Efficient Alternative Route Planning in Road Networks

Ahmed Fahmin, Bojie Shen, Muhammad Aamir Cheema, Adel N. Toosi, and Mohammed Eunus Ali

***Abstract*—Alternative route planning requires finding k alternative paths (including the shortest path) between a given source and target. These paths should be significantly different from each other and meaningful/natural (e.g., must not contain loops or unnecessary detours). While there exists many work on finding high-quality alternative paths, the existing techniques are computationally expensive and are unable to accommodate the high volume of queries required by modern navigation systems. To address this, in this paper, we propose an efficient approach to compute high-quality alternative paths. Our approach employs hub-labeling to efficiently identify candidate alternative paths. The candidate paths are then ranked considering multiple quality metrics and the top-k alternative paths are returned. We propose several non-trivial optimizations to significantly improve the computation time. In our experimental study, we conduct experiments on three diverse real-world road networks and compare our proposed algorithm against six state-of-the-art algorithms. The results demonstrate that our algorithm is not only up to 3 orders of magnitude faster compared to most algorithms but also consistently generates alternative paths that are comparable or even superior in terms of quality across various metrics.**

***Index Terms*—Road networks, alternative routes, route planning, shortest paths**

I. Introduction

In road networks, shortest path queries refer to the task of finding the most cost-effective path (e.g., in terms of travel time, distance, or fuel consumption) from a given source location to a target location. The shortest path queries are among the most fundamental operations on road networks and are commonly used in route planning services such as Google Maps and Bing Maps, as well as in various high-level planning tasks such as location-based services, ride-sharing, traffic assignments, and more. Due to their various applications, the shortest path problem has been extensively studied not only on road networks [1]–[4], but also in other settings such as on social networks [5], indoor venues [6], game maps [7], to name a few.

In many real-world scenarios, it is desirable to provide users not only with the shortest path but also to offer several alternative paths for them to choose from. For example, modern navigation systems often provide users with multiple alternative routes, allowing them to select a path that best suits their preferences for traveling. In many cases, the optimal route to a destination differs from what an individual prefers, influenced by factors such as their familiarity with the area or personal biases towards specific roads. To be useful to the users, the alternative routes suggested by the navigation system must be short, meaningful (e.g., without unnecessary detours) and substantially different from each other. It is worth noting that returning the k shortest paths is not a suitable solution, as they are often very similar to one another (i.e., exhibiting high overlap) and may involve needless detours. Figure 1 shows three alternative paths from Brooklyn Museum to New York Hall of Science displayed by Google Maps. The blue path is the shortest path (in terms of travel time) and the other two paths have comparable travel times. Note that these paths are short, significantly different from each other, and are meaningful (e.g., do not contain any small detours or cycles).

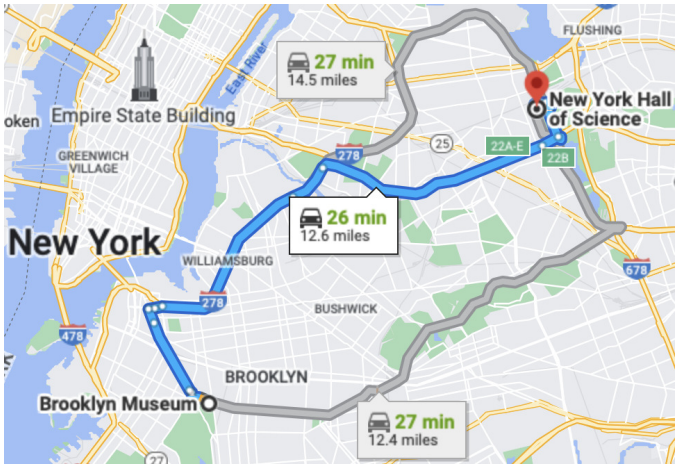

Fig. 1: Three alternative paths from Brooklyn Museum to New York Hall of Science (Google Maps).

In addition to its importance in navigation services, alternative pathfinding has many applications in various other domains. For instance, in traffic shaping [8], enabling self-driving vehicles to autonomously select alternative paths can effectively balance or reduce traffic loads on the road network. In contingency planning [9], the alternative paths often serve as backup plans, assisting drivers in avoiding road accidents or road closures. Additionally, in automated warehousing [10], the alternative paths are frequently utilized to prevent collisions when multiple robots are concurrently executing their tasks. In each application setting, the efficient computation of high-quality alternative paths is of utmost importance.

Limitation of Existing Works: Due to their importance, a considerable amount of research has focused on finding

Manuscript received ...

A. Fahmin (email: ahmed.fahmin@monash.edu), B. Shen (email: bojie.shen@monash.edu), M. A. Cheema (email: aamir.cheema@monash.edu) and A. N. Toosi (email: adel.n.toosi@monash.edu) are with the Faculty of Information Technology, Monash University, Melbourne, Australia.

M. E. Ali (email: eunus@cse.buet.ac.bd) is with Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh

alternative paths in road networks, e.g., [11]–[15] and references therein. The existing body of literature has taken diverse approaches to alternative routing. Some prominent techniques for computing alternative paths involve the use of edge penalties [16]–[18] or plateaus [19]. However, these approaches often adapt a greedy strategy for identifying alternative paths, relying heavily on searching in a frequently updated graph or constructing shortest path trees, thus lead to high computational costs. Other methods incrementally search for alternative paths until a collection of dissimilar paths has been found [20]–[23], or generate an extensive pool of candidate paths and apply ranking criteria for alternative path selection [19], [24], [25]. These existing techniques suffer from high computation costs, may taking up to a second to compute alternative paths. The performance bottleneck inherent in these techniques presents a significant barrier to their real-world application in large-scale navigation systems that require computing tens of thousands of routes per second. To overcome this challenge, there is a need to design scalable algorithms capable of efficiently computing alternative routes.

Our Contributions: To overcome the limitations of the existing works, in this paper, we propose an efficient algorithm that leverages Hub Labeling (HL) [26], a state-of-the-art approach for computing shortest paths, to efficiently compute high-quality alternative routes. Our algorithm bypasses resource-intensive graph searches used by the existing algorithms and instead exploits hub labels to quickly find high-quality alternative routes, thereby enhancing efficiency. To further enhance efficiency, we present a set of filtering rules to eliminate low-quality candidate paths. Additionally, we introduce several non-trivial optimisations designed to speed up candidate path construction and calculation of path quality metrics. We conduct an extensive experimental study using three diverse road networks (Melbourne, Copenhagen and Dhaka) and compare against six state-of-the-art algorithms. The results show that our algorithm is up to 3 orders of magnitude faster compared to the existing algorithms while the path quality is comparable or sometimes superior to the existing approaches. For example, compared to X-CHV [24] which is the fastest existing algorithm, our algorithm does not only compute alternative paths 4-8 times faster than X-CHV but also consistently returns better quality paths considering a variety of quality metrics. Similarly, compared to SVP+ [23], another state-of-the-art algorithm, our algorithm is around 3 orders of magnitude faster on average whereas the alternative paths returned by our approach are slightly better for some quality metrics and slightly worse for some other. These findings demonstrate the effectiveness and efficiency of our proposed algorithm, indicating its potential for practical applications in real-world scenarios.

Our contributions in this paper are summarized below:

- We introduce a novel algorithm, called Hub-VAR, for efficiently computing k alternative routes in a road network. What sets Hub-VAR apart from existing algorithms is its ability to avoid expensive search processes by leveraging hub labels. This approach enables the algorithm to efficiently retrieve a pool of candidate paths.
- We introduce a set of filtering rules to eliminate low-quality candidate and employ several optimization techniques to further enhance the efficiency of the algorithm.
- We perform an extensive experimental study using three diverse real-world datasets. The results of our study demonstrate the promise of our algorithm in terms of both effectiveness and efficiency.

The paper is organized as follows. We discuss the problem definition, path quality metrics and hub-labeling in Section II. Related work is presented in Section III. Our efficient algorithm, called Hub-VAR, is presented in Section IV followed by experiments and conclusion in Sections V and VI, respectively.

II. Preliminaries

A. Problem Formulation

Similar to the existing research [17], [23], [24], for ease of presentation, we assume undirected road networks. However, our techniques readily extend to directed networks. Let $G = (V, E, w)$ be an undirected graph, with vertices V, edges $E \subseteq V \times V$, and $w : E \to \mathbb{R}^+$ a weight function that maps each edge $e \in E$ to a non-negative weight $w(e)$, e.g., travel time, distance, etc. A **path/alternative path** between a start s and a target t in G is a sequence of vertices $\langle v_0, v_1, v_2, \ldots, v_{k-1}, v_k \rangle$, where $v_0 = s$, $v_k = t$, and $(v_i, v_{i+1}) \in E$ for $0 \leq i < k$. We use $|P|$ to denote the length of path, where $|P| = \sum_{i=0}^{k-1} w(v_i, v_{i+1})$. A **sub-path** $P(v_x, v_y)$ is a contiguous subset of vertices $\langle v_x, \ldots, v_y \rangle$ of P, where $x < y$ and $\langle v_x, \ldots, v_y \rangle \subseteq P$. We use $d^P(v_x, v_y)$ to denote its length, i.e., $d^P(v_x, v_y) = \sum_{i=x}^{y-1} w(v_i, v_{i+1})$. A **shortest path** $sp(s,t)$ between s and t is a path such that $|sp(s,t)|$ is minimum among all possible paths between s and t. We denote its length (i.e., shortest distance) as $sd(s,t)$. We use $sp(s,v,t)$ to denote the path $sp(s,v) \cdot sp(v,t)$ where $\cdot$ is concatenation operation. $sd(s,v,t)$ denotes the length of $sp(s,v,t)$ (i.e., $sd(s,v,t) = sd(s,v) + sd(v,t)$).

