Finding Diverse High-Quality Plans for Hypothesis Generation

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

Abstract.
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. 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 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. 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. 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 Introduction

In recent work a new class of AI planning formulations has been developed for solving practical problems in plan recognition, diagnosis of discrete event systems, and explanation generation (e.g., [17, 21, 22]). In these problems, each valid plan can be interpreted as a hypothesis meeting the constraints of the planing task, and providing a possible diagnosis or an explanation.

In prior work, these problems have been studied in satisficing or optimal planning settings. More recently, however, Sohrabi et al. [25] have shown that in malware detection applications, where observations can be noisy or the domain model can be imperfect, finding multiple near-optimal plans makes a significant difference in discovering ground truth scenarios, and therefore improves the overall utility of generated explanations.

Consider the following example of an application where finding multiple low-cost plans is desirable:

Example *In automated malware detection in computer networks, the goal is to provide assistance to network administrators in detecting and predicting behaviors of malware or computer viruses. Observations come from network traffic, but they are unreliable. That is, they can be noisy, incomplete, or ambiguous (indicative of multiple underlying causes). Moreover, the model description may be incomplete. Hence, it may not be possible to explain all observations and some observations may need to be discarded. However, we can help the administrators by providing top alternative hypotheses for further investigation. For example, given an ambiguous observation that could be both a result of normal activity or malware infection, we can present at least two clusters, one that includes the normal activity, and one that includes the possibility of infection. The infection cluster itself can be a result of multiple causes, but we may want to show only one representative per cluster at first, allowing the user to request the remaining hypotheses from the clusters they are interested in. This diverse set of plans will not include unlikely or low-plausibility hypotheses. Hence, it is required to find a set of plausible hypotheses and then group these in some meaningful way before presenting the results. These plans (or equivalently, hypotheses) can be further evaluated automatically, by collecting and analyzing additional data.*

The malware detection problem or more generally the hypothesis generation problem can be encoded as an AI planning problem [25, 19], where the generated plans correspond to the hypotheses, and furthermore, the min-cost plans correspond to the plausible hypotheses. Plausible hypotheses are those that the domain expert believes to be more plausible compared to the other hypotheses. Plausibility can be encoded as 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.

Computing a set of low-cost plans or the top-k plans has the following benefits:

1. one can find plans that satisfy constraints that are not known apriori or are not easy to formalize;
2. by providing a list of alternative plans, one can explore the space of alternatives and hence gain better understanding of the properties of the problem and its optimal solution; and
3. in the hypothesis generation problem, finding the set of top plans is necessary to find the most accurate hypothesis, especially when the observations are not reliable and the model is incomplete.

Furthermore, grouping the top plans or the top-k plans into clusters adds the following benefits:

1. it helps users quickly navigate through the alternatives via cluster hierarchies,
2. the automated system, if in place, can also benefit from exploring cluster representatives rather than all plans.

In this paper, we propose an approach for finding a set of low-cost diverse plans for hypothesis generation. To this end, we propose

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

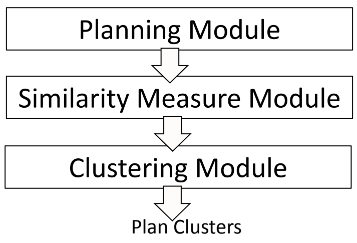

Figure 1. Framework

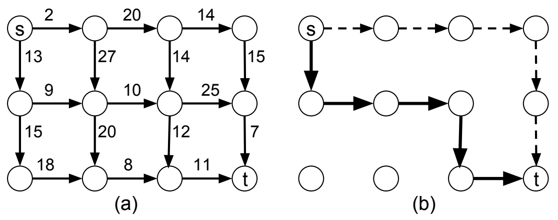

Figure 2. (a) shows a graph with source node s and terminal node t with edge lengths specified on the edges; (b) shows the shortest path in bold arrows and the second shortest path in dashed arrows.

a modular framework, as shown in Figure 1, where we first find a bounded set of low-cost plans, which we refer to as top-k plans, with respect to the given cost metric. We then cluster these plans based on their similarity and present the diverse plans by picking the representative plans from each cluster. The framework allows the use of different planning algorithms, similarity measures, and clustering algorithms in different combinations.

