Efficient High Quality Plan Exploration for Network Security

Anton V. Riabov and **Shirin Sohrabi** and **Octavian Udrea** and **Oktie Hassanzadeh**
IBM T.J. Watson Research Center
1101 Kitchawan Rd, Yorktown Heights, NY 10598, USA
{riabov, ssohrab, udrea, hassanzadeh}@us.ibm.com

Abstract

We consider the application of planning in network security. In this application, plans represent possible attacks on the network, and network administrators need tools that would allow them to explore the plan space quickly and efficiently. Multiple aspects of this problem require generating and inspecting more than one plan, primarily due to limited information about the possible actions of the attacker, and a variety of possible attacks. This problem can be modeled as diverse planning, with the caveat that high quality (or, equivalently, low cost) plans must be prioritized, since those plans typically represent the most efficient attacks that are of highest importance to the administrators. Hence, there is a need for a systematic approach to finding such plans. We propose a new technique based on a top-k planner that finds k optimal or near-optimal plans, followed by plan consolidation, for generating diverse high quality plans. Comparing to existing diverse planners, we show that it is able to meet the high quality and plan diversity requirements efficiently, and therefore we can recommend it for this application.

1 Introduction

Multiple security-related challenges arising in administration and management of computer networks have been successfully tackled using classical planning in prototypes and production systems. In this paper, we propose to address a specific requirement of finding relevant but sufficiently different attacks, which we believe to be common for all proposed planning-based systems, by developing an efficient technique that allows A^*-based planners to produce a diverse set of high quality plans.

In network security applications of planning, domain models are developed by capturing and formalizing expert knowledge. Domain models include the actions of attacker, such as network scans and exploitation of vulnerabilities to infect network hosts with malware, possibly complemented by actions representing actions of users that expose new vulnerabilities, actions that help account for network connectivity, installed software, operating system and other configuration [Boddy *et al.*, 2005; Roberts *et al.*, 2011; Lucngeli *et al.*, 2010]. The individual valid plans then represent possible attacks, and the space of valid plans represents attack graphs, therefore allowing network administrators to explore and analyze the space of possible attacks, for example, by altering goals or initial state configuration.

Another class of domain models, which uses planning-based diagnosis techniques [Sohrabi *et al.*, 2010], allows augmenting the planning problem with a sequence of ambiguous and unreliable observations from network monitoring systems to generate plans of attacks that may be in progress [Sohrabi *et al.*, 2013].

Planners can indeed be very effective in network security applications, even compared to more traditional penetration testing tools and analysis techniques based on attack graphs, because planning provides efficient search in huge attack graphs induced by domain models that are exceedingly small by comparison, and therefore are significantly easier to design and maintain [Lucngeli *et al.*, 2010].

Nevertheless, this still leaves open the question of selecting the relevant portions of attack graphs to bring to the attention of the network administrators (or automated penetration testing tools). One of the planning systems discussed above proposed generating top high quality plans to address this issue [Sohrabi *et al.*, 2013]. While finding multiple high quality plans is beneficial and necessary in order to focus the investigation on the efficient and hence relevant attacks, by itself it is not sufficient. If generated attack plans are too similar to each other in actions and states, administrators can be easily overwhelmed with small variations of the same pattern. In addition, planner performance is important for timely response to ongoing attacks.

We believe this combination of requirements has not been addressed in prior work. The problem we are considering is closely related to diverse planning, but plans found by the diverse planners are not necessarily all of high quality [Nguyen *et al.*, 2012; Sroka and Long, 2012]. On the other hand, top-k planning [Riabov *et al.*, 2014] provides an efficient technique for finding a set of top quality plans, by applying K^* algorithm [Aljazzar and Leue, 2011] to augment A^* search, but it does not guarantee plan diversity.

We propose to address these challenges by using a top-k planner to create a large set of high-quality plans, followed by clustering based on a similarity metric, and finally outputting a set of cluster representatives that are both high quality and

diverse. In the rest of the paper, we briefly describe the top-k planning and clustering techniques we use, and evaluate the end-to-end performance of our approach, comparing to existing diverse planners and a top-k planner.

