Search Problems in the Domain of Multiplication: Case Study on Robust Anomaly Detection using Markov Chains

Yisroel Mirsky
Aviad Cohen
Roni Stern
Ariel Felner
Lior Rokach
Yuval Elovici
Bracha Shapira
Department of Information Systems Engineering
Ben-Gurion University of the Negev
Beer Sheva, Israel
yisroel, aviadd, sternron, felner, liorkk, bshapira, yuval at *post.bgu.ac.il*

Abstract

Most work in heuristic search focused on path finding problems in which the cost of a path in the state space is the sum of its edges' weights. This paper addresses a different class of path finding problems in which the cost of a path is the product of its weights. We present reductions from different classes of multiplicative path finding problems to suitable classes of additive path finding problems. As a case study, we consider the problem of finding least and most probable paths in a Markov Chain, where path cost corresponds to the probability of traversing it. The importance of this problem is demonstrated in an anomaly detection application for cyberspace security. Three novel anomaly detection metrics for Markov Chains are presented, where computing these metrics require finding least and most probable paths. The underlying Markov Chain is dynamically changing, and so fast methods for computing least and most probable paths are needed. We propose such methods based on the proposed reductions and using heuristic search algorithms.

Keywords. Multiplication domain search, anomaly detection, search space monotonicity, Markov Chains, network intrusion detection

Introduction

Many optimization problems can be solved efficiently by formalizing them as a path finding problem in a (either explicitly of implicitly defined) graph. The most common problems found in the literature try to find paths of low cost, where a path's cost is computed as the sum of the weights of the paths' edges. In this paper we consider path finding problems in which the cost of a path is the product of its edge's weights. We call such problems *PROD problems*, as oppose to the regular *SUM problems*.

In particular, we study the problem of finding paths in a Markov Chain. A Markov Chain (MC) is a common model used to describe memory-less random processes. More concretely, an MC is a directed weighted graph where edge weights correspond to the probability of moving from one state to another (the sum of all outgoing edges from one state is equal to 1). The probability of following some trajectory (path) in a MC is the product of its edge's probabilities (weights). We study the problems of finding the *least-* and *most-probable* path in an MC.

This problem has application in many domains. One example is anomaly detection for cyberspace security when monitoring network traffic. Network traffic can be modeled by a MC (Pat 2007; Ye and others 2000) and an anomaly is detected when observing a transition with low probability in the MC. In some cases, anomalies are better detected when the probability of k sequential transitions are considered (i.e. the product of the edge costs along the last k transitions made). However, when k is very large, the probability of any k transitions becomes exponentially small – making it difficult to distinguish between likely transition sequences and anomalies. By knowing the cost of the most probable paths (i.e., MAX) and least probable paths (i.e., MIN) in such a scenario, the achieved probabilities can be placed in their context, thereby assisting in labeling their likelihood. We propose three anomaly score metrics for doing so and demonstrate their effectiveness.

In our application, the MC dynamically changes over time, and thus an efficient algorithm for finding the least and most probable k-step trajectory is needed. One approach is to enumerate all possible k-step paths. Clearly, this is not practical for large MCs and large values of k. Thus, fast methods for finding least and most probable path in MC is required.

The first contribution of this work is in formalizing these problems as minimization or maximization PROD problems. The second contribution of this work is in determining which existing algorithms can be used to solve PROD problems. This is done by analyzing the symmetry between PROD and SUM problems in both maximization (MAX) problems and minimization of problems (MIN). Then, we discuss when and how PROD problems can be translated to SUM problems.

The third contribution is in introducing, as a case study, PROD problems that arise in the context of anomaly detec-

tion using Markov Chains. We evaluate the comparison of known search algorithms in this domain. Our results show that the reduction of the PROD domain to SUM not only opens the doors to many new theoretical questions, but also has practical implications.

As a secondary contribution, we also propose three novel anomaly detection metrics, based on computing least and most probable paths, and show their impact on improving anomaly detection accuracy.

Background: MIN and MAX problems

Most work on solving path finding problems with search algorithms were on domains having two underlying assumptions. The first assumption is that the cost of a path is the sum of the weights (or other costs) along its edges. The second assumption is that lower cost path are preferred, and an optimal solution is a path of minimum cost. Problems with the first assumption are referred to as SUM problems, and problems with the second assumption are referred to as MIN problems. Algorithms such as Dijkstra's algorithm and A* are perfectly suitable for MIN-SUM problems.

Recently, attention has been raised to the converse setting, where one wishes to find a path of maximum reward (Stern et al. 2014). Such problems are called MAX problems. Stern et al. studied the behavior of classic search algorithms on MAX problems. They observed that popular search algorithms such as Dijkstra's algorithm and A* loose their effectiveness or optimality guarantees when applied to some MAX problems.

Stern et al. (2014) suggested the notion of *search space monotonicity* as a class of problems that can be solved by classical search algorithms.