The *planning module* takes as input the planning problem with costs and produces a set of low cost plans. To solve the top-k planning problem, the problem of finding a set of k distinct plans with lowest cost, we propose use of a k-shortest paths algorithm. In particular, we developed an approach that allows us to solve the top-k planning problem by efficiently translating it into a k shortest path problem, and then solving that problem using the K^* algorithm [1]. We call the resulting top-k planner *TK**. Although K^* was developed for the k shortest paths problem, and has not been previously used in AI planning, it is efficient enough to be used in hypothesis generation problems of practical size, as experiments show.

The *similarity measure* module takes as input a pair of plans and decides whether the two are similar, by computing a similarity score and applying a threshold. Multiple similarity measures can be used in combination, and we evaluate a variety of domain-independent and domain-dependent measures.

Finally, the *clustering module* works with the result of the similarity measure module to produce the plan clusters. We evaluate the generated clusters for a set of hypothesis generation problem instances using several domain-independent and domain-dependent evaluation criteria including performance, number of clusters, and plan diversity. We also compare the performance and quality of solutions produced by our top-k planning framework and diverse planners.

The contributions of this paper are:

1. the decomposition of the problem of finding diverse high-quality plans into top-k planning and clustering stages, with configurable similarity measures;
2. a new top-k planner, *TK**, that applies K^* to planning problems;
3. efficient clustering algorithms for forming a set of diverse plans from a larger set of high quality plans; and
4. the evaluation of solution quality and performance of individual stages and overall framework on both manually crafted and random hypothesis generation problems and comparison to existing diverse planners. We find that our approach performs comparably to diverse planners in planning time and diversity, while finding solutions with consistently lower cost.

In what follows, we present the algorithms we use for finding the top-k plans, then we describe several relevant approaches that can be used for computing plan similarity, followed by an introduction of several single-pass algorithms for clustering plans. The experimental evaluation section includes separate subsections for top-k planning and plan clustering, as well as the comparison of the overall framework to diverse planners. We conclude with a discussion of related work and outline new opportunities for future research.

2 Top-k Planning

In this section, we will first formally define the top-k planning problem and then give the necessary background on the k shortest paths problem. We will then describe our top-k planning algorithm, TK*, that uses the K^* algorithm.

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 *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 an 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.

Note, while we indicated that the goal state, $\mathcal{G}$, is in a form of a final-state goal in the definition of R, our approach can be applied to temporally extended goals as well. Temporally extended goals, such as a sequence of observations of a system, either totally ordered or partially ordered, can be compiled away to a final-state goal following a compilation technique discussed in several papers (e.g., [21, 9]).

2.1 Background: K Shortest Path Problem

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 [12]. The following is a formal definition taken from Eppstein [6].

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*

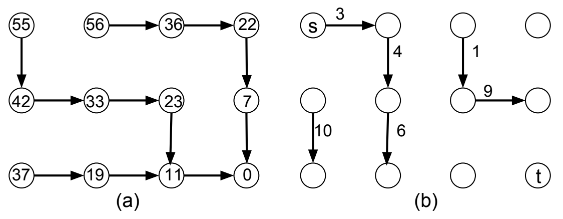

Figure 3. (a) shows the shortest path tree T and distance to destination t; (b) shows the side edges with their associated detour cost.

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 2 shows an example from Eppstein [6] illustrating 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.

2.2 Top-k Planning Using K^*

The K^* algorithm [1] is an improved variant of the Eppstein's k shortest paths algorithm [6] 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 $Q = (G, s, t, k)$, 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 explored *shortest path* tree T are called *tree* edges while all the missing edges (i.e., the edges in $G - T$) are called *sidetrack* edges. Each edge in G is assigned a number that measures 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 3 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 $P(G)$ represents a sidetrack edge. This is followed by the use of Dijkstra search in $P(G)$ 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. For more details see [6].

```
0. Read planning problem $R = (F, A, I, \mathcal{G}, k)$.
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).
```

Figure 4. *TK** planning algorithm applies K^* to search in planning state space.

