Finding Diverse High-Quality Plans for Hypothesis Generation

Shirin Sohrabi and **Anton V. Riabov** and **Octavian Udrea** and **Oktie Hassanzadeh** [1]

Abstract.
In this paper, we address the problem of finding diverse high-quality plans motivated by the hypothesis generation problem. To this end, we present a planner called TK* that first efficiently solves the "top-k" cost-optimal planning problem to find k best plans, followed by clustering to produce diverse plans as cluster representatives.

1 Introduction

New applications that use AI planning to generate explanations and hypotheses have given rise to a new class of planning problems, requiring finding multiple alternative plans while minimizing the cost of those plans [11, 7, 9]. Hypotheses or explanations about a system, such as a monitored network host that could be infected by malware, are generated as candidate plans given a planning problem definition describing the sequence of observations that can be noisy, incomplete, or missing, and a domain model capturing the possible state transitions for the modeled system, as well as the many-to-many correspondence between the states and the observations. The plans must minimize both the penalties for unexplained observations and the cost of state transitions. Additionally, among those candidate plans, a small number of the most diverse plans must be selected as representatives for further analysis.

The malware detection problem or more generally the hypothesis generation problem is encoded as an AI planning problem, where the generated plans correspond to the hypotheses, and furthermore, the min-cost (or top-quality) plans correspond to the plausible hypotheses. Plausible hypotheses are those that the domain expert believes to be more plausible (more likely) compared to the other hypotheses. Plausibility can be encoded as an action cost, where higher costs indicate lower plausibility. Hence, the notion of the top-k plans maps to finding k plans with the lowest cost.

In this paper, we propose an approach for finding a set of low-cost diverse plans for hypothesis generation. To this end, we have developed a planner that first efficiently solves the "top-k" cost-optimal planning problem to find k best plans, followed by clustering to produce diverse plans as cluster representatives. Our framework is modular allowing different planning algorithms, similarity measures, and clustering algorithms in different combinations. Experiments set in hypothesis generation domains show that the top-k planning problem can be solved in time comparable to cost-optimal planning using Fast-Downward. We further empirically evaluate multiple clustering algorithms and similarity measures, and characterize the tradeoffs in choosing parameters and similarity measures.

[1] IBM T.J. Watson Research Center, Yorktown Heights, NY, USA, email: {ssohrab, riabov, udrea, hassanzadeh}@us.ibm.com

2 Top-k Planning Using K*

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 is a clause over F defining the initial state, $\mathcal{G}$ is a clause over F defining the goal state, and k is the number of plans to find. Let $R' = (F, A, I, \mathcal{G})$ be the cost optimal planning problem with n valid plans. The set of plans $\Pi = \{\alpha_1, ..., \alpha_m\}$, where $m = k$ if $k \leq n$, $m = n$ otherwise, is the solution to R if and only if each $\alpha_i \in \Pi$ is a plan for the cost-optimal planning problem R' and there does not exist a plan α' for R', $\alpha' \notin \Pi$, and a plan $\alpha_i \in \Pi$ such that $cost(\alpha') < cost(\alpha_i)$. When $k > n$, Π contains all n valid plans, otherwise it contains k plans. Π can contain both optimal plans and sub-optimal plans, and for each plan in Π all valid plans of lower cost are in Π. If $\Pi \neq \emptyset$, it contains at least one optimal plan.

To solve the top-k planning problem, R, we will apply the k shortest path algorithm, K*, to find the top-k plans. K shortest paths problem is an extension of the shortest path problem where in addition to finding one shortest path, we need to find a set of paths that represent the k shortest paths. K shortest path problem is defined as $Q = (G, s, t, k)$, where $G = (V, E)$ is a graph with a finite set of nodes V and a finite set of edges E, s is the source node, t is the destination node, and k is the number of shortest paths to find. The K^* algorithm [1] is an improved variant of the Eppstein's k shortest paths algorithm [2] (we refer to as EA). The EA algorithm constructs a complex data structure called *path graph* $P(G)$ that stores the all paths in G, where each node in $P(G)$ represents a sidetrack edge. This is followed by the use of Dijkstra search on $P(G)$ to extract the k shortest paths. The major bottleneck of the EA algorithm is the construction of the complete state transition graph, which may include a huge number of states that are very far away from the goal.

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.

Our planner, *TK**, applies K^* to search in state space, with dynamic grounding of actions, similarly to how Fast-Downward and other planners apply A^*. The K^* scheduling condition is evaluated by comparing the state of A* and Dijkstra searches, as defined in K^* algorithm. It determines whether new links must be added to G before resuming Dijkstra search on updated $P(G)$. There is no separate grounding stage, since actions are ground at the same time when they are applied during A* search. The amount of A* expansion required before resuming Dijkstra (in our implementation, 20%) controls the efficiency tradeoff, and 20% is the same value that was used in experiments in the original K^* paper [1]. Soundness and completeness of *TK** follows directly from the soundness and completeness of the K^* algorithm. For further details of the *TK** planner see [10].

3 Finding Diverse Plans via Clustering

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. To consolidate similar plans produced by the top-k planner, we compute a similarity score between plans and apply three clustering algorithms that create clusters of plans where each cluster is disjoint and each plan belongs to only one cluster. We then may choose to present only the cluster representatives from a subset of these clusters to the user or to the automated system for further investigation.