Definition 1. ***(Alternative Pathfinding).*** *Given a graph G, a source s, a target t, and a positive integer k, alternative pathfinding requires computing up to k alternative paths (including the shortest path) between a start point s and a target point t in a road network.*

Intuitively, an alternative path can be any path between s and t; however, the alternative paths that are short, significantly different from each other and do not contain unnecessary detours are of users' interest. As noted in the previous works [11], there is no agreed definition of what constitutes a set of "good" alternative routes. This is because the "goodness" of the alternative routes is mostly subjective. Therefore, the existing works typically do not include the definition of "goodness" in the problem formulation. Instead, path quality metrics are defined separately and are used in the experimental study. Next, we briefly explain the most popular metrics to evaluate the quality of a set of alternative paths P and, in our experimental study, we use these metrics.

B. Path Quality Metrics

Existing works have defined a variety of metrics to evaluate the quality of a set of alternative paths P returned by an algorithm. We introduce these below.

a) Similarity: Similarity [11] of a set of alternative paths P, denoted as $Sim(P)$, measures the pairwise similarity of paths in P and corresponds to the maximum pairwise similarity. Formally, the similarity is defined as:

$$Sim(P) = \max_{\forall (P_i,P_j) \in P : i \neq j} \frac{|P_i \setminus P_j|}{|P_i \cup P_j|} \quad (1)$$

where $|P_i \setminus P_j|$ (resp. $|P_i \cup P_j|$) denotes the total length of the overlap (resp. union) of two paths P_i and P_j. Note that, the alternative paths P should be significantly different from each other, thus lower $Sim(P)$ is better.

b) Distance Ratio: The alternative paths should not be much longer than the shortest path. The distance ratio of P is defined by the relative difference w.r.t the shortest distance:

$$DR(P) = \frac{|P| - sd(s,t)}{sd(s,t)} \quad (2)$$

The alternative path P with lower $DR(P)$ is better, because in real-world navigation scenarios, the users are unlikely to choose alternative paths that are substantially longer than the shortest path. The minimum possible $DR(P)$ is zero which is when each alternative path has the same length as the shortest path. Given a set of alternative paths P, distance ratio of P is the maximum distance ratio among all paths in P, i.e., $DR(P) = max_{\forall p \in P} DR(P)$.

c) Bounded Stretch: Stretch [18], [27] of a path reflects detour length compared to the shortest distance. Given a subpath $P(v_x, v_y)$ of an alternative path P, the stretch of $P(v_x, v_y)$ is defined as the ratio of the path distance (i.e., $d^P(v_x, v_y)$) and the shortest distance (i.e., $sd(v_x, v_y)$) between v_x and v_y. The bounded stretch $BS(P)$ of the path P is the maximum stretch of any of its subpaths:

$$BS(P) = \max_{\forall P(v_x,v_y) \in P} \frac{d^P(v_x, v_y)}{sd(v_x, v_y)} \quad (3)$$

Note that an alternative path with smaller bounded stretch P is preferred because it indicates that detours compared to the shortest paths are short. A shortest path $sp(s,t)$ has a bounded stretch of 1 which is the minimum possible bounded stretch. Given a set of alternative paths P, the bounded stretch of P is the maximum bounded stretch of all paths in P, i.e., $BS(P) = \max_{\forall p \in P} BS(P)$.

d) Local Optimality: Another popular quality metric for evaluating alternative paths is local optimality [18], [27], which is a measure of the optimality of the subpaths of an alternative path. Given a subpath $P(v_x, v_y)$ of P, $P(v_x, v_y)$ is a suboptimal path if it is longer than the shortest path between v_x and v_y, i.e., $d^P(v_x, v_y) > sd(v_x, v_y)$. Let $L(P)$ be the shortest suboptimal subpath of P. This implies that any subpath in P that has length less than L must be an optimal path [27], i.e., $\forall P(v_x, v_y) \in P | d^P(v_x, v_y) < L(P)$: $d^P(v_x, v_y) = sd(v_x, v_y)$. If there is no suboptimal subpath in P because P is an optimal (i.e., shortest) path, $L(P)$ is assumed to be infinity. The local optimality $LO(P)$ normalizes $L(P)$ w.r.t. $sd(s,t)$. Mathematically, this is represented as:

$$LO(P) = \frac{L(P)}{sd(s,t)}$$

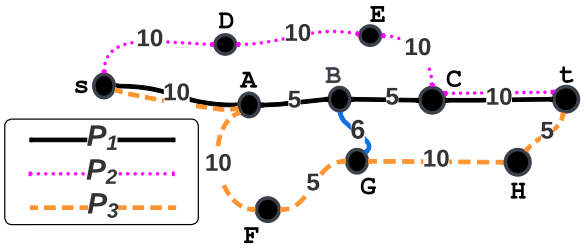

Fig. 2: Illustrating different quality metrics

$$= \begin{cases} 1 & P \text{ is optimal} \\ \min_{\forall (v_x,v_y): d^P(v_x,v_y) > sd(v_x,v_y)} \frac{d^P(v_x,v_y)}{sd(s,t)} & \text{else} \end{cases} \quad (4)$$

Let P be an alternative path with $LO(P) = x$. This implies that the shortest suboptimal path of P has length $x \cdot sd(s,t)$, i.e., every sub-path of P with length less than $x \cdot sd(s,t)$ is an optimal path. A low value of $LO(P)$ implies that there are short subpaths in P that are suboptimal. Therefore, higher $LO(P)$ is better as it indicates the alternative path P does not have obvious flaws such as loops or small detours (e.g., does not have short suboptimal paths). For example, a high local optimality, such as 0.8, indicates that every subpath of P shorter than 80% of the shortest path between s and t is guaranteed to be an optimal path. Given a set of alternative paths P, the local optimality of P is the minimum local optimality of all paths in P, i.e., $LO(P) = \min_{\forall p \in P} LO(P)$. For more details of these quality metrics, we refer the interested readers to the previous works [18], [24], [27].

Example 1. *Fig. 2 shows three paths P_1, P_2 and P_3 between s and t where P_1 is the shortest path with length 30. Similarity of both paths with the shortest path is the same, i.e., $Sim(\{P_1, P_2\}) = Sim(\{P_1, P_3\}) = 10/60 = 0.167$. The total length of both P_2 and P_3 is 40. Thus, their distance ratio is the same, i.e., $DR(P_1) = DR(P_2) = (40 - 30)/30 = 1/3$. The bounded stretch for P_2 is 1.5 because $d^{P_2}(s,C)/sd(s,C) = 30/20 = 1.5$ (which corresponds to the maximum stretch of all subpaths in P_2). For P_3, the bounded stretch is also 1.5 because $d^{P_3}(A,t)/sd(A,t) = 30/20 = 1.5$. For P_2, the shortest suboptimal path on P_2 is $\langle s, D, E, C \rangle$ with length 30. Thus, $LO(P_2) = 30/30 = 1$. For P_3, the shortest suboptimal path on P_3 is $\langle A, F, G \rangle$ with length 15. Thus, $LO(P_3) = 15/30 = 0.5$. Note that P_2 and P_3 have the same length, bounded stretch and similarity with the shortest path but P_2 is better than P_3 in terms of the local optimality.*

When assessing the path quality for alternative paths, it is important to emphasize that all four quality metrics are similarly important and should be evaluated simultaneously. The overall quality of alternative paths cannot be represented by any single metric in isolation. In the context of alternative pathfinding, the objective is typically to identify a set of "high-quality" alternative paths that exhibit low scores in $DR(P)$, $Sim(P)$, $BS(P)$, and high scores in $LO(P)$ simultaneously.

C. Hub Labeling (HL)

Computing shortest paths in road networks has been extensively studied and a variety of techniques have been proposed including Dijkstra's algorithm [1], bi-directional search [28], heuristics estimation [2], [29], edge labeling [30], [31], contraction hierarchies [4], [32], and hub labeling [26], [33], to name a few. Since we exploit hub labeling in our solution, we summarise this approach in this section.

Hub labeling (HL) is the state-of-the-art approach for efficiently computing the shortest path/distance in road networks. Given an input graph G, HL stores a set of hub labels $H(v_j)$ for each vertex $v_j \in V$. Each label is a tuple $(v_i, d_{v_i v_j}, p_{v_i v_j}) \in H(v_j)$ that includes: (i) a hub vertex $v_i \in V$; (ii) the shortest distance $d_{v_i v_j}$ between the hub vertex v_i and v_j; and (iii) the predecessor $p_{v_i v_j}$ of v_j (i.e., the first vertex before v_j on the shortest path $sp(v_i, v_j)$ between v_i and v_j). It is crucial for the hub labeling to satisfy coverage property formally described below.

