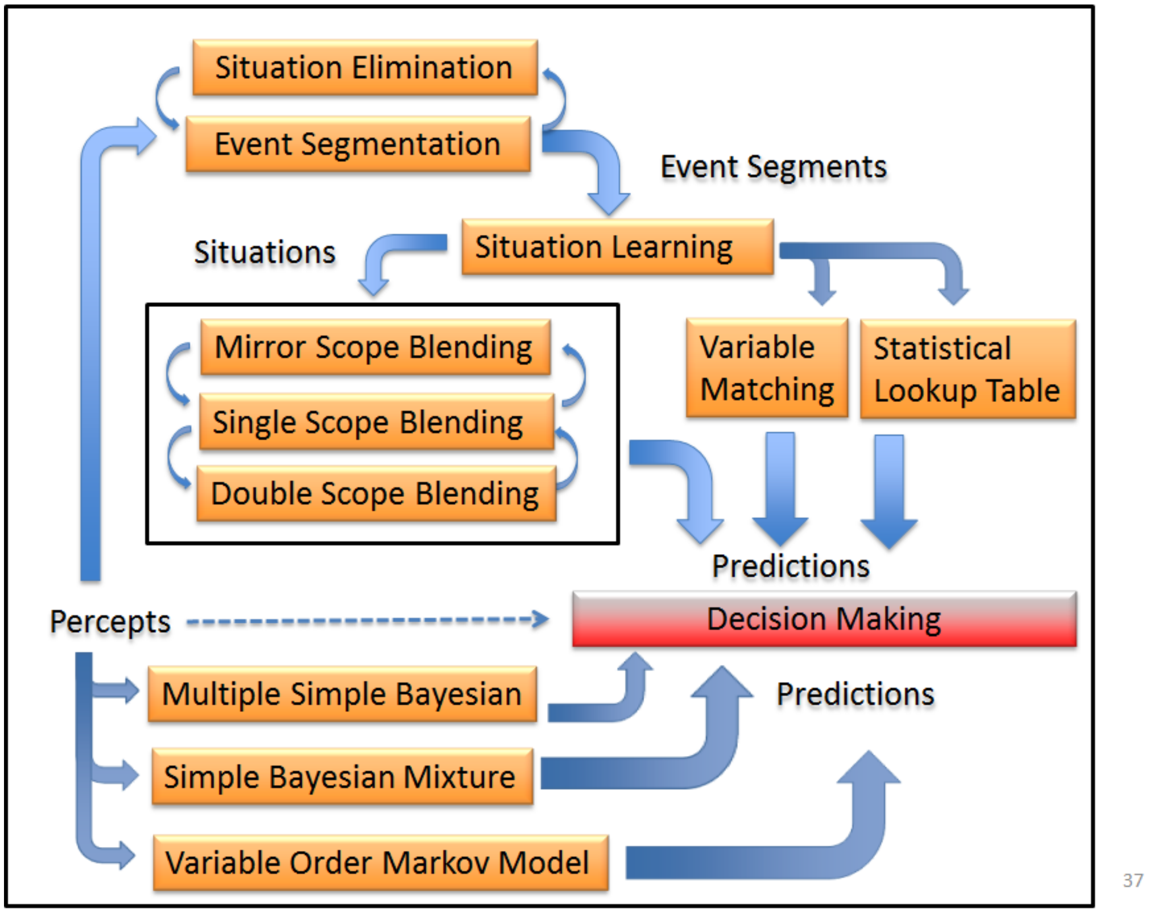


Figure 136. The Framework for Relational Time-series Learning and Prediction.

The main contribution of this dissertation is a new class of prediction techniques inspired by a cognitive science theory, Conceptual Blending that improves prediction accuracy overall with an ability to predict even events that have never been experienced before. We also show that another cognitive science theory, Event Segmentation, when integrated with the Conceptual Blending inspired prediction techniques, results in greater computational efficiency. We implemented the new prediction techniques, and other prediction techniques such as Markov and Bayesian techniques, and compared their prediction accuracy quantitatively for three domains: a role-playing game, intrusion-system alerts, and event prediction of maritime paths in a discrete-event simulator. Other contributions include two new unification algorithms that improve over a naïve one and an exploration of ways on maintaining a minimum size knowledgebase without affecting prediction accuracy.