Definition 1 (Search Space Monotonicity) *A search space is said to be* monotone *w.r.t. a start state s if for any path P that starts at s it holds that P is no better than any of its prefixes, where better is defined w.r.t. the objective function (MAX/MIN).*

Based on this we add the following inverse definition:

Definition 2 (Inversely Monotone Search Space) *A search space is said to be* inversely monotone *w.r.t a start state s if for any path P that starts from s it holds that P is not worse than any of its prefixes, where worse is defined w.r.t the objective function (MAX/MIN).*

Monotonicity is required for several key properties of known search algorithms. For example, in a monotone search space, A^* can halt when a goal node is chosen for expansion. This is not necessarily the case in inversely monotone search spaces. For example, in the longest simple path problem, expanding a goal does not guarantee that the longest path to it has been found since an even longer simple path to a goal may exists elsewhere. Adapting existing search algorithms to domains that are inversely monotone is possible, but may cause the search to be slower. See Stern et al. (2014) for details on how to perform these search algorithms adaptations.

Note that some problems may be not monotone and not inversely monotone. For example, a shortest path problem in a graph which also includes negative-weight edges. We do not consider these cases in this work.

Cost Algebra

A very related work that predates the work of Stern et al. (2014) is the work of Edelkamp et al. (2005) on *cost algebra*. *Cost algebra* (CA) is a general way to express solution costs in search problems. CA allows for other non-trivial meanings to the $+$ and $>$ operators that appear when accumulating costs and when comparing them, respectively. The benefit of formulating a search problem as a CA, is that Edelkamp et al. also provided *cost algebraic* versions of Dijkstra's Algorithm (DA) (Dijkstra 1959) and A* (Hart, Nilsson, and Raphael 1968) suitable for any problem that can be represented as a CA. These algorithms generalize the common, well-known implementations of DA and A* for the shortest-path problem, by again allowing other meanings to the $+$ and $>$ operators that appear in the relevant pseudo code.

The exact relation between search space monotonicity and cost algebra is not known. However, Stern et al. (2014) showed that some search spaces that are inversely monotone cannot be formulated as a cost algebra. An example of a such a problem is the longest simple path problem.

From SUM to PROD

Up until now the assumption was that the costs (in MIN problems) and rewards (in MAX problems) are additive. These are called SUM problems (MIN-SUM, and MAX-SUM, respectively). We now move to problems where the costs/rewards are not additive, but rather multiplicative. These are denoted as product problems (PROD problems). More formally, let $G = (V, E, w)$ be a graph representing a state space of a given problem, where $w(u, v)$ is the weight of edge $(u, v) \in E$. The cost of a path P from a start state $s \in V$ to some state $n \in V$ is denoted by $cost(n)$ and its definition depends on the nature of problem. In SUM, the cost of a path P is defined as

$$cost(n) = \sum_{i=1}^{|P|-1} w(P[i], P[i+1]) \quad (1)$$

whereas in PROD, the cost is defined as

$$cost(n) = \prod_{i=1}^{|P|-1} w(P[i], P[i+1]) \quad (2)$$

The optimal solution for a SUM problem and for a PROD problem can be profoundly different, even for the same state space graph. As illustrated in figure 1, the shortest path in this MIN problem is different when considering SUM or PROD (equations 1 and 2). With SUM, $10 + 10 = 100$ is the minimum. However with PROD, $6 \times 15 = 90$ is the minimum.

There are four classes of problems: MAX-SUM, MAX-PROD, MIN-SUM, and MIN-PROD. In addition, for PROD problems it makes a difference whether the edges' costs are in the range $[0, 1]$ or in $[1, \infty]$. The table in figure 2 summarizes which classes of problems are *monotone* and which are

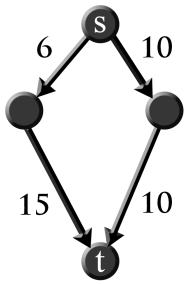

Figure 1: An example where MIN-SUM has a different optimum path than MIN-PROD

	Domain	MIN	MAX
SUM	[0,1]	MIN-SUM (CA) Use a standard algorithm	MAX-SUM Run until the open list is empty (Stern et al.)
SUM	$[1,\infty]$		
PROD	[0,1]	Transform problem to MAX-SUM: 1. $w' \leftarrow log(w)$ 2. $w' \leftarrow (-1) * w'$	Transform problem to MIN-SUM: 1. $w' \leftarrow log(w)$ 2. $w' \leftarrow (-1) * w'$
PROD	$[1,\infty]$	Transform problem to MIN-SUM: $w' \leftarrow log(w)$	Transform problem to MAX-SUM: $w' \leftarrow log(w)$

Figure 2: *monotone* (white) and *inversely monotone* (grey) PROD problems and their reduction to the equivalent SUM problem.

inversely monotone, depending on the range of their edges' weights. The cells with a white background are monotone while the cells with a grey background are inverse monotone. As mentioned above, this distinction has an impact on which search algorithms to apply and how to apply them.

The SUM cells (top) were already described by Stern et al.(2014). They showed that MIN-SUM is monotone while MAX-SUM is inverse monotone, regardless of whether edge weights are smaller or larger than one (as long as the edge weights are non-negative).