2 Network Security: Domain Models

Rather than develop a new domain model for the network security application, our goal is to show via experiments how our approach can benefit previously proposed models. We generated planning task instances for two classes of domain models: attack graphs, as discussed in [Boddy *et al.*, 2005; Roberts *et al.*, 2011; Lucngeli *et al.*, 2010] and hypothesis exploration based on unreliable observations and a model of a dynamical system, as described in [Sohrabi *et al.*, 2013].

For the attack graph model, we followed an approach similar to [Lucngeli *et al.*, 2010]. We obtained a list of products and known vulnerabilities from The U.S. National Vulnerability Database, allowing us to define up to 3554 network exploit actions, each annotated with low, medium or high complexity, and configure the hosts with a selection from 21107 products. We also generated a random hierarchical network topology of up to 2000 nodes on up to 150 networks separated by firewalls, with several VPN connections cutting across the hierarchy. We defined the following actions: *connect-tcp*, which checked network connectivity between hosts, *exploit-net-V*, which represented remote exploitation of a known vulnerability *V* for a compatible product, with higher costs for more complex attacks, and *exploit-local-V*, with similarly assigned costs, which was invoked after the remote exploitation succeeded, to achieve escalation of privilege and fully compromise the host and use it for subsequent attacks. All network hosts, including gateway nodes between networks, were configured to have at least one locally exploitable product and one remotely exploitable product, which ensured a large number of possible attacks, and therefore a large space of plans for exploration.

Our hypothesis exploration domain model, following [Sohrabi *et al.*, 2013], uses two manually defined state transition systems: a malware detection system with 25 states described in that work, and an extended version of it representing two hosts and interactions between them, with 221 states total. To generate problem instances, we generated random observation sequences of varying length, and combined them with the transition systems. In this case, the planning task models the states of the host, transitions between states, and the corresponding observations from network traffic monitoring sensors, with the goal of generating hypotheses explaining the observations. For example, a malware infection state of a host can have a corresponding observation of downloading an executable file, and a possible transition to a command-and-control rendezvous state which has a new Internet Relay Chat (IRC) session observation. The executable download may also be observable in other states, such as remote software installation by administrator, which makes this observation ambiguous, also making multiple hypotheses necessary. Note that due to this structure of the model, and since the same transition system state can be reached through different explanation paths, one state of the modeled system corresponds to many possible planning states. Consequently, this model generates a very large space of plans, as in the attack graph domain.

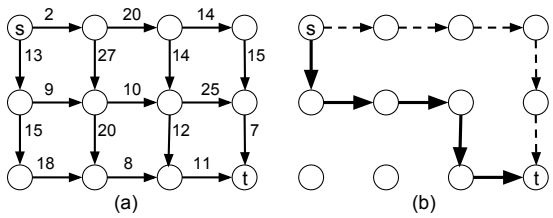

Figure 1: (a) shows a graph with source node s and terminal node t with edge lengths specified on the edges; (b) shows the shortest path in bold arrows and the second shortest path in dashed arrows.

3 Finding Diverse High Quality Plans

To measure plan quality, we assume that action costs can be assigned such that cost of a plan is the sum of costs of actions included in the plan, and the lower the cost of the plan, the higher its quality. The first step of our approach is to generate a fixed number k of lowest-cost plans, and then cluster similar plans based on a plan similarity metric. We require k to be sufficiently large, so that when similar plans are clustered, we obtain the necessary number of clusters. Finally, one lowest-cost representative plan from each cluster is selected to form the resulting set of diverse high quality plans.

In this section, we first introduce what we call a top-k planning problem. Then, we describe how to compute top-k plans using a k shortest paths algorithm. Finally we describe the clustering algorithm and the plan similarity metric used for clustering.