In Pymud, we observed that single-scope blending prediction technique has significantly higher prediction accuracy than other prediction techniques: statistical lookup table, variable matching, multiple simple Bayesian, simple Bayesian mixture, and variable Markov model. The greedy best-first search is a significant improvement over the naïve backtrack technique and the attention model is a significant improvement over the greedy best-first search. The attention model is able to scale much better than the naïve backtrack and greedy best search method for solving a unification problem. The time window has to be small for optimal prediction accuracy and the recommended time window is 0.0sec such that every situation contains just one percept. On breaking ties for equally similar situation, optimal prediction accuracy was achieved by pooling all targets together and choosing the one with the highest number of occurrence. The single scope blending prediction technique is slower than other prediction techniques. To improve time performance further, we showed that Event Segmentation can help to improve computation time. The success of event segmentation depends on the event feature chosen. The event feature ‘event’ works well in the Pymud domain but ‘action’, ‘actor’ and ‘place’ result in poorer prediction accuracy. Situation elimination is another way to improve computation time, by controlling the number of situation in the lookup table without affecting the prediction accuracy significantly. We could eliminate less relevant situation-target tuples using the Consecutive-Learning situation elimination technique. However, situation elimination is suboptimal for stationary domain such as Pymud. We also observed that time prediction works well on Pymud. We also observed that double-scope blending, by using a type-hierarchical knowledge for searching for unification, produced better prediction accuracy on longer time series. Double-scope blending of two previous situations may help to improve prediction accuracy at times when the frequency of new situation is high.

In Cyber intrusion-alert prediction, we also observed that single-scope blending prediction technique has significantly higher prediction accuracy than other prediction techniques: statistical lookup table, variable matching, multiple simple Bayesian, simple Bayesian mixture, and variable Markov model. The greedy best-first search is again a significant improvement over the naïve backtrack technique and the attention model is a significant improvement over the greedy best-first search. The time window has to be small for optimal prediction accuracy and the recommended time window is 0.001sec. On breaking ties for equally similar situation, optimal prediction accuracy was also achieved by pooling all targets together and choosing the one with the highest number of occurrence. Event Segmentation can also help to improve computation time in Cyber intrusion-alert prediction. The success of event segmentation again depends on the event feature chosen. The event feature ‘ID’ and ‘protocol’ work well in Cyber intrusion-alert prediction but using the ‘protocol’ did not achieve as much computation time saving as using ‘ID’. Situation elimination works much better in Cyber intrusion-alert prediction than in Pymud to improve computation time. We could eliminate less relevant situation-target tuples using the Fraction-Learning situation elimination technique. We also observed that time prediction did not works well on Cyber intrusion-alert prediction, particularly because many distinct alerts occur only once and does not allow sufficient data point to compute the time interval based on standard deviation of the arrival time collected. We also observed that double-scope blending of two previous situations may help to improve prediction accuracy at times when the frequency of new situation is high.

In the maritime discrete event prediction, we also observed that the single scope blending has significantly higher prediction accuracy. In fact, only single-scope blending prediction technique and variable matching are able to achieve some number of correct predictions.

This dissertation solves the problem of predicting novelty by developing algorithms and computational models inspired from recent cognitive science theories: conceptual blending theory and event segmentation theory.

B. FUTURE WORK

Relational time-series learning and prediction is useful in both virtual and real worlds. There are still many areas that have not been explored.

1. Varying Time Window

We have shown that the time window is a critical parameter. Instead of fixing it, we want an automated way to vary it as the situation evolves. Further experiments need to be done.

2. Efficient Situation Indexing

Today, we start searching for situation sequentially. It is possible to index the situation to allow efficient search. However, generating an index is itself NP-Complete. It will be good to find an efficient indexing method to help to organize the situation database.

3. Mental Simulation

We have only worked on one-step prediction. Mental simulation is a multi-step prediction process. It is useful to extend our work on single step process into multi-step prediction and to develop new set of prediction approaches to improve prediction accuracy.

C. TRANSITION

The algorithms and code developed in this dissertation can be applied to decision-support systems and can potentially change the current best practice in many domains.

