Finding k-Dissimilar Paths with Minimum Collective Length

Theodoros Chondrogiannis
University of Konstanz
theodoros.chondrogiannis@uni.kn

Panagiotis Bouros
Johannes Gutenberg University Mainz
bouros@uni-mainz.de

Johann Gamper
Free University of Bozen-Bolzano
gamper@inf.unibz.it

Ulf Leser
Humboldt-Universität zu Berlin
leser@informatik.hu-berlin.de

David B. Blumenthal
Free University of Bozen-Bolzano
david.blumenthal@inf.unibz.it

ABSTRACT

Shortest path computation is a fundamental problem in road networks. However, in many real-world scenarios, determining solely the shortest path is not enough. In this paper, we study the problem of finding k-Dissimilar Paths with Minimum Collective Length (kDPwML), which aims at computing a set of paths from a source s to a target t such that all paths are pairwise dissimilar by at least θ and the sum of the path lengths is minimal. We introduce an exact algorithm for the kDPwML problem, which iterates over all possible $s-t$ paths while employing two pruning techniques to reduce the prohibitively expensive computational cost. To achieve scalability, we also define the much smaller set of the simple single-via paths, and we adapt two algorithms for kDPwML queries to iterate over this set. Our experimental analysis on real road networks shows that iterating over all $s-t$ paths is impractical, while iterating over the set of simple single-via paths can lead to scalable solutions with only a small trade-off in the quality of the results.

CCS CONCEPTS

• **Information systems** → **Geographic information systems**;
• **Mathematics of computing** → Graph algorithms;

KEYWORDS

Alternative Routing, Route Planning, Path Similarity

ACM Reference Format:
Theodoros Chondrogiannis, Panagiotis Bouros, Johann Gamper, Ulf Leser, and David B. Blumenthal. 2018. Finding k-Dissimilar Paths with Minimum Collective Length. In *26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL '18), November 6–9, 2018, Seattle, WA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3274895.3274903

1 INTRODUCTION

Computing the shortest path between two locations in a road network is a fundamental problem that has attracted the attention of both the research community and the industry. In many real-world scenarios though, determining solely the shortest path is not enough. Most commercial route planning applications recommend paths that might be longer than the shortest path, but have other desirable properties, e.g., less traffic congestion. However, the recommended paths also need to be dissimilar to each other to be valued as true alternatives by users. Towards this end, various approaches aim at computing short yet dissimilar alternative paths.

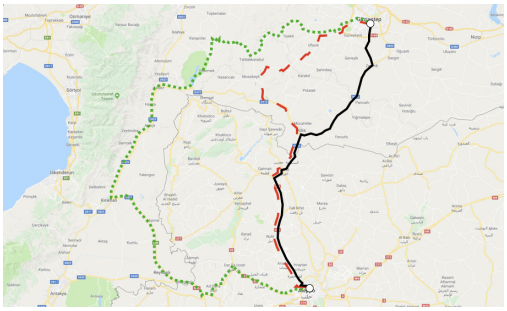


Figure 1: Motivational example.

Related work. The kSPwLO problem introduced by Chondrogiannis at al. [4, 5], aims at computing dissimilar paths while minimizing the length of each subsequent result. In practice, the FindKSPD algorithm of Liu et al. also computes the solution to the kSPwLO [4]. A similar approach proposed by Jeong et al. [8], laxly based on Yen's algorithm [12], computes paths in an iterative fashion and aims at minimizing the similarity of each subsequent result till a path is found that satisfies a user-defined similarity threshold. Akgun et al. [3] present an algorithm which computes alternatives by repeatedly running Dijkstra's algorithm on the road network while imposing a penalty on edges that lies on some already recommended path before each iteration. Last, there exist methods that aim at computing alternatives to the shortest path such as the *Plateaux* method [1] which computes paths that computing paths that cross different highways, the *alternative graphs* which have a similar functionality as the plateaus, and a routing method proposed by Abraham et al. [2] that employs *single-via paths*. In contrast to our work though, none of the aforementioned methods guarantee that the result paths will be dissimilar to each other.