3.1 Tok-k Planning Problem

Definition 1 *We define the top-k planning problem as* $R = (F, A, I, \mathcal{G}, k)$*, where* F *is a finite set of fluent symbols,* A *is a set of actions with non-negative costs,* $I \subseteq F$ *defines the initial state,* $\mathcal{G} \subseteq F$ *defines the goal state, and* k *is the number of plans to find. Let* $R' = (F, A, I, \mathcal{G})$ *be the planning problem with action costs that has* n *plans (*n *can be infinite). The set of plans* $\Pi = \{\pi_1, ..., \pi_m\}$*, where* $m = k$ *if* $k \leq n$*,* $m = n$ *otherwise, is a solution to* R *if an only if each* $\pi_i \in \Pi$ *is a plan for* R' *and there does not exists a plan* π' *for* R'*,* $\pi' \notin \Pi$*, and a plan* $\pi_i \in \Pi$ *such that* $cost(\pi') < cost(\pi_i)$ *.*

The solution to the top-k planning problem is a set of low-cost plans, and are not necessary all optimal. If k is less than the number of optimal plans for R', then Π will contain all of the optimal plans. However, if k is larger than the number of optimal plans for R' then Π will contain some suboptimal plans in addition to all optimal plans. Also note that if $k > n$, Π contains all n valid plans, otherwise it contains k plans

3.2 Background: K Shortest Paths Problem

K shortest paths problem is an extension of the shortest path problem where in addition of finding one shortest path, we need to find a set of paths that represent the k shortest paths [Hoffman and Pavley, 1959].

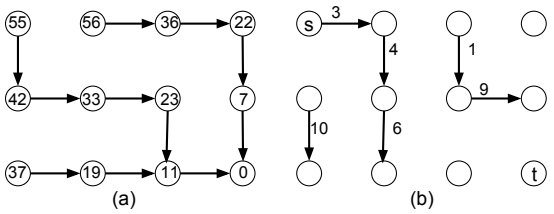

Figure 2: (a) shows the shortest path tree T and distance to destination t; (b) shows the side edges with their associated detour cost.

The following is a formal definition taken from Eppstein [Eppstein, 1998].

Definition 2 (K Shortest Path Problem) *k shortest path problem is defined as 4-tuple $Q = (G, s, t, k)$, where $G = (V, E)$ is a graph with a finite set of n nodes (or vertices) V and a finite set of m edges E, s is the source node, t is the destination node, and k is the number of shortest paths to find. Each edge $e \in E$ has a* length *(or* weight *or* cost*), which is denoted by $l(e)$. The length of a path p, $l(p)$, is consequently defined by the sum of its edge lengths. The* distance *$d(u, v)$ for any pair of nodes u and $v \in V$ is the length of the shortest path between the two nodes. Hence, $d(s, t)$ is the length of the shortest path for the problem Q. Let n = size of the set of all s-t paths in graph G. Then, the set of paths $P = \{p_1, p_2, ..., p_m\}$, $m = k$ if $k \leq n$, $m = n$ otherwise, is the solution to the k shortest paths problem Q if and only if each $p_i \in P$, is a s-t path in graph G and there does not exist a s-t path p' in graph G, $p' \notin P$ and a path $p_i \in P$ such that $l(p') < l(p_i)$.*

Note that if $k > n$, then P contains all s-t paths, otherwise P contains k shortest paths from node s to node t. It follows from the definition that at least one shortest path with length $d(s, t)$ is in the set P if $m > 0$. Figure 1 shows an example from [Eppstein, 1998] to illustrate the terminology. The distance $d(s, t) = 55$, is the length of the shortest path shown in bold; the length of the second shortest path is 58.

The K^* algorithm [Aljazzar and Leue, 2011] is an improved variant of the Eppstein's k shortest paths algorithm [Eppstein, 1998] and hence uses many of the same concepts as in the Eppstein's algorithm (which we refer to as EA). Here, we first outline the EA algorithm, and then discuss K^*.