1. Online Learning and Prediction of IDS Alerts Cyber Security

When an intrusion-detection system alert is triggered, damage might already have been done. The ability to predict attacks earlier can allow preventive measures. We can also use the situations learned on one network domain to be used on another network domain. It would be useful to test the prediction algorithm on production network.

2. A Predictive Approach to Cyber Deception Cyber Security

Appropriate deception strategies on honeypot are important to fool an attacker into certain belief states. A deception strategy should be chosen based on the goal of the attackers. However, such goal information is not available. Nevertheless, through intrusion-detection system alerts, we can predict the future action of the attacker and can better deploy strategy to allow more successful deception such as enticing attackers to stay longer or discourage him to stay away.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] R. Sun and C.L. Giles, “Sequence learning: from recognition and prediction to sequential decision making,” *IEEE Intelligent Systems*, vol.16, no.4, pp.67–70, July-Aug., 2001.

[2] C. A. Kurby and J. M. Zacks, “Segmentation in the perception and memory of events,” *Trends in Cognitive Sciences,* vol.12, no.2, 2007. Available: doi 10.1016/j.tics.2007.11.004.

[3] D. Kunde and C. Darken, “A mental simulation-based decision-making architecture applied to ground combat,” in *Proceedings of the Behavior Representation in Modeling & Simulation*, 2006, Available: doi 10.1.1.64.713.

[4] H. Debar, “An introduction to intrusion-detection systems,” in *Proceedings of Connect*, 2000. Available: doi 10.1.1.101.7433.

[5] M. Roesch, “Snort—lightweight intrusion-detection for networks,” in *Proceedings of LISA ’99: 13th Systems Administration Conference*, 1999, pp. 229-238.

[6] A. Buss, “Component-based simulation modeling with Simkit,” in *Proceedings of the 2002 Winter Simulation Conference*. Available: doi 10.1.1.1.5324.

[7] A. Buss and P. J. Sanchez, “Simple movement and detection in discrete event simulation,” in *Proceedings of the 2005 Winter Simulation Conference IEEE*. Available: doi 10.1109/WSC.2005.1574350.

[8] C. Darken, “Towards learned anticipation in complex stochastic environments,” in *Proceedings of the Artificial Intelligence for Interactive Digital Entertainment 2005*, pp. 27–32.

[9] N. Sapankevych and R. Sankar, “Time-series prediction using support vector machine: a survey,” in *Proceedings of the IEEE Computational Intelligence Magazine*, vol.4, no.2, pp. 24–38, 2009.

[10] G. Luger, *Artificial Intelligence: Structures and Strategies for Complex Problem Solving, 6th Edition*, Boston: Addison Wesley, 2008.

[11] Z. Li, A. Zhang, J. Lei, and L. Wang, “Real-time correlation of network security alerts,” in *Proceedings of IEEE International Conference on e-Business Engineering*, 2007, pp.73–80.

[12] H. Jaeger, M. Zhao and A. Kolling, “Efficient estimation of OOMs,” in *Proceedings of Neural Information Processing Systems*, vol. 8, 2005, Available: http://books.nips.cc/papers/files/nips18/NIPS2005_0547.pdf.

[13] I. Spanczer, “Observable operator models,” *Austrian Journal of Statistics*, vol. 36, no.1, pp.41–52, 2007.

[14] H. Guo and W. Hsu, “A survey of algorithms for real-time Bayesian network inference,” in *Proceedings of the Joint Workshop on Real-Time Decision Support and Diagnosis Systems*, 2002. Available: doi 10.1.1.11.5182.

[15] R. Daly, Q. Shen, S. Aitken, “Learning Bayesian networks: approaches and issues,” *Knowledge Engineering Review*, vol.26, no.2, pp.99–157, 2011.

[16] A. Kott and W., McEneaney, *Adversarial Reasoning: Computational Approaches to Reading the Opponent’s Mind*, Boca Raton: Chapman *and Hall/CRC*, 2006.

[17] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, Upper Saddle River, NJ: *Prentice Hall*, 2010.

[18] K. Lamberts and D. Shanks, *Knowledge, Concepts, and Categories*, Cambridge: The MIT Press, 1997.

[19] L. Getoor, and B. Taskar, *Introduction to Statistical Relational Learning*, Cambridge: The MIT Press, 2007.