Motivation. Consider the scenario of transportation of humanitarian aid goods through unsafe regions. The distribution of the load to several vehicles that follow different routes can increase the chances that at least some of the goods will be delivered. The total distance covered by vehicles must also be taken into account to minimize the overall cost. For example, Figure 1 shows three different paths from the city of Gaziantep in Turkey to the city of

SIGSPATIAL '18, November 6–9, 2018, Seattle, WA, USA

ACM ISBN 978-1-4503-5889-7/18/11.
https://doi.org/10.1145/3274895.3274903

Aleppo in Syria. The solid/black line indicates the shortest path, the dashed/red line the next path in length order, and the dotted/green line a path that is clearly longer, but also significantly different from the other two. Choosing the black and the red paths is not the best option, since the two paths share a large stretch. Among the other two options, the black-green pair has the minimum collective length and, hence, it is a better option than the red-green pair.

The aforementioned scenario is formally captured by the *k-Dissimilar Paths with Minimum Collective Length* (*k*DPwML) problem. Given two locations s and t on a road network, a *k*DPwML query computes a set of k paths from s to t, such that: (1) all paths in the result set are sufficiently dissimilar to each other (w.r.t a user-defined similarity threshold), and (2) the set exhibits the lowest collective path length among all sets of k sufficiently dissimilar paths. *k*DPwML was originally introduced by Liu et al. as the *Top-k Shortest Paths with Diversity* (Top-KSPD) [11], together with a greedy heuristic method that builds on the K-shortest paths [12].

Contributions. In this paper, we present an in-depth analysis of the *k*DPwML problem. First, we conduct a theoretical analysis to prove that *k*DPwML is strongly *NP*-hard. Second, we investigate the exact computation of *k*DPwML queries, which was not covered by Liu et al. [11]. We present an algorithm that, similar to the approach of Liu et al., builds on the computation of the K-shortest paths [12], along with a pair of pruning techniques. Since such approaches require a prohibitively high number K of paths to be examined, we introduce the much smaller set of simple single-via paths, which extends the concept of single-via paths [2]. Then, we present two algorithms that iterate over this set of paths to compute *k*DPwML queries. Our experiments show that algorithms which iterate over all possible s–t paths cannot scale. Instead, iterating over the set of simple single-via paths can lead to scalable solutions with a very small trade-off in the quality of the results. Last, an extended version [6] of this paper is available that contains our full theoretical analysis, detailed description of our algorithms and additional experiments.

2 NOTATION AND PROBLEM DEFINITION

Let $G = (N, E)$ be a *directed weighted* graph representing a road network with a set of nodes N and a set of edges $E \subseteq N \times N$. [1] Each edge $(n_i, n_j) \in E$ is assigned a *positive* weight $w(n_i, n_j)$, which captures the cost of moving from node n_i to node n_j. A (simple) *path* $p(s{\rightarrow}t)$ from a source node s to a target node t is a *connected* and *cycle-free* sequence of edges $\langle (s, n_i), \ldots, (n_j, t)\rangle$. The *length* $\ell(p)$ of a path p is the sum of the weights of all contained edges and the *collective path length* $\mathcal{L}(\mathcal{P})$ of set of paths P as the sum of the lengths of the paths in the set. The *shortest path* $p_{sp}(s{\rightarrow}t)$ is the path with the lowest length among all paths that connect s to t. Last, the *similarity* of two paths p, p' from s to t is denoted by $Sim(p, p')$. Given a similarity threshold θ, paths p, p' are *sufficiently dissimilar* if $Sim(p, p'){<}\theta$. Also, a path p is sufficiently dissimilar to a set of paths P, if p is sufficiently dissimilar to every path in P. As determining the best similarity measure is out of the scope of our work, without loss of generality, we use the Jaccard coefficient.

[1] For ease of presentation, we draw a road network as an undirected graph in our examples. However, our proposed methods directly work on directed graphs as well.