Given a k shortest paths problem Q, the EA algorithm first computes a single-destination shortest path tree with t as the destination (or the reversed single-source shortest path tree) by applying Dijkstra's algorithm on G. The edges in the resulting *shortest path* tree, T are called the *tree* edges while all the missing edges (i.e., the edges in $G - T$) are called the *sidetrack* edges. Each edge in G is assigned a number that measure the detour cost of taking that edge. Consequently, the detour cost of the tree edges is 0, while the detour cost of the sidetrack edges is greater than 0. Figure 2 shows the shortest path tree T and the sidetrack edges along with their detour cost of our earlier example.

The EA algorithm then constructs a complex data structure called *path graph* $P(G)$ that stores the all paths in G, where each node in represents a sidetrack edge. This is followed by the use of Dijkstra search on $P(G)$ to extract the k shortest paths. An important property is that given a sequence of sidetrack edges representing a path in $P(G)$ and the shortest path tree T, it is possible to uniquely construct a s-t path in G. This can be done by using sub-paths from T to connect the endpoints of sidetrack edges. Given this property and the special structure of $P(G)$, it is ensured that the i-th shortest path in $P(G)$ results in a sidetrack sequence which can be mapped to the i-th shortest path in G. By construction, $P(G)$ provides a heap-ordered enumeration of all paths in G, and since every node of $P(G)$ has limited out-degree (at most 4), the complexity of enumerating paths in increasing cost order is bounded. The worst-case runtime complexity of the EA algorithm is $O(m+n \log n+kn)$. This complexity bound depends on a compact representation of the resulting k paths, and can be exceeded if the paths are written by enumerating edges.

Although EA could be used for top-k planning, the K^* algorithm is preferable, because it does not require the complete state transition graph G. Instead, K^* can create G dynamically using $A*$ search, driven by a heuristic toward the goal, an approach commonly used in planners (e.g., Fast-Downward [Helmert, 2006]). In short, the K^* algorithm works as follows. The first step is to apply a forward A* search to construct a portion of graph G. The second step is suspending A* search, updating $P(G)$ similarly to EA, to include nodes and sidetracks discovered by A*, applying Dijkstra to $P(G)$ to extract solution paths, and resuming the A* search. The use of A* search to dynamically expand G enables the use of heuristic search and also allows extraction of the solution paths before G is fully explored. While K^* algorithm has the same worst-case complexity as the EA algorithm, it has better performance in practice because unlike the EA algorithm, K^* does not require the graph G to be completely defined when the search starts.

3.3 Top-k Planning Using K^*

In the implementation of the planning algorithm we follow the algorithm structure imposed by K^*, as follows.

```
 0. Read planning problem R = (F, A, I, G, k).
 1. Expand the state graph G by using A*
    and applying actions to compatible  states
    starting from I, and until  G is reached.
 2. Continue applying A* to expand G
    until 20% increase in links or nodes.
 3. Update P(G) based on new links in G.
 4. Apply Dijkstra step
    to extract the next path from P(G).
 5. If k paths are found
 6.      Goto step 10.
 7. If K* scheduling condition is reached
 8.      Goto step 2.
 9. Goto step 4.
10. Return at most k plans (one plan per path).
```

We expect that with some work this approach can be integrated into planners that use A* search, enabling those planners to solve top-k problems. Also note that the soundness and completeness of planning follows directly from the soundness and completeness of the K^* algorithm.

3.4 Clustering Algorithm

Given the set of top-k plans, in this section, we will discuss how to group the similar plans using clustering techniques. In practice, many of the generated top-k plans are only slightly different from each other. That is, they do seem to be duplicates of each other, except for one or more states or actions that are different. This may be the result of the underlining AI planner which tries to generate all alternative low-cost plans, and while this generates distinct low-cost plans, it does not always mean that these plans are significantly different from each other. Hence, instead of presenting large number of plans, some of which could be very similar to each other, with the help of clustering, we can present clusters of plans, where each cluster can be replaced by its representative plan.