Lemma 1. *(Coverage Property) For every pair of reachable vertices $v_i \in V$ and $v_j \in V$, the hub labeling must ensure that there exists a common hub vertex v_k in both $H(v_i)$ and $H(v_j)$, such that v_k is on the shortest path between v_i to v_j.*

Any hub labeling approach that satisfies coverage property can be used to find the shortest distance/path between v_i and v_j by using the hub labels corresponding to the common hub node v_k (as we explain in detail shortly). Reducing the total number of labels stored while ensuring the coverage property can improve the query performance. However, computing the minimum number of hub labels that satisfies the coverage property is known to be NP-hard [34]. Therefore, many heuristic approaches have been proposed. Although our approach does not rely on any particular hub labeling technique, in our implementation, we use SHP [33], a state-of-the-art hub labeling approach. The details on how the hub labels are computed are omitted and readers are referred to the existing works, e.g., [33]. Next, we show how to efficiently compute the shortest distance and path using the stored hub labels.

a) Computing Shortest Distance: Based on Lemma 1, the shortest distance between s and t is calculated by scanning over the sorted label set $H(s)$ and $H(t)$, and using Eq. (5).[1]

$$sd(s,t) = \min_{v_i \in H(s) \setminus H(t)} (d_{sv_i} + d_{v_i t}) \tag{5}$$

Note that the computational complexity of finding the shortest distance is $O(|H(s)| + |H(t)|)$, where $|H(x)|$ denotes the number of labels in $H(x)$.

b) Computing Shortest Path: In order to retrieve the shortest path, the hub labeling algorithm involves a two-step process: (i) using Eq. (5) to identify the hub labels $(v_i, d_{v_i s}, p_{v_i s})$ and $(v_i, d_{v_i t}, p_{v_i t})$ from $H(s)$ and $H(t)$, respectively, where v_i is the common hub vertex on the shortest path $sp(s,t)$; and (ii) retrieving the shortest subpath $sp(s, v_i)$ and $sp(v_i, t)$, and concatenating them to obtain the shortest path $sp(s,t)$, i.e., $sp(s,t) = sp(s,v_i) \cup sp(v_i,t)$. The step (i) is the same as computing shortest distance. To obtain the subpath $sp(s, v_i)$ in step (ii), we follow the predecessor $p_{v_i s}$ obtained from the label $(v_i, d_{v_i s}, p_{v_i s})$ and recursively extract the next predecessor from s to v_i. Each extraction of a predecessor requires a linear search over the label set of $H(p_{v_i s})$. The same process is repeated to obtain $sp(v_i, t)$. The complexity of step (ii) is $O(N \cdot SP)$, where N is the average label size of a node and SP is the number of vertices on $sp(s,t)$.

[1] In this paper, we often use the notation $H(s) \cap H(t)$ (resp. $H(s) \cup H(t)$) to refer to the common (resp. union of) hub vertices stored in $H(s)$ and $H(t)$.

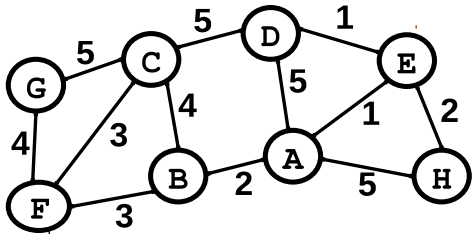

Fig. 3: An example of a small road network graph.

Vertex	Hub labels
A	(A, 0, A)
B	(A, 2, A), (B, 0, B)
C	(A, 6, B), (B, 4, B), (C, 0, C)
D	(A, 2, E), (C, 5, C), (D, 0, D)
E	(A, 1, A), (C, 6, D), (D, 1, D), (E, 0, E)
F	(A, 5, B), (B, 3, B), (C, 3, C), (F, 0, F)
G	(A, 9, F), (B, 7, F), (C, 5, C), (F, 4, F), (G, 0, G)
H	(A, 3, E), (C, 8, E), (D, 3, E), (E, 2, E), (H, 0, H)

TABLE I: Hub labeling for the graph in Fig. 3

Example 2. *Consider Table I that shows hub labels for each vertex of the graph in Fig. 3. To compute the shortest distance $sd(F,H)$ between F and H, HL iteratively scans the label set $H(F)$ and $H(H)$ and finds two common hub vertices A and C. When the common hub vertex A is found with label $(A,5,B)$ in $H(F)$ and label $(A,3,E)$ in $H(H)$, $sd(F,H)$ is updated to be $sd(F,H) = d_{AF} + d_{AH} = 5 + 3 = 8$. Later, the algorithm processes the other common hub vertex C. Since $sd(F,H) < d_{CF} + d_{CH} = 3 + 8 = 11$, the shortest distance $sd(F,H)$ is not updated. Finally, the algorithm returns $sd(F,H) = 8$.*

The shortest path is retrieved as follows. Since the common hub vertex A is on the shortest path, the algorithm computes $sp(F,H)$ as $sp(F,A) \cup sp(A,H)$. To retrieve $sp(F,H)$, the algorithm backtracks using the predecessor of the hub label $(A,5,B)$ in $H(F)$. Specifically, the label indicates that B is the first vertex on the shortest path from F to H. Next, the algorithm accesses the hub labels of B, $H(B)$ and identifies the label $(A,2,A)$ containing the hub vertex A. Since the predecessor in $(A,2,A)$ is A itself, the path extraction concludes, returning the shortest path $\langle F,B,A\rangle$ from F to A. Similarly, the shortest path $\langle H,E,A\rangle$ is extracted from H to A. By concatenating these two shortest paths, the algorithm determines the shortest path between F and H as $\langle F,B,A,E,H\rangle$.

III. Related Work

Navigation in road networks has garnered significant attention, with existing research (see Section II-C) largely concentrating on computing the shortest paths by representing edge weights as static values. However, static road networks merely approximate real-world road conditions. To account for factors like road congestion, closures, and others, numerous studies [35], [36] have endeavored to efficiently compute the shortest path on time-dependent road networks, where the travel times on the edges vary throughout the day. Additionally, recent research [37]–[39] has expanded beyond mere shortest and time-dependent paths to consider other important user preferences, aiming to plan utility-rich driving routes such as the safest and most scenic paths. However, these works are orthogonal to finding high-quality alternative paths which is the focus of this work. In this section, we cover some of the most well-known alternative pathfinding techniques.

a) Penalty: This approach [16], [17], [40] iteratively computes the shortest paths from the source s to the target t. After each iteration, it applies a penalty to each edge on the shortest path found in the previous iteration by increasing its edge weight by a certain penalty factor. This approach aims to find multiple paths that are significantly different from each other, as increasing the edge weight on the shortest path found in the previous iteration will likely lead to a different shortest path being chosen in the next iteration. The algorithm stops when either k unique paths have been found or when the length of the shortest path found in the current iteration is longer than $d(s,t)$ $(1+\epsilon)$ where ϵ is a user-defined parameter. Although the penalty method heavily relies on increasing edge weights to avoid similar paths, it offers no guarantee of identifying alternative paths that are sufficiently dissimilar from each other. Despite its ease of implementation, as shown in our experimental study, this method returns, in some cases, alternative paths that are highly similar to each other.

b) Plateaus: This approach [19] first computes two shortest path trees T_s and T_t rooted at s and t, respectively. Then, the two trees are joined and common branches in the two trees are found. Each common branch is called a plateau. Let $pl(u,v)$ be a plateau such that u is the endpoint closer to s and v is the endpoint closer to t. The algorithm selects the k longest plateaus, and for each of these plateaus $pl(u,v)$, computes an alternative path as $sp(s,u)$ $pl(u,v)$ $sp(v,t)$. Note that building the shortest path trees requires running two Dijkstra searches from the source s and target t, respectively, which results in high query processing time. Our proposed algorithm outperforms this approach by more than two orders of magnitude in terms of running time.

c) SVP+: Similar to Plateaus, this approach [23] generates two shortest path trees T_s and T_t. For a vertex v, its *via-path* through v is $sp(s,v)$ $sp(v,t)$. The algorithm iteratively accesses vertices in the two trees T_s and T_t in ascending order of the lengths of their via-paths and adds a via-path to the set of alternative paths only if its similarity to the already added alternative paths is less than θ where θ is a user-defined similarity threshold. Similar to the Plateaus approach, this approach also suffers from high computation time dominated by the construction of two shortest paths trees.

d) ESX: The key idea of this approach [23] is to incrementally remove edges from the road network and compute the shortest path on the updated network. The order in which the edges are removed from the road network affects both the result quality and the performance. The authors apply multiple strategies (e.g., edge removal, optimisation) to improve its performance and result quality. ESX is an extension of the Penalty approach, incorporating a more greedy strategy by removing edges rather than increasing edge weights. However, it still suffers from a high computation time and is more than two orders of magnitude slower than our approach.

e) X-CHV: This idea was first proposed in [24] and is quite similar to SVP+. The fundamental difference is that X-CHV performs a bidirectional search on the contraction hierarchy (CH) [32] and selects the intersecting nodes of CH as via-nodes. By using these via-nodes, alternative paths are generated by concatenating the two shortest paths. While X-CHV enhances SVP+ by adjusting CH to expedite runtime, it still requires extensive search efforts to discover alternative paths. Consequently, its computation time is still up to an order of magnitude slower than our proposed approach which avoids expensive graph search.

f) DkSP: This approach [15] constructs the shortest path tree from the source node s and generates candidate paths that deviate from the shortest path between s and t using edge deviation, ordered by increasing distance. The primary goal of DkSP is to reduce the lengths of alternative paths while maintaining a relatively high level of similarity between them. For instance, in the experimental study conducted in [15], the allowed similarity between alternative paths can be as high as 90%. However, our approach, along with the majority of existing works, emphasizes ensuring significant dissimilarity between paths, with, for example, a limit of up to 50% similarity. As the similarity threshold decreases, DkSP becomes considerably more resource-intensive, reaching up to three orders of magnitude higher computational costs compared to our approach. This increased expense is attributed to the need to explore a significantly larger number of candidate paths within the shortest path tree.

IV. Hub-based Viable Alternative Routing

Our approach, Hub-based Viable Alternative Routing (HubVAR), consists of two phases: (i) generation of candidate paths (Section IV-A); and (ii) selection of up to k alternative paths from the generated candidates (Section IV-B). In this section, we present the basic ideas for each phase and discuss optimizations to improve the performance.