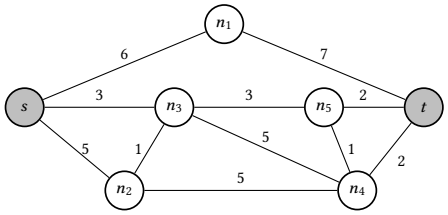


Figure 2: Running example.

We now formally restate the *Top-KSPD* problem [11] as the *k-Dissimilar Paths with Minimum (Collective) Length (kDPwML).*

Problem 1 (*k*DPwML). *Given a road network $G{=}(N, E)$, a source $s{\in}N$, a target $t{\in}N$, a number of requested paths k, and a similarity threshold θ, the P_{kDPwML} is the set of paths from s to t, such that:*

(A) all paths in P_{kDPwML} are pairwise sufficiently dissimilar,
(B) $|P_{kDPwML}| \leq k$ and P_{kDPwML} has the maximum possible cardinality among every set of paths P_A that satisfy Condition (A),
(C) P_{kDPwML} has the lowest collective path length among every set of paths P_{AB} that satisfy both Conditions (A) and (B),

Consider the road network in Figure 2 along with paths $p_{sp} = \langle (s, n_3), (n_3, n_5), (n_5, t)\rangle$, $p_2 = \langle (s, n_3), (n_3, n_5), (n_5, n_4), (n_4, t)\rangle$, $p_3 = \langle (s, n_3), (n_3, n_4), (n_4, t)\rangle$, and $p_4 = \langle (s, n_2), (n_2, n_3), (n_3, n_5), (n_5, t)\rangle$. Let $P_1 = \{p_1, p_2, p_3\}$, $P_2 = \{p_1, p_3, p_4\}$, and $P_3 = \{p_2, p_3, p_4\}$ be three sets of paths with $\mathcal{L}(P_1) = 27$, $\mathcal{L}(P_2) = 29$, and $\mathcal{L}(P_3) = 30$. Consider the query *k*DPwML$(s, t, 3, 0.5)$. While set P_1 has the lowest collective length, it cannot be the result set as p_1 and p_2 are not sufficiently dissimilar, i.e., $Sim(p_1, p_2) = 6/11 = 0.545 > \theta$. On the other hand, both P_2 and P_3 contain sufficiently dissimilar paths, but P_2 is preferred as $\mathcal{L}(P_2) < \mathcal{L}(P_3)$. In fact, P_2 is the set with the lowest collective length among all sets that contain three sufficiently dissimilar paths. Hence, P_2 is the result of the query.

Next, elaborate on the complexity of the *k*DPwML problem. Liu et al. proved in [11] (cf. Lemma 1) the *NP*-hardness of *k*DPwML. Despite the correctness of their finding, the authors' approach on the proof is incorrect as they polynomially reduced *k*DPwML *to* a hard problem, i.e., the Maximum Independent Set problem, instead of providing a polynomial reduction *from* a hard problem. In view of this, we hereby prove the following theorem.

Theorem 2.1. *The kDPwML problem is strongly NP-hard.*

Proof. We prove the lemma by polynomial reduction from the two edge-disjoint path problem (2-DP), which is known to be strongly *NP*-complete [7]. Given a directed graph $G{=}(N, E)$ with $|N|{=}n$ and two source-target pairs (s_1, t_1) and (s_2, t_2), 2-DP asks whether G contains edge-disjoint s_i–t_i paths for $i = 1, 2$. For polynomially reducing 2-DP to the *k*DPwML problem, we define a road network $G'{=}(N', E')$ with $N'{=}N \cup \{s, t, a, b, c, d\}$ and $E'{=}E \cup \{(s, a), (s, c), (a, b), (c, d), (b, s_1), (d, s_2), (t_2, a), (t_1, c), (b, t), (d, t)\}$. We set $k{=}4$, $\theta{=}(2n+1)/(4n^2+5)$, $w(e){=}4n$ for all $e \in E$, and $w(e){=}1$ for all $e \in E' \setminus E$. Then there are edge-disjoint s_i–t_i paths in G just in case the result of a 4-DPwML query against G' has cardinality 4. The proof for this claim is available in the extended version of our paper [6]. Hence, unless $P = NP$, there can be no polynomial or pseudo-polynomial algorithm for answering *k*DPwML queries. □

