Neural Computing and Applications
https://doi.org/10.1007/s00521-022-07485-x

S.I.: EFFICIENT ARTIFICIAL INTELLIGENT ALGORITHMS FOR MEDICAL IMAGE ANALYSIS BASED ON HIGH-PERFORMANCE COMPUTING

Most relevant point query on road networks

Zining Zhang[1] · Shenghong Yang[1] · Yunchuan Qin[1] · Zhibang Yang[1] · Yang Huang[1] · Xu Zhou[1]

Received: 18 February 2022 / Accepted: 26 May 2022

Abstract
Graphs are widespread in many real-life practical applications. One of a graph's fundamental and popular researches is investigating the relations between two given vertices. The relationship between nodes in the graph can be measured by the shortest distance. Moreover, the number of paths is also a popular metric to assess the relationship of different nodes. In many location-based services, users make decisions on the basis of both the two metrics. To address this problem, we propose a new hybrid-metric based on the number of paths with a distance constraint for road networks, which are special graphs. Based on it, a most relevant node query on road networks is identified. To handle this problem, we first propose a Shortest-Distance Constrained DFS, which uses the shortest distance to prune unqualified nodes. To further improve query efficiency, we present Batch Query DFS algorithm, which only needs only one DFS search. Our experiments on four real-life road networks demonstrate the performance of the proposed algorithms.

Keywords Graph · Path enumeration · Relevant vertices · Road networks

1 Introduction

Graphs are used in many practical applications, such as social networks [1], information networks, gene networks, protein interaction networks, and road networks. It is a fundamental problem in graph data management to analyze the relationship between given two vertices; the relationship is popularly measured by the shortest distance [2, 3] or k-shortest paths [4, 5]. Based on them, two variants such as KNN [6, 7] and k-closest pairs [8, 9] are presented. We can make decisions, forecasts or classifications by analyzing the relationship between different vertices. For example, the applications of medicine include tracking infectious diseases, predicting drug side effects and calculating the effect of public health interventions. Recently, it has been applied to COVID 19 outbreak prediction. KNN is also used for classification in machine learning. In the field of medical image, people can judge whether patients have a dis ease, classify benign and malignant, and predict whether there are precursors of disease through machine learning.

Recently, path enumeration problem is proposed to assess how one vertex affects another vertex and attracts growing attention [10]. For example, in biological networks, it can gain the interaction chain by enumerating the paths between two substances [11]. In e-commerce networks, illegal acts such as money laundering can be checked by enumerating the paths of two entities and detecting whether there is a circle when a new edge is inserted [12]. Recently, [13, 14] research the path enumeration problem with a hop constraint, which is efficient to obtain limited essential paths.

✉ Shenghong Yang
ysh@hnu.edu.cn

Zining Zhang
zzn0107@hnu.edu.cn

Yunchuan Qin
qinyunchuan@hnu.edu.cn

Zhibang Yang
yangzb@ccsu.edu.cn

Yang Huang
hy19@hnu.edu.cn

Xu Zhou
zhxu@hnu.edu.cn

[1] College of Computer Science and Electronic Engineering, Hunan University, Changsha, Hunan, China

Published online: 28 June 2022

In many real-life applications, especially location-based services, it requires to compute the shortest paths. For example, we can select the nearest restaurant by calculating the shortest path between two locations in the road network. However, in many scenarios only determining the shortest path is not enough. Users may be interested in alternative paths which own some attributes but are longer than the shortest path. There have been many works about the shortest path with constraints [15]. For example, when we go to a restaurant within a few kilometers, we usually choose the nearest restaurant. However, some roads will be closed in real life because of accidents. The distance from other routes to the destination is much longer, so we will choose another restaurant. Inspired by this, we present a new metric to estimate the correlation between two points. Here, the number of paths under a distance constraint is utilized to measure the selectivity of destinations. The candidate node with more paths is more likely to be chosen.

As shown in Fig. 1, assume that s is the source point; t_1 and t_2 are target points. It needs to choose one of the points of interest as our target. There are two paths $p_1 = (s, v_1, v_4, t_1)$ and $p_2 = (s, v_2, t_1)$ from the original point to t_1 whose lengths are 3.5 km and 4.2 km, respectively. Besides, there are two paths $p_3 = (s, v_1, v_4, t_2)$ and $p_4 = (s, v_1, v_3, t_2)$ to t_2 with lengths 3.2 km and 5.1 km, respectively. Among these paths, path p_3 is the shortest, and we can choose t_2 as a final result.