PROD problems

In order to classify PROD problems, we reduce each PROD problem either to a MAX-SUM or MIN-SUM problem. The first step of these reductions is to provide a transformation of the PROD problem to a SUM problem. For this, we take the *logarithm* of the weights of the edges (denoted $log(w)$). The second step is to prove that the transformation is a proper reduction. This means showing that the optimality of the solution holds across the transformation. We will do this separately for each cell in the Table in figure 2.

Case 1: We start with bottom left cell: MIN-PROD$_{[1,\infty]}$ (i.e., when edges' weights are ≥ 1). A problem P in MIN-PROD$_{[1,\infty]}$ can be transformed to a problem P' in MIN-SUM$_{[0,\infty]}$ as follows. Each weight w in P is transformed to $log(w)$ in P'. This is clearly a transformation. For the reduction step it is enough to prove that:

$$(w_1 \times w_2 < w_3 \times w_4) \Rightarrow (log(w_1) + log(w_2) < log(w_3) + log(w_4)) \quad (3)$$

Since all weights are larger than 1, their *log* will always be positive, and due to the basic attributes of *log*, Equation 3 holds true. Therefore, MIN-PROD $_{[1,\infty]}$ is monotone.

Case 2: Next we move to the bottom right cell: MAX-PROD$_{[1,\infty]}$. We transform each problem P in MAX-PROD$_{[1,\infty]}$ to a problem P' in MAX-SUM$_{[0,\infty]}$. The reduction is identical to the former case as the relation $<$ is robust to the transformation. However, since now we prefer paths with a larger objective function, the problem is reduced to MAX-SUM (and not MIN-SUM). Therefore, MAX-PROD$_{[1,\infty]}$ is inversely monotone and can be solved with methods designed for MAX-SUM.

Case 3: We now move to the next case: MAX-PROD$_{[0,1]}$ (middle, right). We note that the $log(W)$ for $0 < W < 1$ is negative. Therefore, we transform each problem P in MAX-PROD$_{[0,1]}$ to a problem P' in MIN-SUM$_{[0,\infty]}$ as follows. Each weight w in P is transformed to $-log(w)$ in P'. For the reduction step it is enough to prove that:

$$(w_1 \times w_2 > w_3 \times w_4) \Rightarrow -(log(w_1) + log(w_2)) < -(log(w_3) + log(w_4)) \quad (4)$$

Since all weights are less than 1, their *log* is negative, and due to the basic attributes of *log*, equation 4 holds true. Therefore, MAX-PROD$_{[0,1]}$ is monotone.

Case 4: Finally we move to the case of MIN-PROD$_{[0,1]}$. Following a similar process for the reduction of MAX-PROD$_{[1,\infty]}$ to MAX-SUM$_{[0,\infty]}$ we transform w in P to $-log(w)$ in P'. Based on the same line of proof it can be seen that indeed MIN-PROD$_{[0,1]}$ is inversely monotone and can be solved with methods designed for MAX-SUM.

We now have proven that the reduction table in figure 2 can be used to determine what type of algorithm can be used in different PROD problems. For example, MAX-PROD$_{[0,1]}$ is *monotone* and therefore, by taking the logarithm of its weights, it can be solved using a MIN-SUM algorithm such as the traditional A*.

Since log is not defined for negative numbers, and since negative numbers do not apply to this paper's example application (MCs), we defer the discussion of dealing with negative edge weights to future research. Furthermore, we omit the PROD problems whose weights can be both greater than and less than 1. This is because their log reduction is equivalent to a SUM problem whose weights consist of both positive and negative values. Such problems are problematic due to possible loops with negative sum which can infinitely improve the path cost.

Case Study: Pathfinding in Markov-Chains

As a case study for PROD problems, consider the problems of finding the most/least probable paths in a Markov Chain.

As mentioned earlier, a Markov chain (MC) is a weighted directed graph $G = (V, E, w)$ that models a discrete-time

stochastic process. Vertices in this graph are states, where a state is some event or temporal status of the modeled application. Edges represent possible transitions between states and the weight of an edge (u, v) represents the probability of transitioning to from state u to v in a single step. A MC model is a *memory-less process*, i.e., a process where the probability of transition at time t only depends the state at time t and not on any of the states leading up that state.

Typically, an MC is represented as an adjacent matrix M such that M_{ij} stores the probability of transitioning from state i to state j at any given time t. Formally, if X_t is the random variable representing the state at time t then

$$M_{ij} = Pr(X_{t+1} = j | X_t = i) \quad (5)$$

Finding the *least-* and *most-probable* paths in MCs is exactly MIN-PROD and MAX-PROD problems, respectively. To motivate solving these problems, we briefly describe our anomaly detection application and how finding the least and most probable paths in an MC is needed.

Anomaly Detection Using Markov Chains

Anomaly detection with MCs have been proposed in several domains (Ye and others 2000; Jha, Tan, and Maxion 2001; Meng et al. 2006). We consider a specific type of anomaly detection problem, where the task is to identify harmful network activity. Such activity can be caused, for example, by a malicious agent trying to exploit some vulnerability of network protocols to gain unlawful access to private information. A challenging aspect of our anomaly detection problem is that it is *unsupervised*, in the sense that we try to detect malicious behaviors not known a priori, i.e., we do not know how such anomalies manifest in the network packet data.