3 AN EXACT APPROACH

A naïve approach for an exact solution to kDPwML would first identify all paths from a source node s to a target node t and examine all possible sets of at most k paths to determine the set that satisfies the conditions of Problem 1. Such an approach is clearly impractical. In view of this, we present a pair of pruning techniques along with an exact algorithm that employs these techniques to reduce the search space during kDPwML query processing.

Our first pruning technique employs a lower bound on the collective path length, which limits the total number of paths to be constructed. Let P_{all} be the set of all possible paths from node s to t, p_i^* be the i-shortest path in P_{all}, and P_{k-1}^* the set of the $k-1$ shortest paths in P_{all}. For every path $p \in P_{all}$, the collective path length of every $P \subseteq P_{all}$ of k paths that contains p is lower bounded by the sum $\ell(p) + \mathcal{L}(P_{k-1}^*)$; by definition, $\mathcal{L}(P_{k-1}^* \cup \{p\})$ has the lowest collective path length among all subsets of k paths that include p. Hence, during the processing of a kDPwML query, by examining paths in length order we can terminate the examination if a constructed path p violates the aforementioned lower bound. Our second pruning technique prevents the construction of path sets that do not exclusively contain sufficiently dissimilar paths. For this purpose, we employ a dynamic programming scheme named "filling a rucksack" or Algorithm F for simplicity [10], which generates path sets by reusing already generated smaller subsets. Consequently, when computing a kDPwML query, it suffices to incrementally generate new subsets of paths and early prune subsets that contain at least one pair of not sufficiently dissimilar paths.

Algorithm 1 illustrates the pseudocode of our exact KSP-DML algorithm, which employs the two aforementioned pruning techniques. All generated shortest paths are stored in set P_{sp}. In Line 2, we set p as the first shortest path from s to t. From Line 3 to 11, KSP-DML iterates over the next shortest path starting from the first one. In Line 4, current path p is stored in P_{sp}. In Lines 5–6, the collective length $\mathcal{L}_{k-1}$ of the first $k-1$ shortest paths is computed to be used for the lower bound in Line 3. Next, Algorithm F [10] is called in Line 7 to determine all ($\leq k$)-subsets P of P_{sp} that contain p. For each subset P, Line 8 checks whether all paths in P are sufficiently dissimilar to each other (Condition (A) of Problem 1). Subsequently, Line 9 checks whether P contains more paths than $P_{k\text{DPwML}}$ (Condition (B) of Problem 1), or P contains as many paths as $P_{k\text{DPwML}}$ and P has a lower collective length than current $P_{k\text{DPwML}}$ (Condition (B) and (C) of Problem 1). If either case holds, KSP-DML updates the $P_{k\text{DPwML}}$ result in Line 10. The examination of each next shortest path (and KSP-DML overall) terminates when either all possible paths from s to t have been generated or the termination condition of Line 3 is met.

Algorithm 1: KSP-DML

Input: Road network $G = (N, E)$, source node s, target node t, number of results k, similarity threshold θ
Output: Set $P_{k\text{DPwML}}$ of at most k paths