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. Planners commonly deal with this challenge by relying on heuristic search algorithms like A* to dynamically expand only the necessary portion of the state graph during search, while being guided by a heuristic toward the goal (e.g., Fast-Downward [11]). The K^* algorithm combines the best of both worlds: it allows constructing the graph G dynamically using heuristic-guided A* search, while updating its equivalent of $P(G)$ to find k shortest paths.

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^*, following the algorithm above.

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]. Of course, step 2 may also terminate if no new links can be added.

Soundness and completeness of *TK** follows directly from the soundness and completeness of the K^* algorithm.

In our experiments, *TK** with constant 0 heuristic performs very well, and we have not experimented with other, potentially better performing heuristics. This is an interesting direction for improvement that could be explored in future work. Even though this is not a requirement for K^* in general, our implementation requires a consistent heuristic, which did not allow us to experiment with, for example, lookahead heuristics.

3 Finding Diverse Plans via Clustering

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, [2]), machine learning, and Data Management as part of the data cleaning process (e.g., [10]). Many survey papers exist on clustering algorithms (e.g, [28, 7]). 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. In this paper, we cluster the plans without specifying input parameters such as the number of clusters. This is because no prior knowledge on the number of clusters or the size of the cluster is available. Depending on the domain, there could be cases where many plans can be put into a single cluster due to high similarity, and there are also cases that the plans are all different, and the output must contain clusters of size 1.

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 then may choose to present only a subset of these clusters to the user or to the automated system for further investigation.

3.1 Plan Similarity

Finding if two plans are similar has been studied mainly under two categories: plan stability for replanning (e.g., [8]) and finding diverse plans (e.g., [16]). While some domain-dependent approaches exist (e.g., [15]), majority of recent research has focused on domain-independent measures. In this section, we first briefly discuss ways of representing a plan, and then discuss several similarity measures we consider.

Two plans can be compared based on their actions, states, or causal links [16]. In this paper, we focus on actions and states considering them as both sets and sequences. That is we consider both representing a plan by its set of actions as well as its set of states. We also consider representing a plan by its sequence of actions as well as its sequence of states. Our work is in line with prior work, except our states are not planning states (or set of propositions), but rather a possibly hidden behavioral Finite State Machine (FSM) states. They can be inferred from the semantics of the domain using machine learning or process mining. For example, in the malware detection example, a state can be "crawling", "infectionByNeighbor", or "infectionByDownload". Further, we represent a sequence of actions or states as a sequence of strings by treating each action or state as a symbol. This allows us to use a string similarity measure to compare plans. We also consider comparing plans solely based on their costs or their final states, as it may be enough to group plans based on just their costs (notion of plausibly) or the final state in the plan, a major factor in deciding what to do next in order to detect or predict malware.

Next, we go over the similarity measures we consider. Each similarity measure assigns a number between 0 (unrelated) or 1 (if they are the 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.

As we will see in the experiments, the choice of similarity measure influences the quality of the clusters, and our framework allows the users to choose any similarity measure or their combination.

3.1.1 Generalized Edit Similarity (GES)

GES [4] can be used to compare sequences of states (or actions) by viewing each state (or action) as a "token" in a string, and the sequence itself as a sequence of tokens. An important reason for choosing GES is that it not only considers the similarity between sequences, but also considers the similarity between tokens (i.e., states). Therefore, we are able to use any extra domain-dependent knowledge at hand about the relationship between states (or actions) to determine if two plans belong to the same cluster. This allows further semantic information to be included in similarity calculations.

GES takes two strings r and r', in our case the two strings represent sequence of states or actions, and computes their similarity score as a minimum transformation cost required to convert string r to r'. The two strings are first tokenized and then assigned a weight $w(t)$. We use a weight of 1 in our experiments. There are three kinds of transformations: insertion, deletion, and replacement. The token insertion cost is $w(t) \cdot c_{ins}$ where t is the inserted token in r and c_{ins} is the insertion factor which we set to 1. Token deletion has a cost of $w(t)$, where t is the deleted token from r. The replacement cost is $(1 - \text{similarity}(t_1, t_2)) \cdot w(t)$. We can use state/action relationships to determine the similarity between t_1 and t_2. For example, if one state is a child or a parent of another state (or if the two states share a same parent), similarity score is set to a higher number (for example, 0.5), else it is either 0 (if they are unrelated) or 1 (if they are the same).