[20] G. P. Domingos, “Markov logic: a unifying language for information & knowledge management,” 2008. Available: http://videolectures.net/cikm08_domingos_mlmaul/, Nov. 19, 2011.

[21] D. Jensen, J. Neville and B. Gallagher, “Why collective inference improves relational classification,” in *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004, pp. 593–598.

[22] Y. Wang and M. Kitsuregawa, “Link based clustering of web search results,” *Lecture Notes in Computer Science*, pp. 225–236, 2011.

[23] B. Taskar, M. Wong, P. Abbeel and D. Koller, “Link prediction in relational data,” in *Proceedings of the Neural Information Processing Systems*, pp.659–666, 2003.

[24] H. Khosravi and B. Bina, “A survey on statistical relational learning,” in *Proceedings of the Canadian Conference on Artificial Intelligence*, 2010, pp.256–268.

[25] B. Bartsch-Sporl, M. Lenz, and A. Hubner, “Case-based reasoning: survey and future directions,” in *Proceedings of the 5th Biannual German Conference on Knowledge-Based Systems*, 1999, pp. 67–89.

[26] A. Aamodt and E. Plaza, “Case-based reasoning: foundational issues, methodological variations, and system approaches,” *Artificial Intelligence Communications*, vol.7, no.1, pp. 39–59, 1994.

[27] J. W. Murdock, “Structure mapping for jeopardy! Clues,” in *Proceedings of the International Conference on Case-based Reasoning*, 2011, pp.6–10.

[28] A. Abbott and I.Watson, “Ontology-aided product classification: a nearest neighbour approach,” in *Proceedings of the International Conference on Case-based Reasoning*, 2011, pp. 348–362.

[29] K. Bach, K. Althoff, R. Newo and A. Stahl, “A Case-based reasoning approach for providing machine diagnosis from service reports, case-based reasoning in research and development,” in *Proceedings of the 19th International Conference on Case-Based Reasoning*, 2011, pp. 363–377.

[30] E. L. Rissland, “AI and similarity,” *IEEE Intelligent Systems*, vol.21, no.3, pp.39–49, 2006.

[31] R. M. French, “The computational modeling of analogy-making,” *Trends in Cognitive Sciences*, vol.6, no.5, pp.200–205, 2002.

[32] R. Reichman, “Analogies in spontaneous discourse,” in *Proceedings of the 19th Annual Meeting of the Association for Computational Linguistics*, 1981, pp.63–69.

[33] P. Winston, “Learning by understanding analogies,” *Artificial Intelligence*, vol. 35, issue 1, pp.81–25, 1988.

[34] J. Marshall, “A self-watching model of analogy-making and perception,” *Journal of Experimental and Theoretical Artificial Intelligence*, 2006, vol.18, no. 3, pp. 267–307.

[35] K. Mirayala and M. Harandi, “Analogical approach to specification derivation,” in *Proceedings of the Fifth International Workshop on Software Specification and Design*, 1989, pp. 203–210.

[36] B. MacKellar and F. Maryanski, “Reasoning by analogy in knowledge base system,” In *Proceedings of the 4th International Conference on Data Engineering*, 1988, pp.242–249.

[37] K. K. Breitman, S. D. J. Barbosa, A. L. Furtado and M. A. Casanova , “Conceptual modeling by analogy and metaphor,” in *Proceedings of the 16th ACM Conference on Conference on Information and Knowledge Mmanagement*, 2007, pp.865–868.

[38] A. Eremeev and P. Varshavsky, “Methods and tools for reasoning by analogy in intelligent decision support Systems,” in *Proceedings of the International Conference on Dependability of Computer Systems*, 2007, pp.161–168.

[39] D. McSherry, “Case-based reasoning techniques for estimation,” *Colloquium on Case- Based Reasoning*, 1993, IEE Colloquium Digest No.1993/036, pp. 6/1-6/4.

[40] A. Gunes and G. Baydin, “Commonsense case-based reasoning via structure mapping: an application to mediation,” presented at International Conference on Case-Based Reasoning, 2011. Available at http://www. iiia.csic.es/~gunesbaydin/files/ICCBR2011DCPresentationBaydin.pdf.

[41] B. Falkenhainer, K.D. Forbus and D. Gentner, “The structure-mapping engine: algorithm and examples,” *Artificial Intelligence*, 1989, vol. 41, pp.1–63.