A. Candidate Paths Generation

Given user-defined similarity threshold θ and distance threhsold ϵ, Algorithm 1 shows how to generate candidate paths between s and t. First, it utilizes hub-labeling (HL) to compute the shortest distance $sd(s,t)$ and extract the shortest path $sp(s,t)$ between s and t (line 1). Then, the algorithm initializes P, a set containing the candidate alternative paths found so far, by inserting the shortest path $sp(s,t)$ in it (line 2). To generate the candidate alternative paths, the algorithm

Algorithm 1: Candidate Paths Generation

Input: s: start ; t: target ; θ: similarity threshold; ϵ: distance threshold
Output: P: a set of candidate paths

1 Compute shortest distance $sd(s,t)$ and path $sp(s,t)$;
2 $P = \{sp(s,t)\}$;
3 **foreach** $v \in H(s) \cap H(t)$ **do**
4 **if** v *is not on the shortest path* $sp(s,t)$ **then**
5 $sd(s,v,t) = sd(s,v) + sd(v,t)$;
6 **if** $sd(s,v,t) \leq sd(s,t) \cdot (1+\epsilon)$ **then**
7 $sp(s,v,t) = sp(s,v) \cdot sp(v,t)$;
8 **if** $sp(s,v,t)$ *does not contain cycles* **then**
9 **if** $Sim(\{sp(s,t), sp(s,v,t)\}) \geq \theta$ **then**
10 $P = P \cup sp(s,v,t)$;
11 **return** P

iteratively examines the unique hub vertices in $H(s)$ and $H(t)$ (i.e., $v \in H(s) \cap H(t)$) (line 3). The algorithm prunes a hub vertex v if it lies on the shortest path $sp(s,t)$ (line 4) because the shortest via-path through v, $sp(s,v,t)$, is the same as the shortest path $sp(s,t)$. If v does not lie on the shortest path, our algorithm considers the via-path $sp(s,v,t)$ where $sp(s,v,t)$ is concatenation of the shortest paths from s to v and from v to t (i.e., $sp(s,v,t) = sp(s,v) \cdot sp(v,t)$). Although each $sp(s,v,t)$ can be used as a candidate alternative path and stored in P, our algorithm ignores poor quality paths by filtering $sp(s,v,t)$ if:

1) it is much longer than the shortest path, specifically if $sd(s,v,t) > sd(s,t) \cdot (1+\epsilon)$ (line 5 - 6); or
2) it contains cycles (line 7 - 8); or
3) its similarity to the shortest path $sp(s,t)$ is greater than the similarity threshold θ (line 9).

The pruning steps 1 and 3 ensure that the candidate paths satisfy *similarity* and *distance* thresholds. Pruning step 2 ignores any path that contains cycles because such paths are not realistic and have infinite *bounded stretch* (see Eq. (3)). Each via-path $sp(s,v,t)$ which is not pruned by the above pruning rules is appended to the candidate set P (line 10). When each hub vertex v is processed as described above, the algorithm terminates by returning the candidate set P (line 11).

Example 3. *Consider the example in Fig. 4, where we assume the source is F, the target is H, ϵ = 0.5 and θ = 0.5. The algorithm begins by computing the shortest path $sp(F,H)$=⟨F, B, A, E, H⟩ (shown in orange dotted line) and $sd(F,H)$ = 8 using HL. The shortest path is then appended to P as a candidate path. Next, the algorithm employs HL to determine the union of hub vertices {A, B, C, D, E, F, H} stored in F and H, as shown in Table I. The algorithm then iteratively accesses each hub vertex, skipping the hub vertices A, B, E, F and H since they already appear on the shortest path. The remaining hub vertices C and D are processed. When hub vertex C is processed, the via-path $sp(F,C,H)$=⟨F, C, D, E, H⟩ (shown in magenta dashed line) is appended to P because it is not pruned by any of the pruning rules. Specifically, its length 11 is smaller than $sd(F,H) \cdot (1+\epsilon)$ = 8 · 1.5 = 12, it does not contain a cycle and its similarity to the shortest path is less than 0.5. The algorithm then processes the hub vertex D but it is pruned because the via-path $sp(F,D,H)$=⟨F, B, A, E, D, E, H⟩ contains a cycle (e.g., ⟨E, D, E⟩). The algorithm terminates and returns the two candidate paths shown in orange and magenta.*

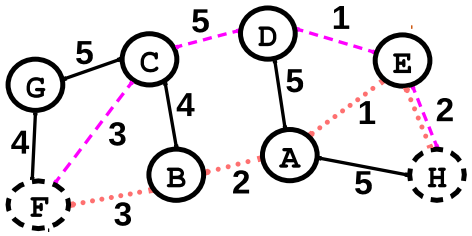

Fig. 4: The source (resp. target) node is F (resp. H). Hub-VAR generates two candidate paths colored in orange and magenta.

Next, we describe how to implement the pruning steps.

1) Implementing Pruning Step 1: For a given $v \in H(s) \cap H(t)$, this pruning step requires calculating $sd(s,v,t)$ which requires computing $sd(s,v)$ and $sd(v,t)$. We efficiently compute $sd(s,v,t)$ using HL as follows. Since $v \in H(s) \cap H(t)$, it must have a hub label in at least one of $H(s)$ or $H(t)$. We have two cases:

Case 1. v is a hub label in both $H(s)$ and $H(t)$. In this case, $sd(s,v)$ and $sd(v,t)$ can be easily retrieved using the hub labels of v in $H(s)$ and $H(t)$, respectively.

Case 2. v is a hub label in only one of $H(s)$ or $H(t)$. Without loss of generality, assume that v is a hub label in $H(s)$ but not in $H(t)$. In this case, $sd(s,v)$ is obtained using the hub label of v in $H(s)$ and $sd(v,t)$ is computing using HL as explained in Section II-C.

Optimisations. For the case 2 above, the algorithm requires computing either $sd(s,v)$ or $sd(v,t)$ using HL. Without loss of generality, assume that $sd(v,t)$ is required to be computed using HL (whereas $sd(s,v)$ is retrieved from the hub label of v in $H(s)$). To further optimise computational efficiency, we delay the distance calculation $sd(v,t)$ as follows. We first check if $sd(s,v) > sd(s,t) \cdot (1+\epsilon)$. If so, $sd(v,t)$ is not required and the path is pruned. Otherwise, we compute a lower bound distance between v and t (denoted as $ld(v,t)$). If $sd(s,v) + ld(v,t) > sd(s,t) \cdot (1+\epsilon)$, the path is pruned and $sd(v,t)$ is not needed to be computed. Although any lower bound distance can be computed, in our implementation, we use landmark based lower bound distance [29].

2) Implementing Pruning Step 2: A simple approach to implement pruning step 2 is to traverse the via-path $sp(s,v,t)$ and see if it contains cycles (i.e., a vertex appears more than once on the path). However, this requires computing and traversing the whole path which may be un-necessary.

Optimisation. Recall that $sp(s,v,t) = sp(s,v) \cdot sp(v,t)$. The following lemma helps in efficiently identifying if $sp(s,v,t)$ contains a cycle or not.

Lemma 2. *Let v_s be the first vertex on the shortest path from v to s. Let v_t be the first vertex on the shortest path from v to t. The path $sp(s,v,t)$ contains a cycle if and only if v_s and v_t refer to the same vertex.*

Proof. If v_s and v_t are the same vertex, then it is easy to see that $sp(s,v,t)$ contains a cycle $\langle v_s, v, v_t \rangle$. Next, we prove that

$sp(s,v,t)$ contains a cycle only if v_s and v_t refer to the same vertex. Assume that $sp(s,v,t)$ contains a cycle but v_s and v_t do not refer to the same vertex. Since there is a cycle, there must be at least one vertex r repeated twice. Since $sp(s,v)$ and $sp(v,t)$ are shortest paths, they cannot contain cycles. Thus, if there is a cycle in $sp(s,v,t)$, it must overlap both $sp(s,v)$ and $sp(v,t)$, i.e., the cycle is $hr, \quad , v, \quad ri$. Since $hr, \quad , vi$ is on $sp(s,v)$ and $hv, \quad , ri$ is on $sp(v,t)$, the cycle is $sp(r,v)$ $sp(v,r)$. Since the graph is undirected[2], all vertices on $sp(r,v)$ and $sp(v,r)$ are repeated[3]. Thus, v_s and v_t must be the same vertex which contradicts our assumption. □

Based on Lemma 2, instead of retrieving the whole via-path, we simply check whether v_s and v_t refer to the same vertex or not. Note that v_s and v_t can be easily retrieved using HL by looking at the predecessor vertex stored in the hub labels (as explained in Section II-C). Also, v_s or v_t may have already been computed during pruning step 1. In short, this optimisation does not require computing the via-path $sp(s,v,t)$ at line 7. Instead, the pruning step 2 is applied by computing only v_s and v_t.

3) Implementing Pruning Step 3: Applying pruning step 3 requires computing similarity between $sp(s,v,t)$ and $sp(s,t)$ (line 9). This can be done by computing $sp(s,v,t) = sp(s,v) \quad sp(v,t)$ and computing its similarity with $sp(s,t)$ using Eq. (1). Next, we present an optimisation that may not require computing the whole path $sp(s,v,t)$ if it can be pruned by the pruning step 3.

Optimisation. First we present the following lemma which is crucial to the optimisation.