Clustering has been a topic of interest in several areas of research within several communities such as Information Retrieval (e.g, [Aslam *et al.*, 2004]), machine learning, and Data Management as part of the data cleaning process (e.g., [Hassanzadeh and Miller, 2009]). Many survey papers exist on clustering algorithms (e.g, [Xu and Wunsch, 2005; Filippone *et al.*, 2008]). While most, if not all, clustering algorithms share a common goal of creating clusters that minimize the intra-cluster distance (distance between members of the same clusters) and maximize the inter-cluster distance (distance between members of different clusters), the assumptions and inputs for these clustering algorithm are often different. For example, several of these approaches assume some given input parameters such as the number of clusters or a cluster diameter. To consolidate similar plans produced by the top-k planner, we apply a clustering algorithm that must satisfy the requirements stated below. One representative plan from each cluster is selected to be included in the final set of diverse plans.

Definition 3 (Clustering Requirements) *Given a set of k sorted plans, Π, create clusters of plans $\mathcal{C} = \{c_1, ..., c_o\}$ where the value of o is unknown ahead of time. Further, for each two clusters c, $c' \in \mathcal{C}$, $c \cap c' = \emptyset$ and $\forall \pi \in \Pi$, $\exists c \in \mathcal{C}$ such that $\pi \in c$. Hence, the clusters are disjoint and each plan belongs to one cluster.*

We propose the use of the following three non-hierarchical clustering algorithms. Each of these algorithms require visiting each plan only once in order to decide to which cluster they belong to; hence, are called single-pass algorithms.

Center-Link

Center-Link clustering algorithm iterates over the top-k plans starting with the lowest-cost plan. For each plan, it computes the similarity to a representative of each cluster created in previous iterations. If there are no clusters that have a representative similar to the plan (i.e., their similarity score is above the threshold θ), a new cluster is created and the plan becomes the representative of that cluster. Otherwise the plan is added to the first cluster whose cluster representative is similar to this plan. Cluster representatives are chosen to be the lowest-cost plans in each cluster. Due to the order of iteration, stating from the lowest-cost plans, the cluster representative is always the first added plan to the cluster. This algorithm is similar to the CENTER algorithm in [Hassanzadeh and Miller, 2009], however, the sorted input is different (i.e., plans, as opposed to records in a database). The Center-Link algorithm could result in small number of similarity comparisons because each plan is only compared to the representative plan of each cluster.

Single-Link

Single-Link clustering algorithm is an extension of the Center-Link algorithm, where instead of comparing only with the representative of a cluster, each plan is compared with all members of a cluster, and if the plan is found to be similar to any of the members of that cluster, then it is assigned to that cluster. Single-Link algorithm is a non-hierarchical variation of single-linkage algorithm [Xu and Wunsch, 2005]; the node joins a cluster as long as there is a single link with one of the members of the clusters. This algorithm could result in the smallest number of clusters.

Average-Link

Average-Link algorithm is a simple extension of the Single-Link algorithm, where each plan is compared with all the members of a cluster and the average similarity score is used to determine if the plan belong to that cluster or not. This algorithm results in many similarity comparisons, and could result in large number of clusters. Note, Average-Link clustering is a non-hierarchical variant of hierarchical average-linkage clustering [Xu and Wunsch, 2005]. In the experiments we show we have used this algorithm because it produced more clusters and more diverse plans.

3.5 Plan Similarity Metric

Finding if two plans are similar has been studied mainly under two categories: plan stability for replanning (e.g., [Fox *et al.*, 2006]) and finding diverse plans (e.g., [Nguyen *et al.*, 2012]). We follow prior work on diverse planning and compute plan similarity during clustering as Jaccard similarity between actions of the plan, thereby grouping similar plans together and increasing the diversity between the clusters.

Jaccard similarity is a score between 0 and 1, which measures the ratio of the number of actions that appear in both plans to the total number of actions appearing in at least one plan. Let A(π) be the set of actions in π, then:

$$\text{sim}_{\text{Jaccard}}(\pi, \pi') = \frac{|A(\pi) \cap A(\pi')|}{|A(\pi) \cup A(\pi')|} \quad (1)$$

We note that Jaccard similarity is the inverse of the plan distance defined in [Nguyen *et al.*, 2012].