Let r, r' be defined as the sequence of states (or actions) in plans π and π' respectively, then:

$$\text{sim}_{\text{GES}}(\pi, \pi') = 1 - \min\left(\frac{mct(r, r')}{wt(r)}, 1.0\right) \quad (1)$$

where $mct(r, r')$ is the minimum cost of the transformation between

	Gamer, top-1			Fast-Downward(A^*), top-1			*TK* *, top-50			*TK* *, top-1000		
Problem (# st., # obs.)	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
random (10,10)	0.65	0.85	1.41	0.15	0.20	0.23	0.05	0.06	0.12	0.32	0.38	0.42
malware (25,10)	1.09	1.63	1.86	0.49	0.49	0.50	0.06	0.07	0.11	0.23	0.32	0.44
random (50,10)	2.03	2.70	3.90	1.20	1.36	1.59	0.09	0.11	0.13	0.43	0.48	0.53
random (100,10)	11.70	15.27	23.64	4.09	4.85	5.27	0.18	0.29	0.44	0.67	0.75	0.81
random (10,60)	2.65	3.30	4.30	0.64	0.79	0.99	0.16	0.19	0.22	1.98	2.10	2.24
malware (25,60)	7.22	12.48	22.80	2.57	2.60	2.62	0.08	0.15	0.23	1.08	1.62	2.27
random (50,60)	110.95	203.40	291.04	7.65	8.65	9.59	0.36	0.53	0.68	2.24	2.52	2.75
random (100,60)	-	-	-	26.15	29.20	32.71	0.94	1.66	2.23	2.96	4.07	4.73
random (10,120)	6.22	10.82	17.22	1.25	1.60	2.01	0.32	0.36	0.40	4.07	4.24	4.44
malware (25,120)	39.58	83.25	164.48	5.48	5.51	5.56	0.14	0.23	0.40	2.04	2.86	4.19
random (50,120)	-	-	-	15.67	18.10	19.55	0.80	1.27	1.83	4.67	5.40	6.01
random (100,120)	-	-	-	69.96	75.25	79.57	2.31	4.27	6.04	6.55	9.13	11.37

Table 1. Top-k Planning Performance: minimum, average, and maximum planning time, in seconds, for 15 instances of each problem.

the two strings, and $wt(r)$ is the total weight of the string r. Note, this calculation normalizes the similarity score. This normalization is helpful since it allows to choose similarity threshold independently of the size of the plan.

Note that while sim_{GES} is asymetric, the effect of this is insignificant due to the use of single pass clustering algorithms that calculates each similarity score only once and that each clustering algorithm iterates over the top-k plans starting with the lowest-cost plan. Clustering algorithms will be described in the next section.

3.1.2 Jaccard Similarity

Jaccard similarity (inverse of the plan distance from [16]) measures the ratio of the number of actions (or states) that appear in both plans to the total number of actions (or states) appearing in one of them. Let A(π) be the set of actions (or states) in π, then:

$$\text{sim}_{\text{Jaccard}}(\pi, \pi') = \frac{|A(\pi) \cap A(\pi')|}{|A(\pi) \cup A(\pi')|} \tag{2}$$

3.1.3 Simple Equality

Let q and q' be defined as the final state (or the total cost) of plans π and π', then:

$$\text{sim}_{\text{Equality}}(\pi, \pi') = \begin{cases} 1 & \text{if } q = q' \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

3.2 Clustering Algorithms

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. Note, when we refer to computation of similarity between plans, it could be that one similarity measure or a weighted combination of similarity measures is used.

3.2.1 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 [10], 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.

3.2.2 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 [28]; 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.

3.2.3 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 [28].