However, all kinds of situations may occur on the road network. If the path (v_1, v_4) is blocked for some reason, we cannot reach t_1 and t_2 through the path (v_1, v_4). As a result, we can only reach t_1 through path p_2 and reach t_2 through path p_4. If we need reach the point of interest within a distance constraint 5 km in order to prevent the cost from being too high, path p_4 is not a good choice since it cannot reach t_2 within 5 km. But we can go through path p_2 to t_1 within 5 km. Finally, t_1 will be selected as the result instead of t_2. Motivated by the above scenario, the number of paths can be utilized to measure the relevance between points on road networks.

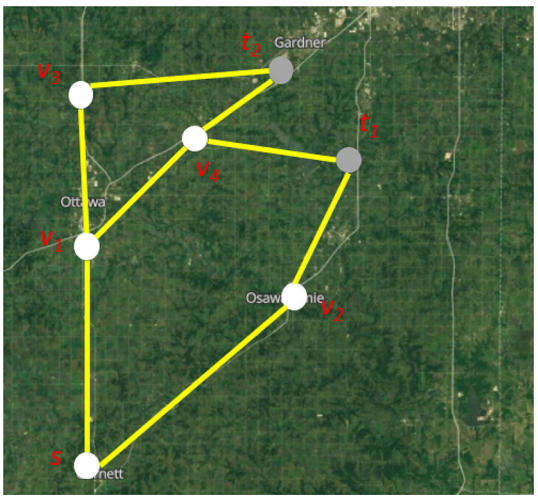

Fig. 1 An illustration of choosing the restaurant in a road network

In this paper, we present a new metric to measure the relevant of two nodes on road networks based on the number of paths within a distance constraint d. The more paths whose lengths do not exceed d, the stronger the correlation between the two nodes. In this way, it can ensure that the number of candidate paths to destination t is the largest under the condition of constraint d. Based on this new metric, the most relevant point query on road networks is formulated.

To process the most relevant point query effectively, it faces the following challenges. Firstly, the main obstacle is to calculate the number of paths satisfying the distance constraint from the source point to all destination points. The path enumeration problem is complex, and its time-consuming increases exponentially. Secondly, similar to the work in [14], with a small value of distance constraint d, it also involves a vast search space because the number of paths increases exponentially w.r.t d.

In this paper, to reduce redundant computation cost, we uses DFS to check the number of paths of all target points in a search process. In addition, unqualified nodes are pruned as early as possible by a lower bound of distance of each point in the search process. This contributes to accelerating the query procedure. Our principal contribution in this paper is summarized as follows.

- We analyze the limitation of the existing way of measuring the relevance of two points and propose a new metric, and formulate the most relevant point query on road network for the first time.
- We use the shortest distance to prune and use BC-DFS of [14] as the basic algorithm. At the same time, we prove that all results can be found in one search in Sect. 4.4, so we propose an optimization algorithm.
- We conducted experiments on four real road networks. Our optimization algorithm performs better when d is small than the basic algorithm. Moreover, because of some deficiencies in the experiment, we pointed out the direction of our future work.

1.1 Organization

The rest of this paper is organized as follows. We give related work in Sect. 2, and we introduce relevant concepts and formally define the problem in Sect. 3. In Sect. 4, we propose the basic scheme of the solution and the optimization algorithm. In Sect. 5, we conduct experimental research. This paper is summarized in Sect. 6.

2 Related work

In Sect. 2, we discuss some existing algorithms related to our problem.

2.1 KNN and K-closet pairs

KNN finds k-nearest neighbors from source point to target set. It is closely related to our life. We can use it to find nearby restaurants, close people and so on. Most of the existing algorithms are designed based on divide and conquer. They search and prune through partitions. [6] proposed an efficient index for KNN Search on road networks, called G-Tree. G-Tree adds two functions to adapt to KNN search on road networks compared with R-Tree. The first is the balanced tree structure, which can help prune the subtree. The road network is recursively divided into subnetworks using a multi-level graph partition algorithm [16]. Each subnetwork corresponds to a node of the G-Tree. The algorithm ensures that the number of boundary points is as few as possible but also that the size of each subgraph is almost the same. The second is to effectively calculate the minimum distance from the query location to the tree node for the best first search. Save the shortest distance from all vertices to boundary points in the leaf node. Save the distance of all boundary points of child nodes in non-leaf nodes. The minimum distance from the point to the tree node is calculated by dynamic programming. The tree node is added to the priority queue for the best priority search if the minimum distance of the tree node is greater than the distance of the k-th neighbor. Recently many different methods have been applied to solve these problems. For example, [17] extended their algorithm to GPU, which greatly accelerated the process. When we solve these problems in a higher-dimensional space, the complexity increases dramatically. Machine learning, which is very popular right now, is used for solving KNN (e.g., [18, 19]).