4 Experimental Evaluation

To compare planner performance, we configure the planners so that approximately 50 diverse plans are generated. We measure plan diversity using stability and uniqueness metrics. We also compare plan cost and planning time.

We measure stability and uniqueness using the following formula from [Roberts *et al.*, 2014]. Note, we modified these formula to make it a number between 0 and 1. Let $\Pi = \{\pi_1, ..., \pi_m\}$ be the set of plans. If $|\Pi| = 1$, $\text{Diversity}_{\text{stability}}(\Pi) = 1$, and $\text{Diversity}_{\text{uniqueness}}(\Pi) = 1$, otherwise for $|\Pi| \geq 1$:

Domain	Top-k + clustering					LPG-d					Div				
	Time	**#Plans**	**Cost**	**D**	**U**	**Time**	**#Plans**	**Cost**	**D**	**U**	**Time**	**#Plans**	**Cost**	**D**	**U**
Malware-5	1	50	1502	0.51	1	1	10	3513	0.80	1	1	9	1789	0.36	0.37
Malware-10	1	50	1586	0.41	0.99	59	10	8426	0.84	1	1	5	3861	0.44	0.54
Malware-20	3	50	1492	0.20	0.99	384	10	16520	0.87	1	1	6	7262	0.46	0.53
Malware2-5	1	50	2005	0.66	0.96	-	-	-	-	-	1	3.2	1454	0.40	0.98
Malware2-10	2	50	2441	0.47	1	-	-	-	-	-	5	6	6092	0.68	0.97
Malware2-20	5	50	2105	0.28	1	-	-	-	-	-	3	6.8	4910	0.35	0.95
AttackGraph-1	3	50	54	0.59	1	1.89	2	59	0.95	0.43	-	-	-	-	-
AttackGraph-2	194	50	65	0.30	1	-	-	-	-	-	-	-	-	-	-

Table 1: Comparisons of planning time, plan diversity and average plan cost.

$$\text{Diversity}_{\text{stability}}(\Pi) = \frac{\sum_{\pi_i,\pi_j \in \Pi, i \neq j} [1 - \text{sim}_{\text{Jaccard}}(\pi_i, \pi_j)]}{|\Pi| \times (|\Pi| - 1)} \quad (2)$$

$$\text{Diversity}_{\text{uniqueness}}(\Pi) = \frac{\sum_{\pi_i,\pi_j \in \Pi, i \neq j} \begin{cases} 0 & \text{if } \pi_i \setminus \pi_j = \emptyset \\ 1 & \text{otherwise} \end{cases}}{|\Pi| \times (|\Pi| - 1)} \quad (3)$$

4.1 Experiment Results

We use a family of malware detection planning problems described in [Sohrabi *et al.*, 2013]. We varied the size of the problem by changing the number of observations, with Malware2 problem also supporting more system states. All planning problems share a planning domain description containing 6 actions and 8 predicates. In this domain, low costs were assigned to actions used in perfect explanations of observations, and high costs to actions representing exceptions, such as unexplained observations or state transitions without observations. For the attack graph model we are using the approach discussed in domain model section.

Table 1 summarizes the experiment results with two domain models. In all experiments we used a dual 16-core 2.70 GHz Intel(R) Xeon(R) E5-2680 processor with 256 GB RAM.The results presented in each row, corresponding to a planning domain. For Malware domains, the results are averages over 5 instances of each size. We have terminated planners after the time limit of 30 minutes was reached.

We compare the performance to two planning systems, LPG-d [Nguyen *et al.*, 2012] and Div [Roberts *et al.*, 2014], with our system that combines top-k and clustering.For both planners we have set the number of plans to 10, with the intent to produce a small number of attack scenarios to review.