4 Experimental Evaluation

We have four objectives in our experiments: (1) evaluate the performance of top-k planning by comparing it to planners finding a single cost-optimal plan, (2) evaluate the clustering algorithms and the sensitivity of the results to the threshold, (3) evaluate the different similarity measures we used, (4) evaluate against different diverse planners. In all experiments we used a dual 16-core 2.70 GHz Intel(R) Xeon(R) E5-2680 processor with 256 GB RAM.

4.1 Planning Problems

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. [25]. 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. The planning problems were represented in a STRIPS-like planning language recognized by our planner, as well as in PDDL[14] for Gamer and Fast-Downward.

To generate a random problem instance, we generated a random state transition system with a given number of states. In this setting the states of the state transition system do not map directly to planning states, instead we apply a domain transformation [25], compiling away temporary extended goals and adding penalty actions for imperfect explanations to generate the planning problem from the state transition system and the sequence of observations. As the result, the planning states combine the state of the state transition system with position in observation trace and other context information necessary to link observations to system state, generating a much larger state space for the planner.

We varied the size of the problem by changing the number of the states of the state transition system (for random systems) and the number of observations (for both random and manually crafted systems). Further, in all problems we randomly introduced a small fraction of random and missing observations in the generated observation sequence, to better simulate the conditions where generating multiple hypotheses is required, namely the presence of noise or incompleteness of models.

In addition to randomly generated problems, we used the manually crafted malware detection problem, described in [25] (also in Example), and referred to as "malware" in results. The malware detection problem requires generating hypotheses about the network hosts by analyzing the network traffic data. To make that possible, the state transition system includes the states of the host (e.g., infected with malware due to downloading an executable file or the *Command & Control Rendezvous* state via Internet Relay Chat (IRC)) and transitions between these states, as well as a many-to-many correspondence between states and observations.

4.2 Top-k Planning Performance

In Table 1, we compare the performance of our top-k planner, *TK**, with k=50 and k=1000. We compare to Gamer [13] (Gamer 2014 version, seq-opt-gamer-2.0) and Fast-Downward [11] (2015 version, with A^*). Both find a single cost-optimal plan, which is equivalent to k=1. Planning time was measured on the same randomly generated problem instances for two different kinds of domains, "malware" and "random", and aggregated over 15 instances of each size, where size was controlled by two domain-specific parameters (the number of system states and observations). We enforced a time limit of 300 seconds. Rows containing "-" are those where none of 15 instances were solved within the time limit.

Overall, *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. Some of the larger instances proved too difficult for Gamer, and it exceeded the time limit. We can also observe that while the worst-case complexity of *TK** includes $O(kn)$ term, we have observed relatively small relative differences in planning time with increasing k, with absolute difference limited by a few seconds, and with relative difference decreasing as problem size increased.

Since *TK** performs A^* search to find top-k plans when k=1, and *TK** top-50 performs similarly to top-1, *TK** top-50 can be expected to perform similarly to Fast-Downward A^*, which we have observed. Although *TK** is not fully PPDL compliant, there is no significant difference in language expressivity or knowledge provided to planners, and the difference in performance most likely is explained by more efficient implementation and differences in preprocessing in *TK**. We do not fully understand why Gamer performed relatively poorly on large problem instances. It was natural to expect a cost-optimal planner to find one optimal plan just as fast or faster than a top-k planner would require to find k plans. 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.

4.3 Evaluation of Clusters

We separate the evaluation of the clustering algorithms from the similarity measures. However, we use the following sets of evaluation measures in both cases: time, measured in second, number of similarity comparisons (# Comp) in thousand, number of clusters (# C), and the following six metrics:

- M1: percentage of clusters with the same final state,
- M2: percentage of clusters with the same last three states,
- M3: inter-cluster diversity via uniqueness,
- M4: inter-cluster diversity via stability,
- M5: intra-cluster diversity via uniqueness, and
- M6: intra-cluster diversity via stability.