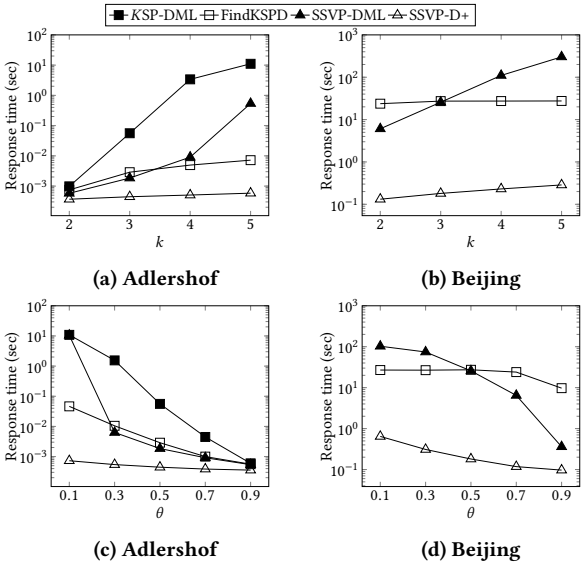
```
1  initialize P_all ← ∅, L_{k-1} ← 0, P_kDPwML ← ∅, P ← ∅;
2  p ← NextShortestPath(G, s, t);                    ▷ Shortest path p_sp
3  while p≠null and (|P_kDPwML|<k or ℓ(p)+L_{k-1} ≤ L(P_kDPwML)) do
4      P_all ← P_all ∪ {p};
5      if |P_all| < k then
6          L_{k-1} ← L_{k-1} + ℓ(p);
7      foreach P ⊆ P_all : |P| ≤ k with p ∈ P do          ▷ Alg. F [10]
8          if ∀ p_i, p_j ∈ P with i ≠ j : Sim(p_i, p_j) < θ then
9              if |P| > |P_kDPwML| or
                 |P| = |P_kDPwML| and L(P) < L(P_kDPwML) then
10                 P_kDPwML ← P;                        ▷ Update result set
11     p ← NextShortestPath(G, s, t);
12 return P_kDPwML;
```

4 HEURISTIC APPROACHES

Despite the pruning criteria introduced in Section 3, we expect the number of paths examined by KSP-DML to be significantly larger than the requested number of results k. In view of this, to reduce the search space even further, a popular solution is to iterate over the much smaller set of *single-via paths* proposed by Abraham et al. [2]. Given a road network $G = (N, E)$, a source node s and a target node t, the single-via path $p_{sv}(n)$ of a node $n \in N \setminus \{s, t\}$ is defined as $p_{sp}(s \to n) \circ p_{sp}(n \to t)$, i.e., the concatenation of shortest paths $p_{sp}(s \to n)$ and $p_{sp}(n \to t)$. By definition, $p_{sv}(n)$ is the shortest possible path that connects s and t through n. However, using the set of single-via paths to process kDPwML queries raises two important issues. First, the single-via path $p_{sv}(n)$ of every node n crossed by the shortest path $p_{sp}(s \to t)$ is identical to p_{sp}. Computing the single-via paths for these particular nodes is unnecessary. Second, there is no guarantee that a single-via path is simple (i.e., cycle-free), which make little sense as alternative paths from a user perspective.

Simple Single-via Paths. To address the aforementioned issues, we introduce the *simple single-via paths* (SSVP). Given a road network G=(N, E), a source s and a target t, the SSVP $p_{ssv}(n)$ of a node $n \in N \setminus \{s, t\}$ is defined only if n does not lie on the shortest path $p_{sp}(s \to t)$. If the single-via path $p_{sv}(n)$ of node n is simple, then $p_{ssv}(n)$=$p_{sv}(n)$. Otherwise, $p_{ssv}(n)$ is the concatenation either of $p_{sp}(s \to n)$ with the shortest path $p'(n \to t)$ from n to t that visits no nodes in $p_{sp}(s \to n)$, i.e., $sp(s \to n) \circ p'(n \to t)$, or the concatenation of the shortest path $p'(s \to n)$ from s to n that visits no nodes in $p_{sp}(n \to t)$ with $p_{sp}(n \to t)$, i.e., $p'(s \to n) \circ p_{sp}(n \to t)$. In this case, the SSVP of n is the concatenated path with the lowest path length. Note that the shortest path $p_{sp}(s \to t)$ is a SSVP by definition. Consider the road network in Figure 2. The single-via path of n_2 is $p_{sv}(n_2) = \langle (s, n_3), (n_3, n_2), (n_2, n_3), (n_3, n_5), (n_5, t) \rangle$, which is clearly not simple. Hence, the simple single-via path $p_{ssv}(n_2)$ is either $p = \langle (s, n_2), (n_2, n_3), (n_3, n_5), (n_5, t) \rangle$ or $p' = \langle (s, n_n 3), (n_3, n_2), (n_2, n_4), (n_4, t) \rangle$. In this particular case, both p and p' have the same length and either can be set as $p_{ssv}(n_2)$.