K-Closet Pairs is extended from KNN. K-Closet Pairs finds k-pairs from source set to target set. [20] proposed a pruning heuristic and two updating strategies for minimizing the pruning distance and use them in the design of three non-incremental branch-and-bound algorithms for K-CPQ between spatial objects stored in two R-trees. [21] studied the problem of processing KCPQs between RAM-based point sets, using plane-sweep (PS) algorithms. [9] proposed G^*-Tree solve the K-Closet Pairs problem based on G-Tree. The goal is to maximize the minimum network distance between subgraphs. Therefore, it uses LEM [22] to select two subgraphs with the shortest distance for folding iteratively. Another difference is that it saves the minimum network distance between each pair of boundary nodes of any two different leaf nodes of the G^*-Tree. [23] proposed a branch-and-bound framework associated with effective lower and upper bound pruning techniques and early stopping conditions for efficiently retrieving relevant top-k closet pairs.

2.2 Shortest path enumeration

The "shortest path" is the shortest of all paths between two points. To enumerate all paths within d, we can keep on using the k-shortest paths algorithm by increasing k until the shortest path detected exceeds the distance constraint d where k is the number of paths. There are many classic algorithms (e.g., [4, 22, 24–26]). The most representative work is Yen's algorithm [27]. Yen finds the next shortest path by continuously deviating from the current shortest path. Many existing algorithms are optimized on this basis. For example, [4] abstracts the shortest distance into a point. Moreover, they use the Yen algorithm and landmark index to find the lower bound; they can then use the best-first search algorithm to prune. Theodoros Chondrogiannis studied an interesting problem in [5]. They aim to find k-shortest paths that are sufficiently dissimilar and as short as possible. To compute kSPwLO (k-shortest paths with limited overlap) queries, they proposed two exact algorithms: one-pass and multi-pass. They also study two classes of heuristic algorithms: (a) performance-oriented heuristic algorithms that trade shortness for performance. (b) Completeness-oriented heuristic algorithms that trade dissimilarity for completeness. Their performance is not ideal when used in the road network and multi-target points.

2.3 Simple path enumeration

There are some existing works on the problem of enumerating s-t simple paths [13, 14, 28–33]. In [28], their focus is how to construct a succinct presentation of simple paths. [30, 31] have proposed polynomial delay algorithms for the s-t path enumeration problem, but the actual effect is not ideal.

In recent years, You Peng et al. have done many studies. [14] studies the Hop-constrained s-t simple path enumerating. So that people pay attention to the limited important paths. They proposed BC-DFS and JOIN to solve this problem.The idea of BC-DFS is "do not fall into the same trap twice by learning from mistakes." JOIN searches from source vertex and target vertex to find the middle vertices cut. Then they will join it based on the middle vertices cut. JOIN has a good performance. The time complexity is $O(km\alpha)$, and the space is bounded by $O(\alpha)$ where α is the number of hop-constrained s-t paths, m is the number of edges, and k is the hop constraint. However, JOIN has used

an unweighted graph. We cannot find the middle vertices cut in the weighted graph. Then [32] proposed the first FPGA-based algorithm PEFP to solve the problem. On the host side, they reduce the graph size and search space by a preprocessing algorithm, Pre-BFS. On the FPGA side in PEFP, they proposed a novel DFS-based batching technique to save on-chip memory efficiently. Thanks to hardware acceleration, the performance is better than JOIN. The latest research on their work is [33]. In addition to the existing JOIN and BC-DFS, they also proposed the SCB algorithm. The main idea of SCB is that when we find a result using DFS method, we need to go back to at most k steps to get a new valid sub-paths with blocking some vertices. Many invalid vertices could be avoided during the process if they violate the diversity constraints.