The high-level approach for such unsupervised anomaly detection using MCs consists of three stages: *training*, *tracking*, and *classifying*. In the training stage, an MC for the *normal* behavior is learned from past data. In the tracking stage, the last k observed state transitions are tracked to form *observed trajectories* (Pat 2007). An observed trajectory P_{obs} is an observed sequence of state transitions, i.e., a path in the MC, and k is a user defined parameter. In the classifying stage, an observed trajectory is analyzed and classified as an anomaly or not.

The classifying stage requires an *anomaly metric*. An anomaly metric is a score given to an observed trajectory to quantify the degree to which that trajectory is anomalous. An observed trajectory P_{obs} is classified as anomalous if the measured anomaly score metric for P_{obs} is lower than some threshold T.

Algorithm 1 summarizes the anomaly detection process described above. It accepts two parameters k and T, where k is the length of the observed trajectories and T is the described above threshold value. *Metric(P_{obs})* denotes the measured anomaly metric for the observed trajectory (P_{obs}). If it is smaller than T, then P_{obs} is considered as an anomaly.

Metrics for Anomaly Detection

The simplest anomaly score metric is the probability of the observed trajectory $P_{obs} = (s_0, \ldots, s_t)$ w.r.t the given MC.

Algorithm 1 Detection of anomalies in MCs

Input: k, *the length of trajectory considered*
Input: T, *the anomaly metric threshold*

```
1: P_obs ← initFIFO(k)
2: while true do
3:    X_t ← getCurrentState()
4:    P_obs ← Add(P_obs, X_t)
5:    score ← Metric(P_obs)
6:    if score > T then
7:       alarm()
8:    end if
9: end while
```

This is given by

$$Pr(P_{obs}) = Pr(\bigwedge_{i=0}^{t}(X_i = s_i)) = \prod_{i=0}^{t-1} M_{s_i, s_{i+1}} \quad (6)$$

(Ye, Zhang, and Borror 2004) discuss the use of MCs as a means for anomaly detection in the application of cyber-attack detection. Nevertheless, they show how using the probability $Pr(P_{obs})$ as an anomaly score metric is not robust due to the presence of noise (rare yet benign sequences of observations). For example, consider the case where one single transition in P_{obs} has a zero probability w.r.t. the MC. In this case, $Pr(P_{obs})$ would report a zero probability not for just one time-tick, but rather k time-ticks. This means that the metric will report a score of 0 (anomaly) even $t + k - 1$ time ticks after the event. Still, $Pr(P_{obs})$ with a $k > 1$ will be better than just considering $k = 1$, however it is not robust when short anomalies or noise occur.

Next, we define two anomaly score metrics that are more robust than only considering $Pr(P_{obs})$, and another metric which is good for only certain scenarios as will be discussed further on. Computing these metrics will require solving PROD problems.

Metric 1 Let $P_{obs}[1]$, $P_{obs}[i]$, and $P_{obs}[k]$ denote the first, i^{th}, and last state observed. The first anomaly score metric we propose is the probability of the most probable k-hop trajectory from $P_{obs}[1]$ to $P_{obs}[k]$ (denoted by $P_{\max}(P_{obs}[k])$). This metric, denoted $Metric1$, is defined as follows:

$$\text{Metric1}(P_{obs}) = P_{\max}(P_{obs}[k]) \quad (7)$$

This metric might seem surprising, as it ignores the probability of the observed trajectory, and only considers the most probable path to reach the end state of the observed trajectory. However, as shown in the experimental results, the performance of Metric1 is surprisingly good. Computing Metric1(P_{obs}) requires solving a single PROD problem of finding the most probable k-step trajectory path from $P_{obs}[1]$ to $P_{obs}[k]$ ($= P_{\max}(P_{obs}[k])$). We call denote this problem as **Problem 1**.

Metric 2 The second anomaly score metric, referred to as Metric2, serves as default choice metric when performance feedback is unavailable (i.e. when feedback about the best

T is unavailable). This is achieved by considering also the least-probable k-hop trajectory from $P_{obs}[1]$ to $P_{obs}[k]$ (denoted by $P_{\min}(P_{obs}[k])$). Metric2 is the probability of the observed trajectory normalized by the most and least probable path that reaches the same state. Formally, Metric2 is computed as follows:

$$\text{Metric2}(P_{obs}) = \frac{Pr(P_{obs}) - P_{\min}(P_{obs}[k])}{P_{\max}(P_{obs}[k]) - P_{\min}(P_{obs}[k])} \quad (8)$$

Computing Metric2 requires solving two search problems: Problem 1 (as defined above), and the problem finding the least probable k-hop path from $P_{obs}[1]$ to $P_{obs}[k]$ (=$P_{\min}(P_{obs}[k])$). We denote the latter problem as **Problem 2**.