To compute all SSVPs it suffices to construct the shortest path tree $T_{s \to N}$ from s to all nodes $n \in N$, and the shortest path tree $T_{N \to t}$ to t from all nodes $n \in N$. The shortest paths $p_{sp}(s \to n)$ and $p_{sp}(n \to t)$ are retrieved from the shortest path trees, thus forming the $p_{sv}(n) = p_{sp}(s \to n) \circ p_{sp}(n \to t)$. If $p_{sv}(n)$ is not simple, $p'(s \to n)$ and $p'(n \to t)$ are computed by running Dijkstra's algorithm from s to n and n to t, respectively, while blacklisting nodes that need to be avoided during each run. Note that for the computation of a kDPwML query there is no need to construct all SSVPs at once. By examining nodes in length order of their single-via path and storing SSVPs in a priority queue temporarily, we restrict the total number of constructed paths. We elaborate more on the incremental construction of SSVPs in the extended version of this paper [6].



(a) Adlershof (b) Beijing

(c) Adlershof (d) Beijing

Figure 3: Response time varying requested paths k (θ=50%) and similarity threshold θ (k=3).

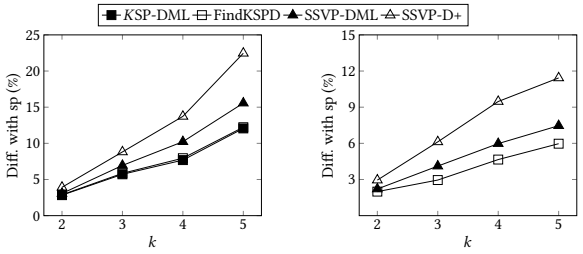
Algorithms. A straightforward way of employing SSVPs for processing kDPwML queries is to alter the exact KSP-DML algorithm from Section 3 such that the algorithm iterates over the simple single-via paths in increasing length order. By the definition of SSVP, the search space will be drastically reduced as at most $|N|-1$ paths will be examined. In addition, we can still use the pruning techniques of KSP-DML to terminate the search and avoid generating candidate path sets that do not contain sufficiently dissimilar paths. Our first heuristic algorithm, termed SSVP-DML, follows this approach. Naturally, as the result of kDPwML queries may not consist exclusively of simple single-via paths, SSVP-DML can only provide an approximate solution to the kDPwML problem.

Despite reducing the search space compared to KSP-DML, SSVP-DML still needs to examine a large number of subsets. In fact, we expect this procedure to dominate the evaluation of the kDPwML query. In view of this, we devise a second heuristic algorithm, termed SSVP-D+, which adopts a similar idea to FindKSPD [11] and to our previous work [5]. The algorithm constructs progressively a result set that (1) always contains the shortest path, and (2) every newly added simple single-via path to the set is both sufficiently dissimilar to the paths currently in the set and as short as possible.

5 EXPERIMENTAL EVALUATION

For our experiments we used the road networks of the city of Adlershof (349 nodes, 979 edges) extracted from OpenStreetMap, and the city of Beijing [9] (74,383 nodes, 222,778 edges). Apart from our algorithms, we include in the evaluation the FindKSPD algorithm, i.e., the greedy approach of Liu et al. [11]. Due to the limited space, we present only a part of our results, while our full results are presented in the extended version of our paper [6].