3 Preliminary

In this section, we introduce relevant definitions. We mainly study the situation in the road network. Firstly, the road network is transformed into a graph, and then the related concepts and problem definitions are introduced.

3.1 Road networks

A road network is modeled as an undirected weighted graph $G = (V, E, W)$, where V is a vertex set and $E \subseteq \{(v_i, v_j) \mid v_i, v_j \in V \wedge v_i \neq v_j\}$ is an edge set. A vertex $v_i \in V$ is either a road intersection or an end of a road, and an edge $e_k = (v_i, v_j) \in E$ represents a road segment that enables travel between vertices v_i and v_j. W assigns a real-valued weight $w(e)$ to an edge e that represents the corresponding road segment's length.

3.2 Distance-constrained s-t path

A path p from the vertex v to the vertex v_h is a sequence of vertices $p = (v_0, v_1, \ldots, v_h)$ such that $(v_{i-1}, v_i) \in E$ for every $i \in [1, h]$. In this paper, we denote a path from u to v by $p(u, v)$. A simple path is a loop-free path where there are no repetitions of vertices and edges. By $len(p)$, we denote the length of the path p (i.e., the sum of the weights of each edge of the path p). If $len(p) \leq d$ where d is the pre-defined distance constraint, we say a path p is a distance-constrained $s - t$ path. For simplicity, we use $dc - s - t$ path to denote distance-constrained $s - t$ simple path.

3.3 Problem definition

Given two vertices u and v, let $num(u, v)$ denote the number of $dc - s - t$ paths from u to v. The more paths between u and v, the more relevant they are. Given a graph G, a distance d, the source vertex s and the target set T, a most relevant point query returns a point t where t satisfies the following conditions.

$$\begin{cases} t \in T \\ \forall v \in T, num(s, t) \geqslant num(s, v) \end{cases} \quad (1)$$

4 Search algorithm

4.1 Basic idea

To solve the problem in this paper, we introduce two solutions.

4.2 Shortest-distance constrained DFS

A simple solution is to start the DFS search from the source point. We stop the search when the target point is found or the length exceeds d. When the current point's qualified paths have been found, we continue to search the number path of the following target point. It is worth noting that the search does not touch the existing vertices in the *visited* set to avoid loops. After all branches of the current vertex have been accessed, it needs to be cleared from the *visited* set.

To reduce the amount of computation, we use simple pruning to reduce the unnecessary search. We can calculate the shortest distance from the target point to all points in the search process. Our scene is under the condition of distance constraint d. When we apply Dijkstra to calculate the shortest distance, if the shortest distance is greater than d, we set $sd[u]$ infinite means unreachable with distance d. When the existing length plus the shortest distance (i.e., $sd[u]$) is greater than d, it can be pruned directly, because $sd[u]$ represents the minimum length required from u to the endpoint.

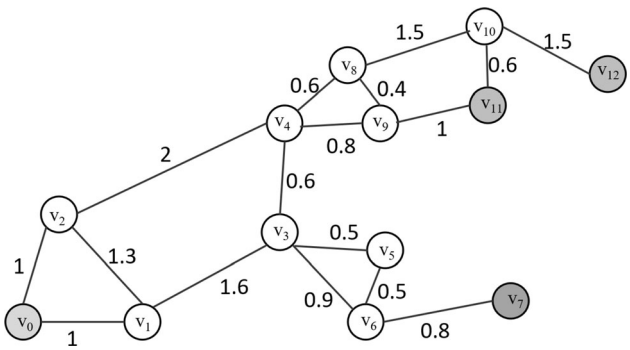

Fig. 2 An example by using SC-DFS

Figure 2 shows an example of the basic method. Given a graph G, the source vertex v_0, the target set $T = v_7, v_{11}, v_{12}$, the distance constraint $d = 5$km. The number on the edge represents the actual distance between two vertices. First, we compute the shortest distance from target vertices. The shortest distance $sd[v_1]$ is 3.3 km for v_7, so if we explore v_1 through v_2, the distance used plus the $sd[v_1]$ is $2.3\text{km} + 3.3\text{km} > 5\text{km}$. We do not need to continue exploring. The same as the v_4. We can find two paths within 5 km. Path $p_1 = (v_0, v_1, v_3, v_5, v_6, v_7)$, path $p_2 = (v_0, v_1, v_3, v_6, v_7)$. Their lengths are 4.4 km and 4.3 km The shortest distance $sd[v_1]$ is 5.6 km and $sd[v_2]$ is 5.8 km for v_{12}. So the vertex v_{12} does not need to explore. Similarly, we can find three paths within 5 km for v_{11}. Path $p_3 = (v_0, v_2, v_4, v_9, v_{11})$, path $p_4 = (v_0, v_2, v_4, v_8, v_9, v_{11})$. Path $p_5 = (v_0, v_1, v_3, v_4, v_9, v_{11})$. Their lengths are 4.8 km, 5 km and 5 km. The $num(v_0, v_{11}) = 3$, the shortest path is $p_3 = 4.8$km. Although the shortest path is $p_2 = 4.3$km, the $num(v_0, v_{11}) > num(v_0, v_7)$. We will choose the point v_{11} as result.