M1 and M2 are examples of a domain-dependent metric while the rest could be thought of as domain-independent measures. We measure stability and uniqueness using the following formula from [20]. Note, we modified these formula to make it a number between 0 and 1. Also for intra-cluster evaluations, Π is the set of plans within a cluster and we take the average over all clusters. For inter-cluster evaluations, Π is the set of all cluster representative plans which we take to be the lowest-cost plan in each cluster. 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 $|\Pi| \geq 1$:

$$\text{Diversity}_{\text{stability}}(\Pi) = \frac{\sum\limits_{\pi_i, \pi_j \in \Pi, i \neq j} [1 - \text{sim}_{\text{Jaccard}}(\pi_i, \pi_j)]}{|\Pi| \times (|\Pi| - 1)} \quad (4)$$

$$\text{Diversity}_{\text{uniqueness}}(\Pi) = \frac{\sum_{\pi_i, \pi_j \in \Pi, i \neq j} \begin{cases} 0 & \text{if } \pi_i \setminus \pi_j = \emptyset \\ 1 & \text{otherwise} \end{cases}}{|\Pi| \times (|\Pi| - 1)} \quad (5)$$

	θ	**Time**	**# of**	**# of**	**Last state(s)**		**Inter-cluster**		**Intra-cluster**	
		(sec)	**Comp**	**C**	**M1**	**M2**	**M3**	**M4**	**M5**	**M6**
Center	0.65	**0.66**	**10K**	38	74%	45%	0.77	0.51	**0.60**	0.20
Link	0.75	0.82	17K	67	80%	56%	**0.78**	0.49	**0.60**	0.15
	0.85	1.32	36K	142	89%	75%	0.77	0.46	0.64	**0.09**
Single	0.65	1.83	48K	**26**	72%	43%	0.76	**0.54**	0.62	0.20
Link	0.75	2.18	67K	48	77%	54%	0.77	0.52	0.62	0.16
	0.85	3.28	106K	115	86%	71%	0.77	0.49	0.65	**0.09**
Avg-	0.65	14.27	356K	41	75%	47%	0.76	0.50	0.61	0.20
Link	0.75	12.14	329K	72	82%	60%	0.77	0.47	0.61	0.15
	0.85	11.37	330K	152	**91%**	**77%**	0.77	0.46	0.64	**0.09**

Table 2. Comparisons of the clustering algorithms.

First	**Second**	**Time**	**# of**	**Last state(s)**		**Inter-cluster**		**Intra-cluster**	
		(sec)	**C**	**M1**	**M2**	**M3**	**M4**	**M5**	**M6**
GES-S		4.04	26	52%	37%	0.87	**0.68**	0.61	0.23
GES-A		3.73	63	57%	47%	0.84	0.63	0.62	0.18
Jaccard-S		4.55	79	65%	58%	**0.96**	0.65	**0.55**	0.08
Jaccard-A		4.11	111	67%	61%	0.83	0.57	0.66	0.11
Last State		6.23	**9**	**100%**	33%	0.70	0.43	0.59	0.26
Cost		**2.32**	13	62%	40%	0.62	0.34	0.65	0.25
GES		3.87	37	54%	41%	0.87	0.67	0.60	0.20
Jaccard		4.28	92	66%	59%	0.90	0.61	0.58	0.09
All		2.95	63	82%	58%	0.76	0.47	0.61	0.17
Last State	GES-S	9.08	33	**100%**	53%	0.71	0.46	0.60	0.20
Last State	GES-A	8.81	74	**100%**	65%	0.70	0.43	0.62	0.15
Last State	Jaccard-S	9.39	106	**100%**	**76%**	0.85	0.54	0.56	**0.06**
Last State	Jaccard-A	9.06	131	**100%**	**76%**	0.73	0.43	0.66	0.09
Cost	GES	3.55	63	73%	60%	0.69	0.41	0.64	0.17
Cost	Jaccard	3.64	155	82%	**76%**	0.77	0.45	0.65	**0.06**

Table 3. Comparisons of the similarity measures.

For both uniqueness and stability we compare plans while representing them by their set of states. We also tested with actions but the results were comparable and not shown. M3-M6 are distance measures with values between 0 (the same) and 1 (different - farthest apart). For M3 and M4, larger the number, more diverse the plans are since we find the diverse plans by presenting only the representative plans from each cluster. For M5 and M6, smaller the number, similar the plans are within a cluster. Hence, the ideal algorithm or clustering measure maximizes M3 and M4 and minimizes M5 and M6. The numbers shown in Table 2 and 3 are averages over all planning problems (5 instances of each size). The bold numbers indicate the best numbers in each case.