The column **Time** in Table 1 contains planning time in seconds. The **#Plans** column contains the number of plans produced. The **Cost** column is the average cost of those plans. The **D** column is the average $\text{Diversity}_{\text{stability}}(\Pi)$, and the **U** column is the average $\text{Diversity}_{\text{uniqueness}}(\Pi)$. The closer these two diversity metrics are to 1, the more different are the plans.

We have selected LPG-d because it creates plans that maximize diversity, and that was confirmed by our experiments. Even though we have set parameter d to 0.1, to be equal to θ for our clustering algorithm, $\text{Diversity}_{\text{stability}}(\Pi)$ is above 0.8 in all experiments for this planner.

Div places greater emphasis on plan cost, and indeed average plan cost is lower than for LPG-d. However it sometimes produces multiple copies of the same plan, resulting in very poor diversity metrics. There are no AttackGraph results for Div, due to a crash without an error message.

The Top-k planner produced 1000 plans, which were later clustered. The number of clusters is determined by the similarity threshold θ, which was set to a low value 0.1 to ensure sufficient number of clusters, further bounded at maximum 50. It was the only planner that could solve the largest AttackGraph-2 instance with 4000 vulnerabilities and 1950 hosts. Overall this simple approach performs well in these domains in terms of planning time, and plan quality, but generally has lower plan stability metric that measures plan diversity. This is expected since we are trading off plan diversity for plan quality. In the network application, prioritizing cost is a desirable property because it eliminates from consideration attacks that are different from previously considered, but too inefficient to be carried out in practice. Of course, the quality and the applicability of the obtained solutions ultimately depends on knowledge engineering, and specifically on how action costs map to relevant attacks.

5 Related Work

Generating a plan set rather than just one plan has been a subject of interest in several recent papers in the context of generating diverse plans (e.g., [Roberts *et al.*, 2014; Coman and Muñoz-Avila, 2011]). Several plan distance measures most of which are domain-independent have been proposed to both guide the search and evaluate the set of diverse of plans (e.g., [Srivastava *et al.*, 2007; Bryce, 2014]). Given some partial preferences or multiple dimensions of quality, such as cost or time, the problem becomes a multi-objective optimization problem where diverse plans should form a Pareto-optimal set [Nguyen *et al.*, 2012]. Sroka and Long 2012 argue that the previous work will not find good-quality plans as they are more focused on finding diverse plans since it is "easier to find diverse sets father away from optimal". The work we presented in this paper falls in between. While we are given some notion of quality as measured by cost, the cost function itself is imperfect, and we are not given other objective functions besides costs. So finding one min-cost plan is not enough, nor is finding a diverse set of plans without taking into consideration the cost function. Hence, finding a set of diverse low-cost plans is required.

6 Conclusions and Future Work

In this paper we propose to address the plan space exploration problem arising in network security applications by generating high-quality diverse plans. We find that the existing work on diverse planning does not address this problem directly, and we propose a new approach specifically for this task, by combining top-k planning and plan clustering. Experimental evaluation shows that our new technique provides significant improvements in both plan quality and planning time. Although the primary focus of this work is to facilitate planning-assisted attack graph exploration carried out by network administrators, the techniques we are using are domain-independent, and future work may involve studying the applicability and the potential benefits of this approach in other applications, as well as integration with network security systems and evaluation via user studies.

References

[Aljazzar and Leue, 2011] Husain Aljazzar and Stefan Leue. K*: A heuristic search algorithm for finding the k shortest paths. *Artificial Intelligence*, 175(18):2129–2154, December 2011.

[Aslam *et al.*, 2004] J. A. Aslam, E. Pelekhov, and D. Rus. The Star Clustering Algorithm For Static And Dynamic Information Organization. *Journal of Graph Algorithms and Applications*, 8(1):95–129, 2004.

[Boddy *et al.*, 2005] Mark S. Boddy, Johnathan Gohde, Thomas Haigh, and Steven A. Harp. Course of action generation for cyber security using classical planning. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 12–21, 2005.