Lemma 3. *Let P_i and P_j be two paths. Similarity between them $Sim(fP_i, P_j g)$ is greater than θ if and only if $jP_i \setminus P_j j > \frac{\theta}{1+}(jP_i j + jP_j j)$ where jXj denotes the length of a path X.*

Proof. We show that $Sim(fP_i, P_j g) > \theta$ is equivalent to $jP_i \setminus P_j j > \frac{\theta}{1+}(jP_i j + jP_j j)$. Since $Sim(fP_i, P_j g) = \frac{jP_i \setminus P_j j}{jP_i [P_j j}$ (see Eq. (1)), we have.

$$Sim(fP_i, P_j g) > \theta$$
$$\frac{jP_i \setminus P_j j}{jP_i [P_j j} > \theta$$

Since $jP_i [P_j j$ is equal to $jP_i j + jP_j j \quad jP_i \setminus P_j j$, we get

$$\frac{jP_i \setminus P_j j}{jP_i j + jP_j j \quad jP_i \setminus P_j j} > \theta$$
$$jP_i \setminus P_j j > \theta(jP_i j + jP_j j \quad jP_i \setminus P_j j)$$
$$(1+\theta)(jP_i \setminus P_j j) > \theta(jP_i j + jP_j j)$$
$$jP_i \setminus P_j j > \frac{\theta}{1+\theta}(jP_i j + jP_j j)$$

□

[2]For ease of presentation, in this paper, we assume an undirected graph. But the proof can be extended for the directed graphs as well.

[3]Even if there are more than one shortest paths between r and v, hub labeling construction using a shortest path tree rooted at v ensures that a unique shortest path is selected.

We exploit Lemma 3 as follows. Instead of computing the whole path $sp(s,v,t)$, we incrementally retrieve edges on $sp(s,v,t)$ and calculate $jsp(s,v,t) \setminus sp(s,t)j$. When $sp(s,t)$ is computed at line 1 of Algorithm 1, we mark every edge that lies on $sp(s,t)$. This allows checking whether an edge on $sp(s,v,t)$ overlaps with $sp(s,t)$ in $O(1)$. We incrementally update $jsp(s,v,t) \setminus sp(s,t)j$ as new edges on $sp(s,v,t)$ are retrieved and, at any stage, if it becomes bigger than $\frac{\theta}{1+}(sd(s,v,t) + sd(s,t))$, we prune the path, where $sd(s,v,t)$ and $sd(s,t)$ are already computed by the algorithm earlier. This allows pruning the via-path $sp(s,v,t)$ without retrieving the whole path.

Remark: Algorithm 1 might return fewer than k candidate paths if, for instance, the union of $H(s)$ and $H(t)$ has fewer than k vertices. Our experimental investigation indicates that $H(s) [H(t)$ typically encompasses numerous vertices, making it unlikely to have fewer than k vertices. However, there are cases where this could occur. In such instances, akin to numerous existing methodologies, the algorithm will yield fewer than k alternative paths. As an alternative approach, the algorithm can generate additional candidate paths by progressively considering nodes in $H(v)$ for each v 2 $H(s) [H(t)$ until a sufficient number of candidate paths are obtained.

B. Alternative Paths Selection

After Algorithm 1 generates the candidate paths P, we pass P to another algorithm which selects up to k alternative paths from P and returns to the user. The aim here is to return a result set $R \quad P$ containing up to k paths such that R is of the highest quality according to the metrics defined in Section II. One straightforward approach is to select paths based on a single metric, disregarding others. While this method may expedite the selection process, it often leads to poor path quality. Another strategy is to identify a set of candidate paths that are not dominated by other paths considering all four metrics. However, computing the pareto frontier can yield a large number of paths which may overwhelm the users. To overcome these issues, we introduce a combined objective function that integrates the four path quality metrics in order to select up to k alternative paths from P. The main intuition is to acknowledge that all path quality metrics are important, thereby avoiding the selection of alternative paths that favor only a specific metric.

Specifically, $DR(R)$, $BS(R)$, $Sim(R)$ should be as small as possible and $LO(R)$ should be as large as possible. One possible approach is to consider all possible path combinations in P containing k paths such that $LO(R) \quad BS(R) \quad DR(R) \quad Sim(R)$ is as large as possible. However, this approach requires evaluating $\binom{N}{k}$ path sets which is computationally expensive because $\binom{N}{k}$ may be a large number and computing the quality metrics for each path set is computationally expensive. Next, we present a greedy algorithm which returns R aiming to maximise the objective function $LO(R) \quad BS(R) \quad DR(R) \quad Sim(R)$. Note that this objective function gives equal weight to each quality metric and, depending on the application or user requirements, a weighted aggregate of these values can also be used. Also, we normalise all these values using Min-Max normalization.

Algorithm 2: Get Alternative Paths

Input: P: set of candidate paths; k: # of required paths
Output: Result set R containing up to k alternative paths

```
1 R = {sp(s,t)}; P = P \ sp(s,t)
2 foreach P' ∈ P do
3 |  P'.score = LO(P') · BS(P') · DR(P')
4 while P is not empty and R contains less than k paths do
5 |  foreach P' ∈ P do
6 |  |  P'.quality = P'.score · Sim(P' | R)
7 |  X = the path in P with the highest P.quality
8 |  R = R ∪ X; P = P \ X
9 return R;
```

Algorithm 2 shows the details of the greedy algorithm for alternative path selection. Initially, it inserts the shortest path $sp(s,t)$ in R and removes it from the candidate paths P (line 1). For the remaining paths in P, we compute quality score of each path P, denoted as $P.quality$, and incrementally insert the path P with the highest $P.quality$ in R. The quality score of each path P is $LO(P) \cdot BS(P) \cdot DR(P) \cdot Sim(P \mid R)$. Note that $LO(P)$, $BS(P)$ and $DR(P)$ of a path P remain unchanged in each iteration whereas $Sim(P \mid R)$ needs to be recomputed as more paths are inserted in R. Therefore, we first compute $LO(P) \cdot BS(P) \cdot DR(P)$ for each path, denoted as $P.score$ (see lines 2-3). Then, in each iteration, we compute $Sim(P \mid R)$ and subtract this value from $P.score$ to obtain its quality score $P.quality$ (lines 5-6). Then, the path with the highest quality score is inserted in R and removed from the candidate paths P (lines 7-8). When P becomes empty or R contains at least k paths, the algorithm terminates by returning R (lines 4 and 9).

The algorithm requires computing $DR(P)$, $BS(P)$, $LO(P)$ and $Sim(P \mid R)$. Computing $DR(P)$ is computationally cheap as it needs $|P|$ and $sd(s,t)$ which are already known to the algorithm at this stage. Next, we present two optimizations to improve the computational cost for the other quality metrics.

1) Efficiently computing $BS(P)$ and $LO(P)$: Recall that bounded stretch $BS(P)$ of P is the maximum stretch of any of its subpaths $P(v_x, v_y) \in P$ where stretch of a subpath $P(v_x, v_y)$ is $d^P(v_x, v_y)/sd(v_x, v_y)$ (see Eq. (3)). Thus, one simple approach to compute $BS(P)$ is to compute stretch of each subpath $P(v_x, v_y)$ and maintain the maximum stretch. Also, recall that $LO(P)$ is computed by dividing the length of the shortest suboptimal path $L(P)$ by $sd(s,t)$ (see Eq. (4)). $L(P)$ can be computed by considering all subpaths $P(v_x, v_y)$ and maintaining the smallest length for paths that are not optimal (i.e., $d^P(v_x, v_y) > sd(v_x, v_y)$). Let n be the number of vertices on the path P, the above approaches to compute $BS(P)$ and $LO(P)$ require considering $O(n^2)$ subpaths. The following lemma helps us significantly reduce the number of subpaths considered for computing $BS(P)$ and $LO(P)$

Lemma 4. *Consider a path P and its subpath $P' \in P$. If P' is an optimal path, each subpath $P(v_x, v_y)$ where both v_x and v_y lie on P' can be safely ignored to correctly compute $BS(P)$ and $LO(P)$.*

Proof. We show that the stretch of each such $P(v_x, v_y)$ is 1 and thus is not required to be specifically computed. Since P' is an optimal path, for every (v_x, v_y) on P', $P(v_x, v_y)$ is also an optimal path, i.e., $d^P(v_x, v_y) = sd(v_x, v_y)$. Therefore, its stretch $d^P(v_x, v_y)/sd(v_x, v_y) = 1$. Computing $LO(P)$ requires computing the length of the shortest suboptimal subpath. Since P' is an optimal path, $P(v_x, v_y)$ is also an optimal path and thus is not required to be considered. □

Now, we present Algorithm 3 which efficiently computes $BS(P)$ and $LO(P)$ for a via-path $sp(s,v,t)$ by exploiting Lemma 4. The algorithm initializes the bounded stretch (BS) to be 1 which is the minimum possible bounded stretch, and the length of the shortest suboptimal path L to ∞. Since $sp(s,v,t) = sp(s,v) \cdot sp(v,t)$, we do not need to consider any pair of vertices (v_x, v_y) where both v_x and v_y lie either on $sp(s,v)$ or on $sp(v,t)$ (see Lemma 4). Therefore, the algorithm only considers the vertices (v_x, v_y) where v_x lies on $sp(s,v)$ and v_y lies on $sp(v,t)$. Specifically, the algorithm employs two nested loops where the outer loop incrementally considers each v_x on the shortest path from s to v (line 1) and the inner loop incrementally processes each v_y on the shortest path from t to v (line 2). In each iteration of the inner loop, the algorithm calculates the shortest distance $sd(v_x, v_y)$ using HL. Note that $d^P(v_x, v_y)$ can be obtained in $O(1)$ if the path distance $d^P(s, v')$ from s to each vertex v' on the path $sp(s,v,t)$ is recorded at the beginning of the algorithm by traversing the path $sp(s,v,t)$. Specifically, $d^P(v_x, v_y) = d^P(s, v_y) - d^P(s, v_x)$.