Summary of our results with respect to the clustering algorithms is shown in Table 2. 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 the evaluation metrics, the results does not show a superior clustering algorithm: Average-Link is slightly better with respect to M1 and M2, Single-Link produces slightly more diverse plans with respect to M4, and Center-Link also has slightly better numbers with respect to M5. Hence, the clustering algorithms alone do not seem to influence the metric evaluations. However, Center-Link is the best performing algorithm with respect to time. It also compares fewer plans and produces medium size clusters.

Summary of our results with respect to the similarity measures is shown in Table 3. The top part of the table shows the result where a particular similarity measure is used: GES-S and GES-A indicate that we used equation 1, representing the plan by its sequence of states (or actions); Jaccard-S and Jaccard-A indicate that we used equation 2, representing the plan by its set of states (or actions); and "Last State" and "Cost" indicate we used equation 3. The middle part of the table indicates that we used a combination of similarity measures: GES indicates that we used both GES-S and GES-A (assigning equal weights to both); Jaccard indicates that we used both Jaccard-S and Jaccard-A; and "All" indicates that we used all six similarity measures assigning equal weights to each. The bottom part of the table shows the results for when we first cluster all plans based on the similarity measure shown under the "First" column, then within each cluster, run the clustering algorithm again using the similarity measure shown under the "Second" column.

The results show that grouping based on cost may be fastest and grouping based on the last state satisfies M1 (it forces it to be true). However, these similarity measures give the worst results with respect to the nearly all other metrics. On the other hand, using just Jaccard-S produces most diverse plans with respect to uniqueness (best number for M3) and produces similar plans within a cluster (best numbers for M5 and M6) but it suffers in the M1 and M2 categories. GES-S also produces most diverse plans with respect to stability (largest M4 value). While the time and number of clusters is still reasonable, combining all of the metrics (middle part of the table) do not provide better results. However, the best results are found when we combine measures and run the clustering algorithm for the second time. At the expense of time increase, M1, M2, and M6 results are best when we first group based on the last state and then use Jaccard-S. The M3 and M5 numbers are also close to the best numbers. In conclusion, if time is most important then one can just group based on cost. If having the best results for domain-dependent measures such as M1 and M2 is important, one can enforce these metrics when clustering. To only find diverse plans, you can use either GES-S or Jaccard-S. Finally, if satisfying both the domain-dependent and domain-independent metrics is important then combining similarity measures and using for example "last state" followed by "Jaccard-S" will give the best results.

4.4 Comparison With Diverse Planners

We selected two representative diverse planners, LPG-d [16] (with d=0.1) and Div (Multi-queue A^* MQA_{TD}) [20], and compared to our implementation that included top-k and Average-link clustering,

	Top-k + Average Link				LPG-d				Div			
	T	Cost	M4	M3	T	Cost	M4	M3	T	Cost	M4	M3
(25, 5)	1	1502	0.51	1	1	3513	0.80	1	1	1789	0.36	0.37
(25, 10)	1	1586	0.41	0.99	59	8426	0.84	1	1	3861	0.44	0.54
(25, 20)	3	1492	0.20	0.99	384	16520	0.87	1	1	7262	0.46	0.53

Table 4. Comparison to diverse planners: planning time, T, in seconds, average plan cost and plan diversity on the malware domain. M3 measures plan diversity via uniqueness. M3 measures plan diversity via stability.

using measures M3 and M4 based on Jaccard similarity. The results in Table 4 are averaged over 5 instances of each size, with 30 minutes time limit. The top-k approach produced 50 plans while LPG-d and Div produced at most 10.

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 poor diversity. As expected, the top-k approach produces the lowest average cost with somewhat lower diversity.

5 Related Work

In prior work, we have looked at several problems involving hypothesis generation by planning, including a short version of the present work [24], a study of planner-generated hypotheses in goal and plan recognition settings [23], and applications in malware detection and healthcare [25, 19, 18].