[Bryce, 2014] Daniel Bryce. Landmark-based plan distance measures for diverse planning. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 56–64, 2014.

[Coman and Muñoz-Avila, 2011] Alexandra Coman and Hector Muñoz-Avila. Generating diverse plans using quantitative and qualitative plan distance metrics. In *Proceedings of the 25th National Conference on Artificial Intelligence (AAAI)*, pages 946–951, 2011.

[Eppstein, 1998] David Eppstein. Finding the k shortest paths. *SIAM Journal on Computing*, 28(2):652–673, 1998.

[Filippone *et al.*, 2008] M. Filippone, F. Camastra, F. Masulli, and S. Rovetta. A Survey of Kernel and Spectral Methods for Clustering. *Pattern Recognition*, 41(1):176–190, 2008.

[Fox *et al.*, 2006] Maria Fox, Alfonso Gerevini, Derek Long, and Ivan Serina. Plan stability: Replanning versus plan repair. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 212–221, 2006.

[Hassanzadeh and Miller, 2009] Oktie Hassanzadeh and Renée J. Miller. Creating Probabilistic Databases from Duplicated Data. *VLDB Journal*, 18(5):1141–1166, 2009.

[Helmert, 2006] Malte Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[Hoffman and Pavley, 1959] Walter Hoffman and Richard Pavley. A method for the solution of the *n*th best path problem. *Journal of the ACM*, 6(4):506–514, 1959.

[Lucngeli *et al.*, 2010] Jorge Lucngeli, Carlos Sarraute, and Gerardo Richarte. Attack planning in the real world. In *Workshop on Intelligent Security (SecArt 2010)*, 2010.

[Nguyen *et al.*, 2012] Tuan Anh Nguyen, Minh Binh Do, Alfonso Gerevini, Ivan Serina, Biplav Srivastava, and Subbarao Kambhampati. Generating diverse plans to handle unknown and partially known user preferences. *Artificial Intelligence*, 190:1–31, 2012.

[Riabov *et al.*, 2014] Anton Riabov, Shirin Sohrabi, and Octavian Udrea. New algorithms for the top-k planning problem. In *Proceedings of the Scheduling and Planning Applications woRKshop (SPARK) at the 24th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 10–16, 2014.

[Roberts *et al.*, 2011] M. Roberts, A. Howe, I. Ray, M. Urbanska, Z. S. Byrne, and J. M. Weidert. Personalized vulnerability analysis through automated planning. In *Working Notes of IJCAI 2011, Workshop Security and Artificial Intelligence (SecArt-11)*, 2011.

[Roberts *et al.*, 2014] Mark Roberts, Adele E. Howe, and Indrajit Ray. Evaluating diversity in classical planning. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 253–261, 2014.

[Sohrabi *et al.*, 2010] Shirin Sohrabi, Jorge Baier, and Sheila McIlraith. Diagnosis as planning revisited. In *Proceedings of the 12th International Conference on the Principles of Knowledge Representation and Reasoning (KR)*, pages 26–36, 2010.

[Sohrabi *et al.*, 2013] Shirin Sohrabi, Octavian Udrea, and Anton Riabov. Hypothesis exploration for malware detection using planning. In *Proceedings of the 27th National Conference on Artificial Intelligence (AAAI)*, pages 883–889, 2013.

[Srivastava *et al.*, 2007] Biplav Srivastava, Tuan Anh Nguyen, Alfonso Gerevini, Subbarao Kambhampati, Minh Binh Do, and Ivan Serina. Domain independent approaches for finding diverse plans. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2016–2022, 2007.

[Sroka and Long, 2012] Michal Sroka and Derek Long. Exploring metric sensitivity of planners for generation of pareto frontiers. In *Proceedings of the 6th Starting AI Researchers' Symposium (STAIRS)*, pages 306–317, 2012.

[Xu and Wunsch, 2005] R. Xu and I. Wunsch. Survey of Clustering Algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, 2005.