The pseudocode of SC-DFS is shown in Algorithm 1. We calculate the $num(s, t)$ of each pair by **CalculateNum**(lines 4). Before that, we compute the shortest distance by applying Dijkstra (lines 3). Then we choose the point t whose path number is maximal (lines 5-7). Finally, t is returned as the most relevant point.

As shown in Algorithm 2, we invoke **CalculateNum()** to compute the number of paths between two points. Initially, a set *visited*[*u*] is initialized to *false*, and the current path distance *dis* is initialized to 0. If the target node is visited at current, we increase the number of paths by one (lines 2-4). Besides *visited*[*u*] is utilized to check whether *u* has been accessed (lines 8). Lines 9-10 check whether the current path satisfies the distance constraint.

Algorithm 1: SC-DFS

```
   Input: source point s ∈ V,target set T
   Output: the most relevant point
1  Initialize max←0 ;
2  foreach v∈T do
3      Compute the shortest distance from v;
4      Compute the num(s, v) by CalculateNum(s,0,v);
5      if num(s, v) > max then
6          max←num(s, v);
7          t←v;
8  return t;
```

4.3 Barrier-based constrained DFS

The second baseline algorithm draws on the idea of [14], namely BC-DFS. The idea of BC-DFS is "do not fall in the same trap twice by learning from mistakes." BC-DFS will explore wrong branches, but it also learns from the mistakes. For each outgoing neighbor v of u to be visited in

Algorithm 2: CalculateNum(u, dis, t)

```
   Input: road network G,the vertex to be processed u,temporary
          distance dis,target point t
   Output: num(s, t)
1  Initialize visited[]←∅, dis←0 ;
2  if u == t then
3      num(s, t) + + ;
4      return;
5  else
6      visited[u]←true;
7      foreach neighbor w of u do
8          if visited[w] == fasle then
9              if dis + weight(w, u) + sd[w] < d then
10                 CalculateNum(w, dis + weight(w, u), t);
11     visited[u]←fasle;
```

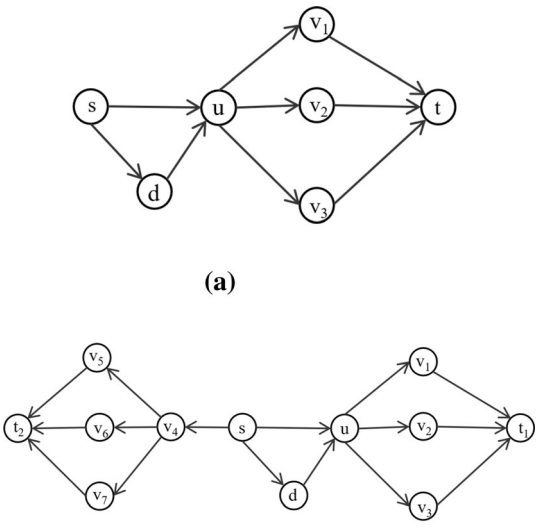

Fig. 3 An example by using BC-DFS

BC-DFS, BC-DFS will set a barrier for v. If "$|S| + 1 + barrier[v] > k$", the search will not continue. There is an example in Fig. 3a. In this example, the hop constraint $k = 3$ and search stack $S = \{s, u\}$. We will explore the outgoing neighbor (v_1, v_2, v_3) of u. But we cannot reach t with the hop constraint $k = 3$. So we set the barrier of u is 2. It represents at least 2 hop from u to t. Then when we are searching another path $p = \{s, d\}$, we will not continue to explore u. Because the length of stack S plus the barrier of u is 4, $|S| + 1 + 2 = 4 > 3$.