[42] G. Fauconnier and M. Turner, *The Way We Think: Conceptual Blending and the Mind's Hidden Complexities*, New York: Basic Books, 2002.

[43] G. Fauconnier and M. Turner, “The origin of language as a product of the evolution of double-scope blending,” *Behavioral and Brain Sciences*, vol. 31, pp.520–521, 2008.

[44] G. Fauconnier and M. Turner, “Rethinking metaphor,” *Cambridge Handbook of Metaphor and Thought*, vol.31, no. 1999, pp.1–32, 2008.

[45] M. Lee and J. Barnden, “A computational approach to conceptual blending within counterfactuals,” University of Birmingham, Cognitive Science Research Centre, 2001. Available at http://www.cs.bham.ac.uk/~mgl/test.pdf.

[46] J. Alexander, “Mathematical blending,” *Departments of Mathematics and Cognitive Science, Case Western Reserve University*, 2009, Available at SSRN: http://ssrn.com/abstract=1404844 or http://dx.doi.org/10.2139/ssrn.1404844.

[47] I. Manuel and D. Benyon, *Designing with Blends: Conceptual Foundations of Human-Computer Interaction and Software Engineering*, Cambridge: The MIT Press, 2007.

[48] J. Goguen and F. Harrell, "Foundations for active multimedia narrative: Semiotic spaces and structural blending,” *Interaction Studies: Social Behaviour and Communication in Biological and Artificial Systems*, 2004. Available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.62.9372.

[49] F. Pereira, *Creativity and Artificial Intelligence: A Conceptual Blending Approach*, Berlin: Mouton de Gruyter, 2007.

[50] T. Tan and K. Kwok, “Scenario generation using double scope blending,” *AAAI Fall Symposium*, 2009. Available at http://www.aaai.org/ocs/index.php/FSS/FSS09/paper/view/926/1213.

[51] B. E. Ozkan, “Autonomous agent-based simulation of a model simulating the human air-threat assessment process,” M.S. thesis, Naval Postgraduate School, Monterey, California, March 2004.

[52] K. Tan, “A multi-agent system for tracking the intent of surface contacts in ports and waterways,” M.S. thesis, Naval Postgraduate School, Monterey, California 2005.

[53] T. Tan, “Tactical plan generation software for maritime interdiction using conceptual blending theory,” M.S. thesis, Naval Postgraduate School, Monterey, California, 2007.

[54] P.Winston, “Learning and reasoning by analogy,” *Communications of the ACM*, 1980, vol. 23 no.12, pp.689–703.

[55] P. Buhlmann and A. J. Wyner, “Variable length Marov chains,” *The Annals of Statistics*, 1999, vol.27, no.2, pp.480–513.

[56] J.J. McGregor, “Backtrack search algorithms and the maximal common subgraph problem,” *Software-Practice and Experience*, 1982, vol.12, pp.23–34.

[57] L. Itti, C. Koch and E. Niebur, “A model of saliency-based visual attention for rapid scene analysis,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1998, vol.20, no.11, pp.1254–1259.

[58] P. Jungkunz and C. Darken, “A computational model for human eye-movements in military simulations,” *Computational and Mathematical Organization Theory*, 2011, vol.17, no.3, pp.229–250.

[59] S. F. Chen and J. Goodman, “An empirical study of smoothing techniques for language modeling,” *Computer Speech Language*, 1999, vol.13, pp.359–393.

[60] J. Undercoffer, A. Joshi, and J. Pinkston, “Modeling computer attacks: ontology for intrusion detection.” *Advances in Intrusion Detection*, vol.23, Issue 17, pp.113-135, 2003

[61] Valdes and K. Skinner, “Probabilistic alert correlation.” *Lecture Notes in Computer Science*, 2001, vol. 2212, pp.54–68.

[62] K. Julisch, “Clustering intrusion detection alarms to support root cause analysis.” *ACM Transactions on Information and System Security*, vol.6, no.4, pp.443–471, 2003.

[63] P. Ning, Y. Cui, and D. Reeves, “Constructing attack scenarios through correlation of intrusion alerts,” in *Proceedings of the 9th ACM Conference on Computer & Communications Security*, 2002, pp.245–254.