Figure 3 reports on the response time of the algorithms over 1,000 random queries varying the requested number of paths k (fixing θ to 0.5) and the similarity threshold θ (fixing k to 3). First,



(a) Adlershof (b) Beijing

Figure 4: Result quality varying requested paths k (θ=50%).

we observe that the exact algorithm KSP-DML is clearly impractical. For the road network of Adlershof and the default values of k and θ, KSP-DML may provide reasonable response time, but it is at least one order of magnitude slower than its competitors. Hence, we present the runtime of KSP-DML only for Adlershof. With regard to the heuristic algorithms we observe that SSVP-D+ is the fastest algorithm in all cases. For FindKSPD and SSVP-DML, the runtime increases with an increasing k and a decreasing θ. For an increasing k, though, the increase in the runtime of SSVP-DML is much more abrupt. With regard to θ, we observe that SSVP-DML is faster for large θ, while FindKSPD is faster for small θ values.

Figure 4 reports on the quality of the computed results. Considering only queries for which all algorithms returned k paths, we compare the average length of each result set to the length of the shortest path. We observe that the exact KSP-DML algorithm computes the paths with the smallest collective length on average. FindKSPD produces paths with an average length very close to KSP-DML. SSVP-DML comes next, while SSVP-D+ recommends the paths with the highest length on average. However, the difference between the result sets of KSP-DML and FindKSPD and the result sets of SSVP-DML and SSVP-D+ is quite small.

REFERENCES

[1] 2005. Choice Routing. Cambridge Vehicle Information Technology Ltd.. (2005).
[2] Ittai Abraham, Daniel Delling, Andrew V Goldberg, and Renato F Werneck. 2013. Alternative Routes in Road Networks. *Journal of Exp. Alg.* 18 (2013), 1–17.
[3] Vedat Akgun, Erhan Erkut, and Rajan Batta. 2000. On Finding Dissimilar Paths. *European Journal of Operational Research* 121, 2 (2000), 232–246.
[4] Theodoros Chondrogiannis, Panagiotis Bouros, Johann Gamper, and Ulf Leser. 2015. Alternative Routing: K-shortest Paths with Limited Overlap. In *Proc. of the 23rd ACM SIGSPATIAL Conf.* 68:1–68:4.
[5] Theodoros Chondrogiannis, Panagiotis Bouros, Johann Gamper, and Ulf Leser. 2017. Exact and Approximate Algorithms for Finding k-Shortest Paths with Limited Overlap. In *Proc. of the 20th EDBT Conf.* 414–425.
[6] Theodoros Chondrogiannis, Panagiotis Bouros, Johann Gamper, Ulf Leser, and David B. Blumenthal. 2018. Finding k-Dissimilar Paths with Minimum Collective Length. (2018). arXiv:cs.DS/XXXX.XXXX
[7] Tali Eilam-Tzoreff. 1998. The disjoint shortest paths problem. *Discrete applied mathematics* 85, 2 (1998), 113–138.
[8] Yeon-Jeong Jeong, Tschangho John Kim, Chang-Ho Park, and Dong-Kyu Kim. 2009. A Dissimilar Alternative Paths-search Algorithm for Navigation Services: A Heuristic Approach. *KSCE Journal of Civil Engineering* 14, 1 (2009), 41–49.
[9] Alireza Karduni, Amirhassan Kermanshah, and Sybil Derrible. 2016. A Protocol to Convert Spatial Polyline Data to Network Formats and Applications to World Urban Road Networks. *Scientific Data* 3, 160046 (2016).
[10] Donald E. Knuth. 2005. *The Art of Computer Programming, Vol. 4, Fas. 3: Generating All Combinations and Partitions.* Addison-Wesley Professional, Boston, MA, USA.
[11] H Liu, C. Jin, B Yang, and A. Zhou. 2017. Finding Top-k Shortest Paths with Diversity. *IEEE TKDE* 30, 3 (2017), 488–502.
[12] Jin Y Yen. 1971. Finding the K Shortest Loopless Paths in a Network. *Management Science* 17, 11 (1971), 712–716.