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., [20, 5, 5]). When no preferences or quality or cost metric is provided, it is argued that generating a set of diverse plan is the right approach [16]. 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., [26, 3]). On the other hand, 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 [16]. In particular, Sroka and Long [27] 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

The contributions of this paper are the following: 1) the planning framework based on the decomposition of the problem of finding diverse high-quality plans into top-k planning and clustering stages, with configurable similarity measures; 2) a new top-k planner, *TK**, that applies K^* algorithm to planning problems; 3) efficient clustering algorithms for forming a set of diverse plans from a larger set of high quality plans; and 4) the evaluation of solution quality and performance of individual stages and overall framework on both manually crafted and random hypothesis generation problems and comparison to existing diverse planners.

Our framework allows plugging in different top-k planning techniques, different clustering algorithms, and different similarity measures. We evaluate each of these components separately before carrying out the end-to-end evaluation. Our experiments show that planning time required for top-k planning is comparable to cost-optimal planning that finds a single cost-optimal plan using, for example, Fast-Downward. Our empirical evaluation of the three clustering algorithms we proposed for this task show that Center-Link is the best performing algorithm for our setting as it requires less time, compares fewer plans, and produces medium size clusters, while performing similarly to other algorithms in all evaluation metrics. Our findings with respect to similarity measures show that depending on what is most important, the user can choose the best similarity measure (or a combination). Finally, comparing the end-to-end performance of our framework to diverse planners we find that our approach performs comparably to diverse planners in planning time and diversity, while producing diverse plans with consistently lower cost.

While we considered clustering as a post-processing step to finding top-k plans, it might be possible to both guide the search towards diverse plans as well as towards min-cost plans. In future we plan to study this problem and evaluate whether it provides any significant improvements to our results.

REFERENCES

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

[2] 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).

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

[4] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani, 'Robust and Efficient Fuzzy Match for Online Data Cleaning', in *ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pp. 313–324, (2003).

[5] 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)*, pp. 946–951, (2011).

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

[7] 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).

[8] 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)*, pp. 212–221, (2006).

[9] Patrik Haslum and Alban Grastien, 'Diagnosis as planning: Two case studies', in *International Scheduling and Planning Applications woRKshop (SPARK)*, pp. 27–44, (2011).

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

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

[12] 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).

[13] Peter Kissmann, Stefan Edelkamp, and Jörg Hoffmann, 'Gamer and Dynamic-Gamer symbolic search at IPC 2014', in *8th International Planning Competition Booklet (IPC-2014)*, (2014).

[14] Drew V. McDermott, 'PDDL — The Planning Domain Definition Language', Technical Report TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, (1998).

[15] Karen L. Myers and Thomas J. Lee, 'Generating qualitatively different plans through metatheoretic biases biases', in *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI)*, pp. 570–576, (1999).

[16] Tuan Nguyen, Minh 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).

[17] Miquel Ramírez and Hector Geffner, 'Plan recognition as planning', in *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1778–1783, (2009).

[18] Anton Riabov, Shirin Sohrabi, Octavian Udrea, and Oktie Hassanzadeh, 'Efficient high quality plan exploration for network security', in *International Scheduling and Planning Applications woRKshop (SPARK)*, (2016).

[19] 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 *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 282–290, (2015).

[20] 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)*, pp. 253–261, (2014).

[21] 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)*, pp. 26–36, (2010).

[22] Shirin Sohrabi, Jorge A. Baier, and Sheila A. McIlraith, 'Preferred explanations: Theory and generation via planning', in *Proceedings of the 25th National Conference on Artificial Intelligence (AAAI)*, pp. 261–267, (2011).

[23] Shirin Sohrabi, Anton Riabov, and Octavian Udrea, 'Plan recognition as planning revisited', in *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*, (2016).

[24] Shirin Sohrabi, Anton Riabov, Octavian Udrea, and Oktie Hassanzadeh, 'Finding diverse high-quality plans for hypothesis generation', in *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI)*, (2016).

[25] 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)*, (2013).

[26] 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)*, pp. 2016–2022, (2007).

[27] 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)*, pp. 306–317, (2012).

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