BC-DFS is a polynomial delay algorithm with $O(km)$ time per output where k is the hop constraint, and m is the number of edges. We can extend it to the road network. However, the performance of BC-DFS is not good in some cases. As shown in Fig. 3b, the barrier of u does not affect the vertices on the left. We set the barrier of u to 2 after we search the path $p = \{s, u\}$. However, when we explore v_4 on the left, it will not reach u again. The setting of some obstacles does not affect other parts. Secondly, for our problem, it still needs one to one search and needs to set a different barrier for source vertices.

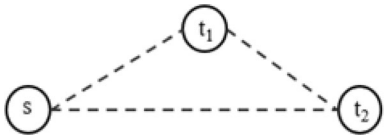

Fig. 4 Proof outline for BQ-DFS

4.4 Batch query DFS

As discussed in Sect. 4.2, SC-DFS uses the shortest distance to prune unqualified nodes. BC-DFS cannot be utilized to our problem in this paper. It is since its inspection cost is too high and it is not easy to set the barrier for multi-target points. Inspired by the Dijkstra algorithm, we can share the paths searched. The subpath of the shortest path is also the shortest path, so that the Dijkstra algorithm can find all shortest paths in one search from one point. We also prove that it can gain the number of paths for all nodes only in one search. Accordingly, we will not stop the search procedure when we explore a target.

Theorem 1 *The number of paths for all target points can be found in BQ-DFS.*

Proof As shown in Fig. 4, suppose that the target points are t_1 and t_2. There are two ways from the source point s to the target point t_2. One is to reach t_2 through t_1(i.e., $s- > t_1- > t_2$), and the other is not to reach t_2 through t_1(i.e., $s- > t_2$). For the first case, when we reach t_1 by using BQ-DFS, we will record the path number of $s- > t_1$ and the paths from s to t_1 have been explored. We will not stop and continue to search downward. If the path's length from s to t_1 is $length(s, t_1)$, then we just need to explore the path from t_1 to t_2 within $d - length(s, t_1)$. d is the constraint distance. The path from s to t_1 can be shared. But in SC-DFS, when we reach t_1, we will stop and restart the search from s. A large number of repeated searches can be reduced because the paths before t_1 can be shared in BQ-DFS. For the second case, t_1 does not affect these paths. Similarly, when there are multiple target points, the paths of other target points can also be searched by sharing the explored paths in one search.

Algorithm 3: BQ-DFS(u,dis)

Input: road network $G = (V, E, W)$,source $s \in V$,target set T,constraint distance d
Output: the most relevant point

```
1  Initialize max←0,visited[]←∅, dis←0 ;
2  if T.find(u)! = ∅ then
3      num(s, u) + + ;
4  else
5      visited[u]←true;
6      foreach neighbor w of u do
7          if visited[w] == fasle then
8              if dis + weight(w, u) <= d then
9                  BQ-DFS(w, dis + weight(w, u));
10     visited[u]←fasle;
11 foreach v ∈ T do
12     if num(s, v) > max then
13         max←num(s, t);
14         t←v;
15 return t;
```

Considering the pruning strategy in SC-DFS is too expensive for multiple vertices. We do not use the shortest distance as the lower bound. In BQ-DFS, we find all paths in one search. If we use the shortest distance to prune, we must choose the minimum distance as the lower bound to ensure that the resulting search space is not pruned. So the points originally cut out in SC-DFS cannot be reduced here.

4.5 Example

In BQ-DFS, since Dijkstra costs too much and has poor pruning efficiency, we stop the BQ-DFS algorithm by the constraint d. Figure 5 shows an example. The constraint is $d = 5$ km. When we reach t_1, we will record the number of the path from s to t_1. Then we will continue to explore. The next vertex is t_3, and the current length is 4.5 $km < d$. We can record the number of the path from s to t_3. When we search v, the current distance is 5.1 $km > d$. We will stop searching and backtrack.

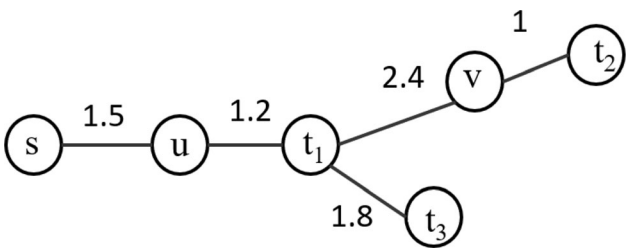

Fig. 5 An example by using BQ-DFS

Algorithm 3 shows BQ-DFS's pseudocode. We just use d to check whether continue search (lines 8-9). When we search one target point, we record number of the path. And we would not stop exploring (lines 2-3). Finally, we select the point as a final result of the most relevant point query.