If the subpath $P(v_x, v_y)$ is not a shortest path (i.e., $d^P(v_x, v_y) \neq sd(v_x, v_y)$), the algorithm computes stretch of $P(v_x, v_y)$ and updates BS if this stretch is higher than the current BS (line 4). Also, L is also updated if the length of this path $d^P(v_x, v_y)$ is smaller than the current L (line 5). If the subpath $P(v_x, v_y)$ is a shortest path, the algorithm exploits Lemma 4 and breaks the inner loop (line 6). When both nested loops are concluded, the algorithm returns the bounded stretch BS and local optimality which is $L/sd(s,t)$ (line 7).

While Algorithm 3 correctly computes $BS(P)$ and $LO(P)$, the algorithm to return top-k alternative paths (Algorithm 2) does not necessarily need exact $BS(P)$ and $LO(P)$, i.e., approximate values of $BS(P)$ and $LO(P)$ may be sufficient to return high quality paths. Therefore, it is not necessary to traverse every vertex v_x and v_y in the two nested loops of Algorithm 3. Instead, we can skip x nodes in each iteration of inner and outer loops. We tried different values of x and observed that, for $x = 10$, Algorithm 3 still returned very good approximates of $BS(P)$ and $LO(P)$ while significantly reducing the computation time.

2) Memoization: Algorithm 2 requires computing $BS(P)$ and $LO(P)$ for all candidate paths $P \in P$ and, we observed that, there were repetitive calculations of the shortest distance $sd(v_x, v_y)$ for numerous duplicate pairs (v_x, v_y) across different candidate paths. To further improve the computational efficiency, we implemented a hash table to store the values of $sd(v_x, v_y)$ for each vertex pair (v_x, v_y). This approach avoids redundant computations of the shortest distances using HL when the same vertex pair has already been cached. Additionally, Algorithm 2 requires computing similarity $Sim(P \mid R)$

Algorithm 3: Optimized calculation of BS and LO

Input: P: a via-path $sp(s, v, t)$
Output: path quality metrics: $BS(P)$ and $LO(P)$
Initialisation: $BS = 1$; $L = 1$
foreach v_x *on the shortest path* ***from*** s ***to*** v **do**
  foreach v_y *on the shortest path* ***from*** t ***to*** v **do**
    if $d^P(v_x, v_y) \leq sd(v_x, v_y)$ **then**
      $BS = max(BS, \frac{d^P(v_x, v_y)}{sd(v_x, v_y)})$;
      $L = min(L, d^P(v_x, v_y))$;
    else break;
return BS and $L/sd(s, t)$;

for each candidate path $P \in R$ whenever a new alternative path is appended to the result set R. The similarity metric $Sim(P, R)$, as defined in Eq. (1), represents the maximum similarity between any pair of paths (P_i, P_j) in $P \cup R$. To avoid redundant computations of the similarity between pairs (P_i, P_j), we also employ caching to store the computed similarity values for each pair of paths (P_i, P_j). Similarly, we also integrate these caching strategies into Algorithm 3. When implementing caching, it is important to note that the hash table created for each individual query can be deleted once the query results have been computed. Consequently, the size of the hash table does not accumulate, leading to negligible memory overhead as verified in the experiments.

V. Experiments

A. Settings

We run experiments on an Apple M1 Pro machine with 10-core CPU with 32 GB of RAM. All algorithms have been implemented in C++ and compiled with the -O3 flag. To ensure a fair comparison, the algorithms use the same implementation of common routines. We obtained real-world road networks from OpenStreetMap[4] for three demographically diverse cities: Melbourne, Dhaka, and Copenhagen (see Table II). For each road network, we show results for two sets of experiments: one where the edge weights correspond to the travel times; and the other where edge weights correspond to the travel distances. As noted in the previous works [27], if an approach returns less than k alternative paths, it may unfairly receive better quantitative scores for certain quality metrics. Therefore, we generated 10,000 queries for each map with source and target for each query uniformly distributed over the map, ensuring that all approaches returned k alternative paths.

B. Algorithms Evaluated

Our approach, Hub-based Viable Alternative Routing, is shown as **Hub-VAR** in the experiments. To implement the hub labeling, we employ the state-of-the-art algorithm, Significant path-based Hub Pushing (SHP) [5], using the implementation taken from the public repository[5]. For reproducibility, implementation of Hub-VAR will be made available online[6].

[4]www.openstreetmap.org
[5]http://degroup.cis.umac.mo/sspexp
[6]http://will_publish_after_acceptance

We compare our approach with the six well-known and state-of-the-art algorithms discussed in Section III. To ensure a fair comparison, we strive to utilize the original source code released by the creators of each algorithm, whenever accessible. Therefore, for **SVP+** and **ESX** [23], we utilize the original authors' implementations from the publicly available repository[7]. Similarly, we use the publicly available implementation of **DkSP** [15] from the online repository[8]. However, for Penalty and Plateaus, the algorithm inventors have not released their implementations. Therefore, we opted for the implementations provided by the authors of a recent comparative study [11]. Notably, the X-CHV implementation was not available, prompting us to develop it ourselves. Note that implementing X-CHV involves creating contraction hierarchies (CH), for which we utilized the CH implementation from RoutingKit[9], implemented by the same research group that proposed X-CHV. We make our best efforts to implement the algorithm following the details provided by the authors in the original work. Similar to most of the existing studies [11], [22], [23], [25], all algorithms are configured to compute $k = 3$ alternative paths. The similarity threshold θ and the distance threshold ϵ are both set to 0.5, unless specified otherwise.

C. Results

1) Preprocessing: In Table II, we present the build time and memory costs associated with hub labels. In all cases, the build time is under two minutes and the memory required to store hub labels is under 2GB, which is reasonable for modern systems that have much larger main memories. Similar to existing works, these labels are generated offline and persist in memory throughout query processing. Additionally, we offer insights into the maximum and average label sizes for the vertices in each network. On average, the vertices have a sufficient number of hub labels for Algorithm 1 to identify candidate paths. According to the Table II, the hub labeling technique performs better for the travel time network. The label size in the distance network is higher, thereby incurring higher query runtime, as demonstrated in later sections.

2) Query Runtimes: Figure 5 illustrates the runtime for different algorithms on the three road networks using either the travel time or the distance. Similar to existing studies for shortest path queries on road networks [4], we sort the queries based on the shortest distances between their respective start and target locations. The x-axis of each figure represents the percentile ranks of the queries in this sorted order (e.g., a query with a larger percentile has a longer distance between its start and target). Note that the y-axis is in log scale. As expected, the cost of each algorithm increases as the distances between query start and target location increases. Our approach, Hub-VAR, consistently outperforms all competitors for all road networks and for both travel time and distance. Specifically, Hub-VAR is 4-8 times faster than X-CHV and is 2-3 orders of magnitude faster than all other approaches. Notably, Hub-VAR exhibits faster performance on travel time maps as

[7]https://github.com/tchond/kspwlo
[8]https://github.com/AngelZihan/Diversified-Top-k-Route-Planning-in-Road-Network
[9]https://github.com/RoutingKit/RoutingKit

City Name	#V	#E	Travel Time				Distance			
			Build Time (Mins)	Memory (MB)	Node Label Size		Build Time (Mins)	Memory (MB)	Node Label Size	
					MAX	AVG			MAX	AVG
Melbourne	907k	1978k	0.5394	974	252	108.5	1.244	1563	350	151.9
Dhaka	484k	1011k	0.2182	348	325	85.6	0.7198	836	362	104.7
Copenhagen	258k	556k	0.0919	240	207	78.1	0.5693	593	287	95.4

TABLE II: Number of vertices ($\#V$) and edges ($\#E$) in each map. We also show preprocessing cost for constructing the hub labeling, including build time in minutes, memory cost in megabytes, and average/maximum label size.

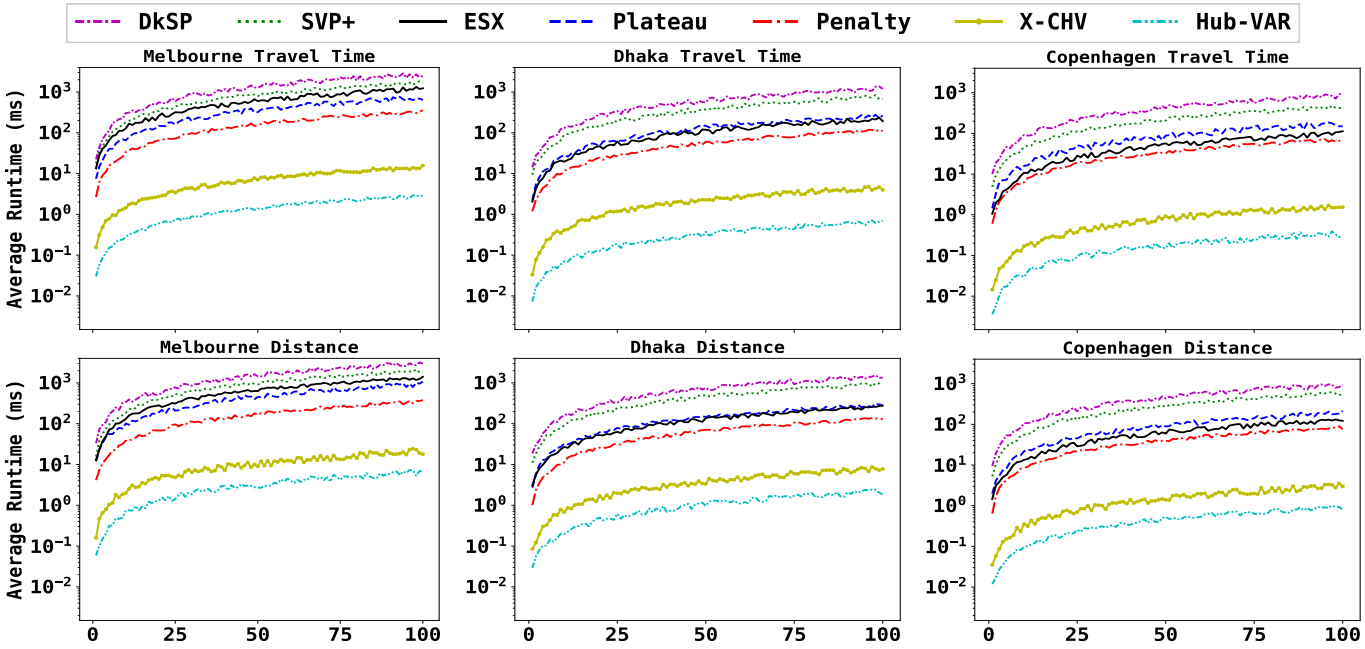

Fig. 5: Runtime comparisons on the three road networks. The x-axis shows the percentile ranks of queries sorted based on the shortest distances between start and target.

opposed to distance maps. This distinction can be attributed to the underlying hub labeling technique, SHP, which has been proven to operate more efficiently on travel time maps in previous studies on the shortest path queries [4].