Metric 3 The third anomaly score metric we propose, referred to as Metric3, considers *all* most-probable k-step trajectories that start from $P_{obs}[1]$. Metric3 is computed as follows. Let $S_k(P_{obs}[1])$ denote the set of all states that are reachable with a k-step trajectory from $P_{obs}[1]$. For every state $s_i \in S_k(P_{obs}[1])$, we compute the most probable path from $P_{obs}[1]$ to s_i, denoted by $P_{max}(s_i)$. Then, we consider the values of $PDF_{max}(s_i)$ as a probability density function and compute the corresponding cumulative distribution function

$$CDF_{max}(v) = sum(\{P_{max}(s_i) | s_i \in S_k(P_{obs}[1]) s.t. P_{max}(s_i) \leq v\}) \quad (9)$$

In words, $CDF_{max}(v)$ represents the sum of probabilities over all states s_i for which $P_{max}(s_i) \leq v$. Thus, having a high $CDF_{max}(Pr(P_{obs}))$ suggests that P_{obs} is normal. Metric3 combines this information with Metric1 as follows.[1]

$$\text{Metric3}(P_{obs}) = \text{Metric1}(P_{obs}) + \text{CDF}_{max}(Pr(P_{obs})) \quad (10)$$

Computing Metric3 is much more demanding than Metric1, as it requires finding the most probable k-hop from $P_{obs}[1]$ to each of the states in $S_k(P_{obs}[1])$. We refer to this problem as **Problem 3**.

With regards to robustness, Metrics 1 and 3 are more robust w.r.t just using the basic metric $Pr(P_{obs})$. This is because the output of Metric1 is unaffected by a short low probable trajectories (observations), since Metric1's scores are based on the most probable path in the given MC. In applications such as network intrusion detection, the MCs are well-connected (can get from every state to every state). Therefore, scores as low as 0 will not to occur. Furthermore, the metric calculated after the anomalous event (up to $t + k - 1$ time tick afterwards) will not be affected by the low probable transitions in the middle of the observed trajectory.

Underlying Search Problems

Computing the three anomaly score metrics presented above requires solving a set of PROD problems: Problem 1, Problem 2, and Problem 3. Problem 1 is a MAX-PROD$_{[0,1]}$ problem, and thus can be reduced to a standard MIN-SUM problem where all search algorithms work well. Problem 2 is a MIN-PROD$_{[0,1]}$ problem, and thus is not reducable to MIN-SUM problems but to MAX-SUM problems. Thus, Problem 2 is more challenging computationally and require the adaptations of some search algorithms, as outlined by Stern et al. (2014). Finally, Problem 3 is a set of MAX-PROD$_{[0,1]}$ problems, which we solved sequentially.

Note that solving any of these problems by enumerating all k-hop paths is impractical for large k, as thus using search algorithms is needed.

One might consider computing the most- and least-probable k-step trajectories for all states upfront. However, for our application the matrix M is constantly updated with the observed trajectories since network traffic behavior changes over time. In fact, we update the MC every time that a transition occurs between a state s_i and a state s_j then M_{ik} (for every k including $k = i$ as we allow self loops) are updated according to the new transition distribution seen thus far. Thus, we need a fast method to compute the least and most probable k-steps paths (trajectories).

Before running any search algorithm, we performed the reduction of our problem to MIN-SUM$_{[0,\infty]}$ and MAX-SUM$_{[0,\infty]}$ accordingly, as explained above. Following the reduction, we considered the use of three traditional search algorithms: breadth first search (BFS), uniform cost search (UCS), and A*.[2]

Search Space

All paths that need to be considered in the MC for solving all problems have at most k steps in them, and may contain loops (i.e., repeated states) and even self loops (i.e., an edges from a state to itself). To handle these problem attributes, we formalized the search tree as follows.

A node in the search tree is a tuple $\langle v, d \rangle$ where v is a state in the MC and d is a the depth in the search tree. The initial node is $\langle P_{obs}[0], 0 \rangle$. The successors of each node $\langle i, d \rangle$ are all possible transitions from state i and they are denoted as $\langle j, d+1 \rangle$ where j is a state reachable from i (i.e. $M_{ij} \neq 0$). We keep the depth of the node in the search tree as an inherent part of the node definition to enable duplicate pruning of states of the MC revisited at the same depth in the search tree. This is done to accommodate the possibility of self loop edges.

The goal node is defined according to the two problem definitions given above. For Problem 1 and Problem 2 the goal node is $\langle P_{obs}[k], k \rangle$, and for Problem 3 the goal nodes are the set $G = \{\langle v, k \rangle | v \in S_k(P_{obs}[1])\}$. We note that if the goal state $P_{obs}[k]$ was reached, but it was not at depth k, then search continues (e.g., with self loops).

The cost function g is given according to Equation 1 (after the reduction). UCS expands nodes according to their g-value. Unlike UCS, A* also considers a heuristic estimate $h(n)$ as to the cost left between node n and the goal node, such that $f(n) = g(n) + h(n)$. The topic of heuristics in

[1] Metric3 uses Metric1 and not Metric2 because, as shown below, Metric1 was superior to Metric2 in our experimental results.

[2] Note that UCS is a best-first search implementation of Disjktra's algorithm (Felner 2011).