5 Experiments

In this section, we evaluate the efficiency of proposed algorithms by comprehensive experiments.

5.1 Experimental setting

5.1.1 Datasets

We evaluate our algorithms on three real-world datasets, namely Amsterdam, Berlin and Oslo, which are road networks [34], with a size of hundreds of thousands of vertices. Besides, we use the real traffic network in Beijing, which the number of vertices and edges is more than one hundred thousand. In order to explore the influence of edge weight, we also calculate the average weight of each road network. The statistics of these datasets are illustrated in Table 1.

5.1.2 Query sets

To evaluate the search performance, we randomly choose 100 vertices as the query location, and for each query location, we generate 50 groups of target objects. We set d based on the average weight and report the average query

Table 1 Datasets

Data	Number of vertices	Number of edges	Average weight
Amsterdam	106, 600	130, 091	28
Berlin	428, 769	504, 229	31
Oslo	305, 175	330, 633	16
Beijing	165, 990	225, 998	343

response time of the algorithms to evaluate their time efficiency.

5.1.3 Algorithms

We evaluate the performance of three algorithms as follows:

- SC-DFS: The distance-constrained DFS pruned by shortest distance presented in Sect. 4.2.
- BC-DFS: The barrier-based constrained DFS introduced in Sect. 4.3.
- BQ-DFS: The Batch Query DFS presented in Sect. 4.4.

5.1.4 Implementation

All algorithms were implemented in C++ and conducted on an Intel(R) Core(TM) CPU i7-7700HQ@2.80GHz with 16GB RAM.

5.2 Evaluation of algorithms

Firstly, we compare the average running time of three algorithms (SC-DFS, BC-DFS, BQ-DFS) on four road networks. To achieve similar performance, the distance constraint d is set to 1500 m in the Beijing road network, while for the others, the d is set to 400 m.

As shown in Fig. 6, the average running time of these three algorithms increases with the graph size grows. The overall performance of BQ-DFS is the best, followed by BC-DFS, and the performance of SC-DFS is the worst. BQ-DFS can be regarded as an upgraded version of SC-DFS, because it can effectively improve the query performance by sharing the search cost. Although BC-DFS can terminate the search early, the barrier is set for one target point. For multi-target points, it becomes difficult to set the barrier. Therefore, it can only search nodes one by one, and its performance is worse than ours. SC-DFS has the worst performance among four road networks, and its average running time is dozens of times than that of other algorithms.

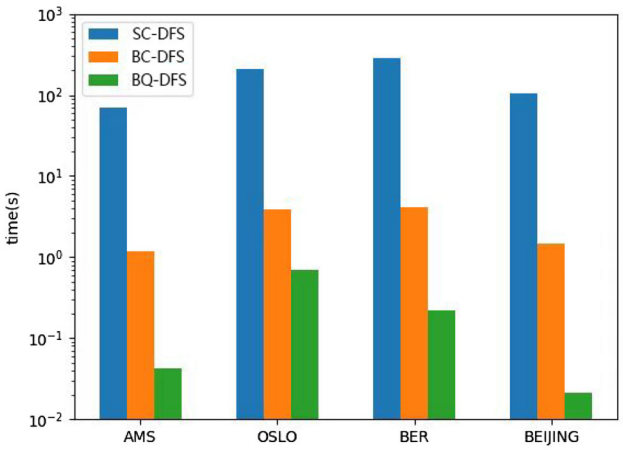

Fig. 6 Average runtime comparison

5.3 Effect of distance-constraint *d*

In the experiments of [14], they tested the average running time and the number of paths by varying the value of the hop constraint k and found that these two parameters grow exponentially with k. Therefore, we similarly evaluate the running time of the algorithm on different road networks by changing the distance constraint d. Differently, [14] uses the unweighted graph, while we use the weighted graph.

We set the distance constraint d according to their experimental results. According to the average weight of each road network in Table 1, we set d as 400 m, 500 m, 600 m, 700 m, 800 m, 900 m for AMS, BER and Oslo and set d as 1500 m, 2000 m, 2500 m, 3000 m for Beijing.