3) Path Quality: Table III compares the quality of the alternative paths for Hub-VAR against the competitors. Our evaluation is based on the alternative paths metric defined in Section II: (i) Bound Stretch $BS(P)$; (ii) Similarity $Sim(P)$; (iii) Local Optimality $LO(P)$; and (iv) Distance Ratio $DR(P)$. For each metric, we present both the average value as well as the worst reported value across all queries. We remark that although our algorithm, Hub-VAR, computes approximate values of $LO(P)$ and $BS(P)$ to optimise the query performance (see Section IV-B), the quality metrics reported in Table III correspond to the exact values for all metrics.

The results show that no single approach dominates all other algorithms across all metrics. The Plateau method selects alternative paths with longer plateaus which are locally optimal. Consequently, it performs better in terms of $LO(P)$ but exhibits lower quality in $DR(P)$ due to longer paths. The Penalty approach adjusts the penalty factor for each newly selected alternative path, making it more likely to choose dissimilar paths and thus performing better in terms of $Sim(P)$. However, this comes at the cost of other metrics as the Penalty method lacks guarantees in those areas. While most of the algorithms are pretty similar in terms of path quality metrics, DkSP is an outlier. This is mainly because, as noted in Section III, DkSP aims to minimize the lengths of alternative paths and allows very high similarity between the alternative paths. As a consequence, it achieves better performance in terms of $DR(P)$ and $BS(P)$ whereas it performs very poorly on $Sim(P)$ and $LO(P)$. Specifically, the paths reported by DkSP are around 75% similar to each other on average (the maximum similarity is 98%) and the local optimality is also very small. This is undesirable in most real world applications where the users are interested in retrieving paths that are significantly different from each other, and do not have small detours (i.e., do not have very low local optimality).

Hub-VAR considers a combination of quality metrics during alternative path computation. By doing so, it ensures that it does not perform poorly in any particular criterion, making it consistently competitive across all metrics. At the same time, its runtime is 2-3 orders of magnitude lower than most of the approaches making it suitable for deployment in real-world navigation systems that are expected to handle tens of thousands of queries per second. X-CHV is the closest to Hub-VAR

Algorithm	Travel Time									Distance								
	$BS(P)$#		$Sim(P)$#		$LO(P)$"		$DR(P)$#		Runtime	$BS(P)$#		$Sim(P)$#		$LO(P)$"		$DR(P)$#		Runtime
	AVG	**MAX**	**AVG**	**MAX**	**AVG**	**MIN**	**AVG**	**MAX**	(in ms)#	**AVG**	**MAX**	**AVG**	**MAX**	**AVG**	**MIN**	**AVG**	**MAX**	(in ms)#
Melbourne																		
Plateau	1.29	2.28	0.39	0.86	**0.36**	**0.008**	0.19	0.48	352.7	1.24	2.24	0.37	0.84	0.31	**0.005**	0.15	0.49	427.3
Penalty	1.33	2.16	**0.29**	0.95	0.15	0.004	0.15	0.48	153.6	1.26	3.04	**0.27**	0.89	0.36	**0.005**	0.14	0.50	161.4
SVP+	1.39	3.63	0.36	0.50	0.21	0.004	0.07	0.28	851.2	1.15	2.32	0.28	0.50	0.38	**0.005**	0.06	0.34	993.5
ESX	1.32	3.33	0.38	0.50	0.28	0.004	0.10	0.32	515.7	1.17	2.98	0.29	0.50	**0.39**	**0.005**	0.08	0.35	621.1
DkSP	**1.08**	**2.07**	0.77	0.97	0.03	0.001	**0.003**	**0.04**	1350.9	**1.07**	**2.02**	0.75	0.96	0.01	0.001	**0.002**	**0.03**	1546.3
X-CHV	1.42	4.02	0.42	**0.49**	0.21	0.004	0.10	0.37	5.2	1.26	3.54	0.35	0.49	0.29	**0.005**	0.08	0.38	8.6
Hub-VAR	1.34	4.07	**0.29**	**0.49**	0.25	0.004	0.13	0.43	**0.8**	1.24	3.16	**0.27**	**0.48**	0.30	**0.005**	0.12	0.38	**1.8**
Dhaka																		
Plateau	1.44	2.54	**0.32**	0.80	**0.35**	**0.007**	0.20	0.46	116.8	1.36	3.63	0.40	0.87	0.28	**0.006**	0.15	0.47	126.4
Penalty	1.48	2.85	0.34	0.94	0.18	0.003	0.16	0.50	52.5	1.38	3.73	0.33	0.93	0.35	**0.006**	0.10	0.49	61.4
SVP+	2.64	7.13	0.35	**0.50**	0.25	0.004	0.09	0.38	578.4	1.38	4.09	**0.28**	0.50	0.33	**0.006**	0.06	0.33	672.3
ESX	1.76	3.73	0.36	**0.50**	0.27	0.004	0.11	0.44	109.7	1.31	3.83	**0.28**	0.50	**0.37**	**0.006**	0.07	0.39	127.6
DkSP	**1.11**	**2.24**	0.81	0.97	0.01	0.001	**0.005**	**0.05**	933.4	**1.08**	**2.16**	0.74	0.95	0.02	0.001	**0.001**	**0.04**	1021.6
X-CHV	2.31	6.85	0.39	**0.50**	0.27	0.003	0.11	0.41	1.6	1.41	4.81	0.37	0.50	0.28	**0.006**	0.08	0.38	2.9
Hub-VAR	1.73	5.93	**0.32**	**0.50**	0.28	0.004	0.15	0.45	**0.2**	1.39	4.52	0.30	**0.49**	0.29	**0.006**	0.10	0.41	**0.7**
Copenhagen																		
Plateau	1.31	2.30	0.38	0.77	**0.38**	**0.009**	0.16	0.48	81.2	1.32	3.53	0.35	0.88	0.27	**0.008**	0.12	0.47	88.9
Penalty	1.33	2.50	**0.33**	0.86	0.11	0.005	0.12	0.48	35.8	1.32	3.58	0.39	0.90	0.31	**0.008**	0.10	0.49	40.3
SVP+	1.52	3.76	0.35	0.50	0.30	0.005	0.07	0.30	218.2	1.33	3.49	**0.30**	0.50	0.33	**0.008**	0.08	0.40	287.9
ESX	1.43	3.63	0.37	0.50	0.31	0.005	0.07	0.35	47.6	1.33	3.69	**0.30**	0.50	**0.35**	**0.008**	0.09	0.42	61.1
DkSP	**1.09**	**2.03**	0.78	0.98	0.02	0.001	**0.002**	**0.03**	427.6	**1.07**	**1.97**	0.76	0.96	0.03	0.004	**0.002**	**0.04**	464.4
X-CHV	1.57	3.84	0.41	0.50	0.24	0.005	0.08	0.38	0.7	1.39	4.92	0.38	0.50	0.28	**0.008**	0.09	0.40	1.3
Hub-VAR	1.40	3.77	0.36	**0.49**	0.28	0.005	0.09	0.43	**0.1**	1.34	4.16	0.32	**0.49**	0.29	**0.008**	0.11	0.48	**0.3**

TABLE III: Path quality and computational efficiency comparison. Arrows indicate whether smaller or larger values are better. We show $BS(P)$ (smaller the better), $Sim(P)$ (smaller the better), $LO(P)$ (larger the better), $DR(P)$ (smaller the better), and average runtime (smaller the better). Best values for each column are shown in bold.

in terms of runtime but is still 4-8 times slower. Furthermore, Hub-VAR consistently performs better or comparable to X-CHV in terms of almost all path quality metrics.

4) Other experiments: Fig. 6 shows the impact of varying different important parameters on runtime of different algorithms. In each experiment, only one parameter is varied while the others are set to their default values. As expected, the runtime of most of the algorithms increase as the values of k and ϵ increase because the search space increases. However, Hub-VAR consistently outperforms all existing algorithms. As θ increases the cost of Hub-VAR and X-CHV increases because there are more candidate paths that need to be processed. The cost of DkSP significantly decreases as θ is increased because it needs to explore fewer candidate paths from the shortest path tree (see Section III). For very high values of θ (e.g., > 0.8), DkSP outperforms all algorithms in terms of running time. However, note that DkSP is orthogonal to our algorithm and the other competitors as it is designed to minimise lengths of alternative paths while allowing very high similarity and very low local optimality. In real-world navigation scenarios where diverse alternative paths are recommended, DkSP is not suitable due to the high similarity among the returned alternative paths, reaching up to 0.98 in Table III. DkSP performs very poorly in terms of $Sim(P)$ and $LO(P)$ metrics, e.g., for Melbourne Distance graph (Table III), its average $Sim(P)$ and $LO(P)$ are 0.75 and 0.01, respectively, compared to 0.27 and 0.3, respectively, for Hub-VAR. Thus, DkSP should only be used in applications where length of the paths is the key focus regardless of their similarity and local optimality.