MCs is rather unexplored. Since our problem requires that we find optimal paths consisting of exactly k-steps, we propose the heuristic:

$$h(n) = w_{\min} \cdot (k - i) \tag{11}$$

where $w_{\min}$ is the smallest edge weight in M_t (after the reduction), and i is the number of hops taken so far from the initial state to n. We note that this heuristic is admissible since it never overestimates the cost of reaching the goal.

Note that this problem before the reduction is equivalent to having the search algorithm modified to select the node with the *largest* $f(n) = g(n) \cdot h(n)$ where $g(n)$ is equation 2, and $h(n) = (w_{\max})^{k-i}$.

Experimental Results

In this section we first describe the network intrusion dataset used and how the MC was generated from it. Afterwards we evaluate the results of BFS, UCS and A* as they pertain to problems 1 and 3. Lastly, we evaluate and discuss the results of proposed anomaly metrics (equations 7, 10, and 8) on the network intrusion MC.

The Network Intrusion Dataset

We used the KDD'99 cup network intrusion dataset in our evaluations (Lichman 2013). The dataset is a time-series consisting of several million observations of network connections inside a simulated military network. Each observation has a number of features ranging from connection time to protocol specific settings. The observations are paired with one of 23 possible labels that indicate the class of the observation; *normal* or one 22 possible attacks. Since our application is anomaly detection, we considered all attack types as single label. Therefore, each observation had the label *normal* or *anomalous*, where approximately 40% of the observations were *anomalous*.

In our evaluation, we divided the dataset into two sets; the training set and test set. The training set consisted of the first 25% instances of the dataset. To simulate an unsupervised anomaly detection setting, we removed the observations labeled *anomalous* from the training set, and only the *normal* instances were used to model the MC. The test set was the last 75% of the dataset, and was used to evaluate the anomaly score metric w.r.t. the trained MC.

The first step we took to generate the MC on the training set was to define what a *state* is in the MC. Since anomaly detection is an unsupervised method of machine learning, we chose to detect the states using a stream clustering algorithm called *pcStream* (Mirsky et al. 2015). pcStream dynamically detects contexts (clusters) in numeric data-streams, assigning observations to one of the known contexts (based on statistical similarity measures). Simply put, pcStream detects the state of the time-series so that we can track its transitions.

Now that we were able to track the state of the network (during a chronological pass over the training set) we generated our MC using an Extensible Markov Model (EMM) (Dunham, Meng, and Huang 2004). An EMM counts the number of transitions which occur from state i to state j (denoted n_{ij}). From here, the MC can be obtained by

$$M_{ij} = \frac{n_{ij}}{n_i}$$

where n_i is the total number of out-going transitions observed by state i.

The final MC generated from our training set consisted of 2122 states.

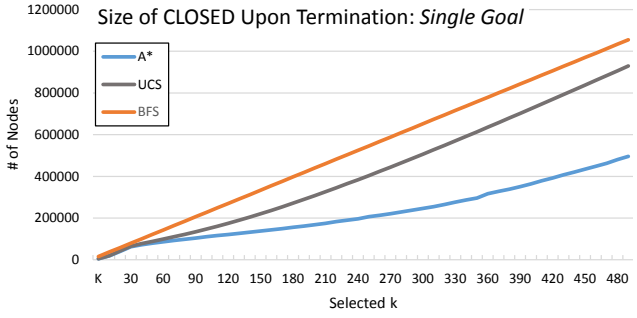

Figure 3: The number of expanded nodes for BFS, UCS, and A* for Problem 1 as a function of k

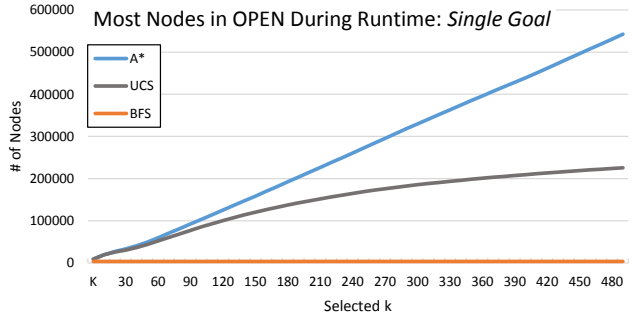

Figure 4: The largest size of OPEN over the duration of solving *problem 1*

Search Performance in a Network Intrusion MC

In the application described above, the transition probabilities of the EMM (our MC) change after each time tic. Therefore we do not consider preprocessing as a means of improving the search time.

We experimented from various states in the MC (obtained above). We repeated this many times, each time with a larger k. We also attempted to use a version of DFS but even for very small values of k (3-4) it proved impractical due to the lack of duplicate detection. This is why we included breadth-first search (BFS) (to help contrast A* and UCS's performance). All code was written in Java and run on a single core of an Intel i5-3470 CPU.