Finding if two plans are similar has been studied mainly under plan stability for replanning and finding diverse plans (e.g., [3, 6]). Two plans can be compared based on their actions, states, or causal links. We also consider comparing plans based on their costs or their final states. Each similarity measure assigns a number between 0 (unrelated) or 1 (same). Two plans are said to be similar if their similarity score is above a predefined threshold θ. The similarity measures can be used individually or be combined using a weighted average.

We have implemented several similarity measures including Generalized Edit Similarity (GES) and Jaccard Similarity, an inverse of the plan distance from [6]. An important desired property of GES is that it not only considers the similarity between sequences, but also considers the similarity between tokens. Therefore, we are able to use any extra domain-dependent knowledge at hand about the relationship between, for example, actions to determine if two plans belong to the same cluster. This allows further semantic information to be included in similarity calculations.

We have also implemented three clustering algorithms: Center-Link, Single-Link, and Average-Link [10]. Each of these algorithms require visiting each plan only once in order to decide the cluster they belong to; however, depending on which algorithm is used, the plans are compared to representative element of the cluster or all plans.

4 Experimental Evaluation

We used both manually crafted and random problems to create our evaluation benchmark. Our problems are based on the hypothesis generation application described by Sohrabi et al. [11]. This application is a good example of a challenging top-k planning problem, and generated problems typically have a very large number of possible plans with different costs. We report a summary of our evaluations.

Top-k Planning Performance We compare the performance of our top-k planner, *TK**, with k=50 and k=1000 to Gamer [5] (Gamer 2014 version) and Fast-Downward [4] (2015 version, with A^*). Both find a single cost-optimal plan, which is equivalent to k=1. Overall, our results showed that *TK** is very efficient at finding top-k plans, and in our implementation and our set of problems performs at least as fast or faster than Fast-Downward and Gamer, which is essential for use in applications. Due to soundness and completeness of K^*, *TK** is guaranteed to produce top-k plans and that was confirmed in our experiments. Overall, these experiment results support our claim that top-k problems can be solved just efficiently as cost-optimal ones, at least within a certain class of planning domains.

Evaluation of Clusters We evaluate the different clustering algorithms and similarity measures we used. Our results show that Center-Link algorithm is the best algorithm with respect to time as fewer number of similarity comparisons is performed since each plan is only compared to the representatives. Average-Link produces more clusters compared to the other two. As the threshold increases, the number of clusters also increases for all algorithms. With respect to stability and uniqueness measures [8], the results does not show a superior clustering algorithm. Furthermore, the results show that grouping based on cost or last state may be fastest but these similarity measures give the worst results with respect to stability and uniqueness. On the other hand, using Jaccard produces most diverse plans with respect to uniqueness and GES also produces most diverse plans with respect to stability.

Comparison With Diverse Planners We selected two representative diverse planners, LPG-d [6] (with d=0.1) and Div (Multi-queue A^* MQA_{TD}) [8], and compared to our implementation that included top-k and Average-link clustering, using Jaccard similarity. We averaged over 5 instances of each size and had a 30 minutes time limit. We measure time in seconds, shown under T, plan diversity by stability, S, and by uniqueness, U, using formula from [8]. Subset of the results are shown in the following table. The results show that Div places greater emphasis on plan cost but sometimes produces multiple copies of the same plan, resulting in poor diversity. LPG-d produced diverse plans but with higher average cost. Our approach, the top-k plus clustering, produces the lowest average cost with somewhat lower diversity compared to LPG-d.

	Top-k + Average Link				**LPG-d**				**Div**			
Prob	**T**	**Cost**	**S**	**U**	**T**	**Cost**	**S**	**U**	**T**	**Cost**	**S**	**U**
1	1	1502	0.51	1	1	3513	0.80	1	1	1789	0.36	0.37
2	1	1586	0.41	0.99	59	8426	0.84	1	1	3861	0.44	0.54
3	3	1492	0.20	0.99	384	16520	0.87	1	1	7262	0.46	0.53

REFERENCES

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

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

[3] Maria Fox, Alfonso Gerevini, Derek Long, and Ivan Serina, 'Plan stability: Replanning versus plan repair', in *ICAPS*, pp. 212–221, (2006).

[4] Malte Helmert, 'The Fast Downward planning system', *JAIR*, **26**, 191–246, (2006).

[5] Peter Kissmann, Stefan Edelkamp, and Jörg Hoffmann, 'Gamer and Dynamic-Gamer symbolic search at IPC 2014', in *IPC-2014*, (2014).

[6] Tuan Nguyen, Minh Do, Alfonso Gerevini, Ivan Serina, Biplav Srivastava, and Subbarao Kambhampati, 'Generating diverse plans to handle unknown and partially known user preferences', *AIJ*, **190**, 1–31, (2012).

[7] Anton V. Riabov, Shirin Sohrabi, Daby M. Sow, Deepak S. Turaga, Octavian Udrea, and Long H. Vu, 'Planning-based reasoning for automated large-scale data analysis', in *ICAPS*, pp. 282–290, (2015).

[8] Mark Roberts, Adele E. Howe, and Indrajit Ray, 'Evaluating diversity in classical planning', in *ICAPS*, pp. 253–261, (2014).

[9] Shirin Sohrabi, Anton Riabov, and Octavian Udrea, 'Plan recognition as planning revisited', in *IJCAI*, (2016).

[10] Shirin Sohrabi, Anton Riabov, Octavian Udrea, and Oktie Hassanzadeh, 'Finding diverse high-quality plans for hypothesis generation', in *MRC workshop at ECAI*, (2016).

[11] Shirin Sohrabi, Octavian Udrea, and Anton Riabov, 'Hypothesis exploration for malware detection using planning', in *AAAI*, (2013).