The results for path quality metrics for varying these parameters are not shown due to the space limitations. However, the results follow the same trend as shown in Table III, e.g., most algorithms are comparable to each other except DkSP which has significantly better $BS(P)$ and $DR(P)$ but very poor $Sim(P)$ and $LO(P)$. We also assess the memory consumption of the Memoization technique. As mentioned earlier, the size of the hash table incurs negligible memory overhead considering large main memory available in the modern systems. In the Melbourne Travel Time network experiment, the average size of the hash table is 0.7988 MB (max: 2.8229 MB), and the use of the hash tables results in an average performance gain of 2.81 times (max: 5.23 times).

VI. Conclusion

Our study is the first to exploit hub labeling for alternative routing in road networks. We propose an efficient approach that generates candidate paths using hub labeling and returns high-quality candidate paths by ranking these candidate paths

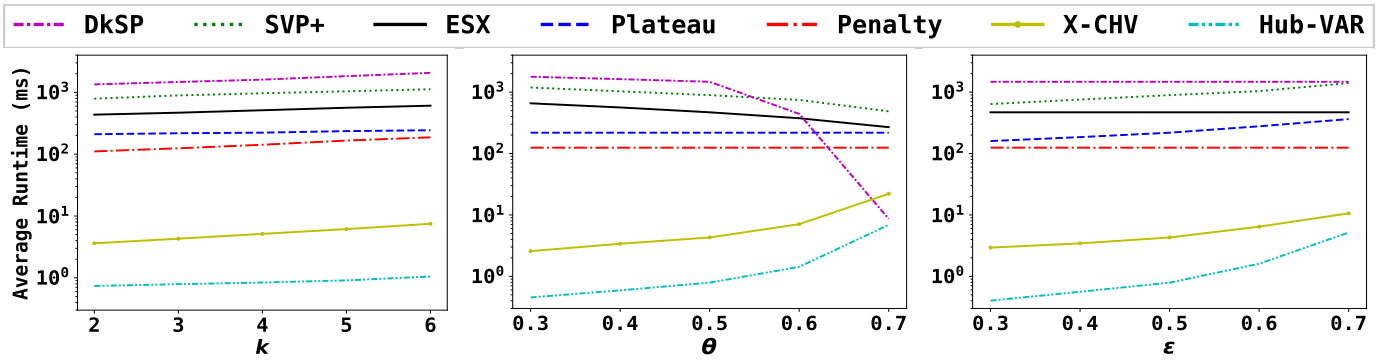

Fig. 6: Runtime comparisons on the Melbourne (travel time) road network for different values of k, θ, and ϵ.

considering a combination of path quality metrics. We present several non-trivial optimisation to improve the efficiency. Our experimental results demonstrate that our proposed approach is up to three orders of magnitude faster than the existing alternative routing approaches and 4-8 times faster than the existing most efficient algorithm. Furthermore, the results show that none of the existing techniques outperforms all other approaches across all performance criteria, while our algorithm is capable of generating high-quality paths of comparable quality at a much lower computational cost and, therefore, is suitable for deployment in large-scale navigation systems.

Acknowledgements

Muhammad Aamir Cheema is supported by the Australian Research Council DP230100081 and FT180100140.

References

[1] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[2] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[3] A. V. Goldberg, H. Kaplan, and R. F. Werneck, "Reach for A*: Efficient point-to-point shortest path algorithms," in *ALENEX*, 2006, pp. 129–143.

[4] B. Shen, M. A. Cheema, D. D. Harabor, and P. J. Stuckey, "Contracting and compressing shortest path databases," in *ICAPS*, 2021, pp. 322–330.

[5] Y. Li, L. H. U, M. L. Yiu, and N. M. Kou, "An experimental study on hub labeling based shortest path algorithms," *Proceedings of the VLDB Endowment*, vol. 11, no. 4, pp. 445–457, 2017.

[6] M. A. Cheema, "Indoor location-based services: challenges and opportunities," *SIGSPATIAL Special*, vol. 10, no. 2, pp. 10–17, 2018.

[7] J. Du, B. Shen, and M. A. Cheema, "Ultrafast euclidean shortest path computation using hub labeling," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 10, pp. 12 417–12 426, Jun. 2023.

[8] F. Barth and S. Funke, "Alternative routes for next generation traffic shaping," in *International Workshop on Computational Transportation Science*, 2019, pp. 6:1–6:8.

[9] A. Botea, E. Nikolova, and M. Berlingerio, "Multi-modal journey planning in the presence of uncertainty," in *ICAPS*, 2013.

[10] F. Islam, C. Paxton, C. Eppner, B. Peele, M. Likhachev, and D. Fox, "Alternative paths planner (APP) for provably fixed-time manipulation planning in semi-structured environments," in *IEEE International Conference on Robotics and Automation, ICRA 2021, Xi'an, China, May 30 - June 5, 2021*. IEEE, 2021, pp. 6534–6540.

[11] L. Li, M. A. Cheema, H. Lu, M. E. Ali, and A. N. Toosi, "Comparing alternative route planning techniques: A comparative user study on melbourne, dhaka and copenhagen road networks," *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[12] L. Li, M. A. Cheema, M. E. Ali, H. Lu, and D. Taniar, "Continuously monitoring alternative shortest paths on road networks," in *Proceedings of the VLDB Endowment*, 2020, pp. 2243–2255.

[13] C. Häcker, P. Bouros, T. Chondrogiannis, and E. Althaus, "Most diverse near-shortest paths," in *Proceedings of the 29th International Conference on Advances in Geographic Information Systems*, 2021, pp. 229–239.

[14] A. Moghanni, M. Pascoal, and M. T. Godinho, "Finding shortest and dissimilar paths," *International Transactions in Operational Research*, vol. 29, no. 3, pp. 1573–1601, 2022.

[15] Z. Luo, L. Li, M. Zhang, W. Hua, Y. Xu, and X. Zhou, "Diversified top-k route planning in road network," *Proceedings of the VLDB Endowment*, vol. 15, no. 11, pp. 3199–3212, 2022.

[16] V. Akgün, E. Erkut, and R. Batta, "On finding dissimilar paths," *European Journal of Operational Research*, 2000.

[17] Y. Chen, M. G. Bell, and K. Bogenberger, "Reliable pretrip multipath planning and dynamic adaptation for a centralized road navigation system," *IEEE Transactions on Intelligent Transportation Systems*, vol. 8, no. 1, pp. 14–20, 2007.

[18] M. Kobitzsch, "An alternative approach to alternative routes: Hidar," in *European Symposium on Algorithms*. Springer, 2013, pp. 613–624.

[19] A. H. Jones, "Method of and apparatus for generating routes," Aug. 21 2012, uS Patent 8,249,810.

[20] Y.-J. Jeong, T. J. Kim, C.-H. Park, and D.-K. Kim, "A dissimilar alternative paths-search algorithm for navigation services: A heuristic approach," *KSCE Journal of Civil Engineering*, vol. 14, pp. 41–49, 2010.

[21] H. Liu, C. Jin, B. Yang, and A. Zhou, "Finding top-k shortest paths with diversity," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 3, pp. 488–502, 2017.

[22] T. Chondrogiannis, P. Bouros, J. Gamper, U. Leser, and D. B. Blumenthal, "Finding k-dissimilar paths with minimum collective length," in *SIGSPATIAL*, 2018, pp. 404–407.

[23] T. Chondrogiannis, P. Bouros, J. Gamper, U. Leser, and D. B. Blumenthal, "Finding k-shortest paths with limited overlap," *The VLDB Journal*, vol. 29, no. 5, pp. 1023–1047, 2020.

[24] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck, "Alternative routes in road networks," *Journal of Experimental Algorithmics (JEA)*, vol. 18, pp. 1–1, 2013.

[25] R. Bader, J. Dees, R. Geisberger, and P. Sanders, "Alternative route graphs in road networks," in *Theory and Practice of Algorithms in (Computer) Systems: First International ICST Conference, TAPAS 2011, Rome, Italy, April 18-20, 2011. Proceedings*. Springer, 2011, pp. 21–32.

[26] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck, "A hub-based labeling algorithm for shortest paths in road networks," in *International Symposium on Experimental Algorithms*, 2011.

[27] L. Li, M. A. Cheema, M. E. Ali, H. Lu, and H. Li, "Efficiently computing alternative paths in game maps," *World Wide Web (WWW)*, 2023.

[28] I. Pohl, "Bi-directional and heuristic search in path problems," Ph.D. dissertation, Stanford Linear Accelerator Center, USA, 1969.

[29] A. V. Goldberg and C. Harrelson, "Computing the shortest path: A* search meets graph theory," in *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005*, 2005, pp. 156–165.

[30] M. Hilger, E. Köhler, R. H. Möhring, and H. Schilling, "Fast point-to-point shortest path computations with arc-flags," in *The Shortest Path Problem, Proceedings of a DIMACS Workshop*, vol. 74, 2006, pp. 41–72.

[31] U. Lauther, "An experimental evaluation of point-to-point shortest path calculation on road networks with precalculated edge-flags," in *The Shortest Path Problem, Proceedings of a DIMACS Workshop*, vol. 74, 2006, pp. 19–39.