Results for Problem 1

The number of nodes expanded for BFS, UCS, and A* for the most probable k-hop single source to the single target problem (*problem 1*), can be found in figure 3. Each trial

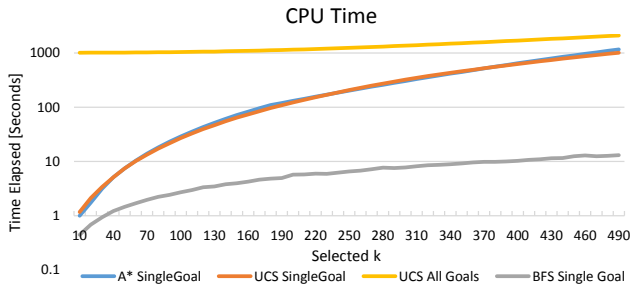

Figure 5: Runtimes for BFS, UCS, and A* for both problems 1 and 3

used a different k and the same network intrusion dataset MC. The figure clearly shows that A* outperforms UCS for all k in terms of nodes expanded.

Figure 5 shows the CPU time in seconds. Surprisingly, here A* was slower than UCS. The explanation to this relies on Fig. 4 which shows the largest number of nodes that were concurrently present in OPEN for the different k values. Clearly, A* had a larger OPEN. Thus, the time overhead was larger despite the fact that fewer nodes were expanded. This is an interesting phenomenon which may appear in many circumstances. This subject requires further research to understand the circumstances under which this phenomenon occurs.

Similarly, BFS's OPEN is relatively small, making its overhead negligible. Therefore, although it expands far more nodes than UCS and A*, it runs much faster. Moreover, we did not implement the priority queue of UCS and A*'s OPEN in the most efficient way since our focus was on the number of nodes expanded by the respective algorithms. Therefore, an algorithm such as A* would be preferred to solve such problems when dealing with MCs with more than 2122 states.

Results for Problem 3

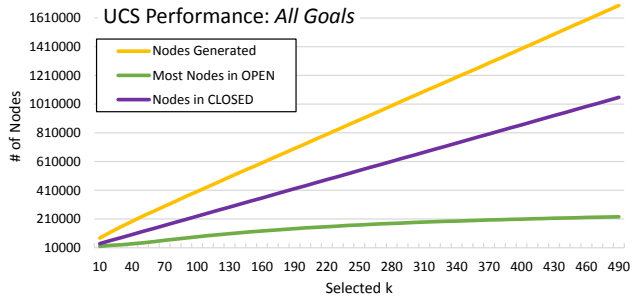

Figure 6: Results for the single source all targets problem using UCS only.

Figures 6 and 5 show the number of nodes expanded and the CPU time for the problem of multiple goals (*problem 3*). Here, only UCS is used. In order to reach all goals, regardless of k, an additional amount of time (nearly constant) was required in comparison with the single goal search.

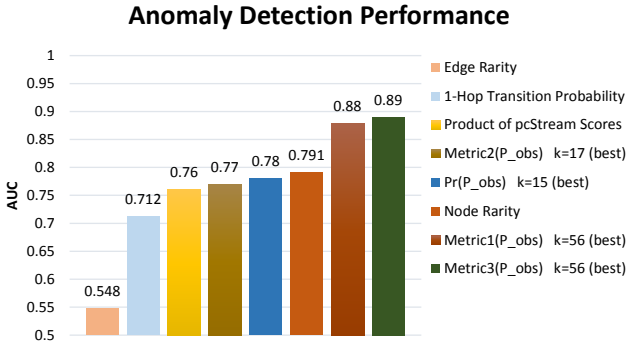

Figure 7: The performance (in terms of AUC) of the different anomaly detection scoring methods using the MC that was trained on clean network traffic

Anomaly Detection Performance

The performance of an anomaly detection algorithm can be measured by its true positive rate (TPR) and false positive rate (FPR), where the TPR is calculated as:

$$\frac{\#\text{True Positives}}{\#\text{True Positives} + \#\text{False Negatives}}$$

and the FPR is calculated as:

$$\frac{\#\text{False Positives}}{\#\text{False Positives} + \#\text{True Negatives}}$$

Typically, an anomaly detection algorithm will assign a score to each observation that indicates to what degree the observation is anomalous. Changing the threshold sensitivity T (i.e. scores above the threshold indicates an anomaly) produces different a TPR and FPR. The plot of these normalized values is called the Receiver Operating Characteristic (ROC). The area under this curve (AUC) gives the probability that the anomaly detection algorithm will rank a randomly chosen anomalous instance higher than a randomly chosen normal one. Therefore, an achieved AUC of $\frac{1}{2}$ indicates a random classifier (the worst), and an achieved AUC of 1 indicates a perfect classifier (the best). We use the AUC as our evaluation metric.

To evaluate the proposed anomaly score metrics (equations 7,10, and 8) we did the following to each observation o in the set (chronologically). First we used the trained pcStream model to determine what X_t is from o w.r.t. the trained MC. Afterwards, we computed metrics, and assigned the scores to o. Once all observations were scored, the AUCs were computed (for each metric). We repeated this for various sizes of k.

We found that the logarithm of the paths costs in $S_k(P_{obs}[1])$ conformed to a Gamma distribution. Therefore, we fitted the PDF from 9 used in Metric3 accordingly. Note that since a Gamma distribution is defined for positive values only (and the log of values on the domain $[0, 1]$ are negative), we fit the distribution to the absolute values of the $logged$ costs, and then computed the CDF as $1 - CDF$.