[64] F. Cuppens and A. Miege, “Alert correlation in a cooperative intrusion detection framework,” *In Proceedings of the IEEE Symposium on Security and Privacy, 2002*, pp. 202–215.

[65] S. Cheung, U. Lindqvist, and M. Fong, “An online adaptive approach to alert correlation,” in *Proceedings of the 7th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, 2010, pp.153–172.

[66] R. Sadoddin and A. Ghorbani, “Alert correlation survey: framework and techniques,” in *Proceedings of the International Conference on Privacy, Security and Trust*, 2006, pp.1–10.

[67] S. Sundaramurthy, L. Zomlot, and X. Ou, “Practical IDS alert correlation in the face of dynamic threats,” in *Proceedings of the International Conference on Security and Management*, 2011, available at http://130.203.133.150/viewdoc/summary?doi=10.1.1.218.535.

[68] S. J. Templeton and K. Levitt, “A requires/provides model for computer attacks,” *in Proceedings of the 2000 Workshop on New Security Paradigms*, 2000, pp. 31–38.

[69] X. Qin. “A probabilistic-based framework for INFOSEC alert correlation,” Ph.D. dissertation, Georgia Institute of Technology, Atlanta, Georgia, 2005.

[70] E. Frederick, N. Rowe and A. Wong, “Testing deception tactics in response to cyberattacks,” in *Proceedings of the National Symposium on Moving Target Research, Annapolis*, 2012, available at http://cps-vo.org/node/3711.

[71] C. Shannon, “A mathematical theory of communication, *Bell System Technical Journal*, vol.27, pp.379–423, 1948.

[72] F. Khong, “Performance assessment of network intrusion-alert prediction,” M.S. thesis, Naval Postgraduate School, Monterey, California, 2012.

[73] M. Galli, J. Turner, K. Olson, M.Mortensen, N. Wharton, E.Williams, T. Schmitz, M. Mangaran, G. Nachmani, K. Cheng, C. Goh, L. Ho, H. Meng, L. Leong, M. Lim, W. Mak, P. Sim, M. Ong, E. Pond, J. Sundram, B. Tan, K. Tan, C. Teng, T. Yow and W. Ong, “Riverine sustainment 2012,” M.S. thesis, Naval Postgraduate School, Monterey, California, 2007.

[74] M. Ames, C. Chan, K. Chng, A. Cole, D. Johnson, K. Kwai, K. Koh, H. Lim, T. Lim, Y. Liu, A. Marsh, C. McRoberts, C. Ng, C. Ng, M. Ng, H. Nguyen, L. Okruhlik, P. Oh, K. Ong, L. Peh, W. Soh, C. Tan, L. Toh, J. Torian and Y. Wong, "Port security strategy 2012," *M.S. thesis, Naval Postgraduate School*, Monterey, California , 2007.

[75] L. Schruben, "Simulation modeling with event graphs," *Communications of the ACM*, 1983, pp.957–963.

[76] A. Buss, P. Sanchez, "Building complex models with LEGOs (Listener Event Graph Objects)," in *Proceedings of the 2002 Winter Simulation Conference*, 2002, vol.2, pp.iii–xxiv.

[77] J. Zacks and J. Sargent, "Event perception: A theory and its application to clinical neuroscience," *Psychology of Learning and Motivation*, vol.53, pp. 253–299, 2009.

[78] J. Zacks, "Using movement and intentions to understand simple events," *Cognitive Science*, vol.28, pp.979–1008,2004.

[79] J. Magliano and J. Zacks, "The impact of continuity editing in narrative film on event segmentation," *Cognitive Science*, vol.35, Issue: 8, pp.1489–517, 2011.

[80] K. Swallow, D. Barch, D. Head, C. Maley, D. Holder, J. Zacks, "Changes in events alter how people remember recent information," *Journal of Cognitive Neuroscience*, vol.23, pp.1052–1064, 2012

[81] J. Reynolds, J. Zacks, T. Braver, "A computational model of event segmentation from perceptual prediction," *Cognitive Science*, vol.31, pp.613–64, 2007.

[82] J. Zacks, S. Kumar, R. Abrams and R. Mehta, "Using movement and intentions to understand human activity," *Cognition*, vol.112, pp. 201–216, 2009.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California