Figure 7 shows the proposed anomaly scoring methods compared to other EMM anomaly scoring methods (Meng

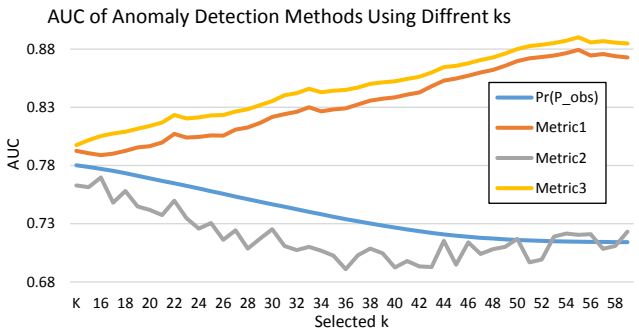

Figure 8: The impact the selection of k has on the anomaly scoring methods that depend on k

et al. 2006): Normalized Transition Probability (edge rarity) $\frac{n_{ij}}{\sum n_i}$, and Occurrence Frequency (node rarity) $\frac{n_j}{\sum n_i}$. Moreover, we also include the commonly used transition probability $Pr(P_{obs})$ from equation 6, and the product of each observation's cluster similarity score from pcStream as well.

Figure 8 shows the performance of Metrics 1 and 3 when selecting various ks in relation to $Pr(P_{obs})$. The results show that both Metrics 1 and 3 provide better anomaly scores then the other methods. This is due to their added robustness.

Metric3 achieves a slightly better AUC than Metric1 (on this dataset). However, the runtime of Metric1 is much shorter than Metric3. This is because Metric1 only needs to perform the single goal search (*problem 1*) to find the **most** probable path. In contrast, Metric3 requires solving the all goal problem (problem 3) for both **most** probable paths. For future work, we intend on testing the proposed anomaly detection methods on other datasets and MCs from other domains.

Conclusion and Future Work

In this work we studied the properties of PROD problems, where the cost of a path is the product of the weights of its constituent edges. A complete map is given for reducing MAX-PROD and MIN-PROD problems to MIN-SUM and MAX-SUM problems (see Figure 2). This enables solving PROD problems with the suitable search algorithm. Next, we described specific MIN-PROD and MAX-PROD problems: finding the least probable and the most probable path in a Markov Chain. These problems are important building blocks for designing an anomaly detection system, specifically for cyberspace security in network intrusion. Different search algorithms were evaluated for solving these problems and their performance was analyzed.

To the best our knowledge, very little attention has been given to the full scope of PROD problems and in this work we aimed to start closing this gap. However, much theoretical and practical research is needed. On the theoretical side, it is still not clear how the classification provided in Figure 2 and the definitions of monotone and inversely monotone relates to key search space properties such as the principle of optimality. Another open question is which PROD problems can and cannot be formulated as cost algebra.

Acknowledgments

This research was supported by the Ministry of Science and Technology of Israel, as well as the Israel Science Foundation (ISF) under grant #417/13 to Ariel Felner.

References

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische mathematik* 1(1):269–271.

Dunham, M. H.; Meng, Y.; and Huang, J. 2004. Extensible markov model. In *Data Mining, 2004. ICDM'04. Fourth IEEE International Conference on*, 371–374. IEEE.

Edelkamp, S.; Jabbar, S.; and Lluch-Lafuente, A. 2005. Cost-algebraic heuristic search. In *AAAI*, 1362–1367.

Felner, A. 2011. Position paper: Dijkstra's algorithm versus uniform cost search or a case against dijkstra's algorithm. In *SOCS*, 47–51.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* SSC-4(2):100–107.

Jha, S.; Tan, K.; and Maxion, R. A. 2001. Markov chains, classifiers, and intrusion detection. In *IEEE Workshop on Computer Security Foundations*, CSFW.

Lichman, M. 2013. UCI machine learning repository.

Meng, Y.; Dunham, M. H.; Marchetti, M. F.; and Huang, J. 2006. Rare event detection in a spatiotemporal environment. In *IEEE International Conference on Granular Computing (GrC)*, 629–634.

Mirsky, Y.; Shapira, B.; Rokach, L.; and Elovici, Y. 2015. pcstream: A stream clustering algorithm for dynamically detecting and managing temporal contexts. PAKDD.

2007. An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer Networks* 51(12):3448–3470.

Stern, R.; Kiesel, S.; Puzis, R.; Felner, A.; and Ruml, W. 2014. Max is more than min: Solving maximization problems with heuristic search. In *Symposium on Combinatorial Search (SoCS)*.

Ye, N., et al. 2000. A markov chain model of temporal behavior for anomaly detection. In *Proceedings of the 2000 IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop*, volume 166, 169. West Point, NY.

Ye, N.; Zhang, Y.; and Borror, C. M. 2004. Robustness of the markov-chain model for cyber-attack detection. *Reliability, IEEE Transactions on* 53(1):116–123.