As shown in Fig. 7, for $d < 700\,m$ on Amsterdam, OSLO, Berlin and for $d < 2500\,m$ on Beijing road network, we observe similar results as those in Fig. 6. For $d >= 700\,m$ on Amsterdam, OSLO, Berlin and $d >= 2500\,m$ on the Beijing road network, the performance of BQ-DFS and BC-DFS becomes similar. In Sect. 5.4,we know that when d increases, the number of paths will explode, which makes the cost of searching very high. Since the average weight is small relative to d, an edge may be accessed many times. The number increased exponentially. In addition, we find that the low growth rate of SC-DFS is due to its utilization of the shortest distance for pruning, which takes much time to find the shortest distance for each point. Subsequently, it can be used for pruning, but the time still increases when d grows.

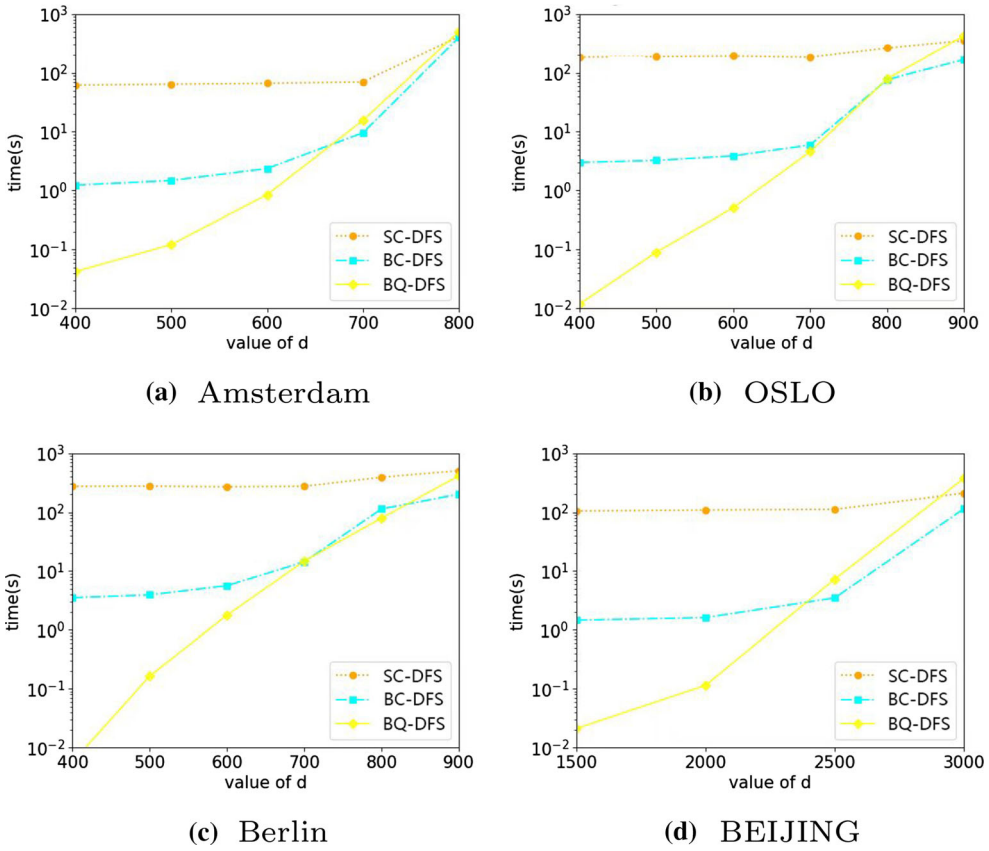

Fig. 7 Effect of distance-constraint d

5.4 Number of d-constrained paths

In this subsection, we report the average number of $dc - s - t$ paths on all datasets with different distance constraints values in Fig. 8. As expected, the number of $dc - s - t$ paths grows exponentially with d. The number of $dc - s - t$ paths increases rapidly because the weight of edges is too small; thus, the number of edges visited increases and the same edges will be visited repeatedly. However, these paths with large repetition are of little significance. We discussed it in our future work.

6 Conclusion

In this paper, firstly, we define a new relationship model with constraints used to evaluate the relationship between two points in the graph. Secondly, we propose a basic algorithm named SC-DFS. SC-DFS uses the shortest

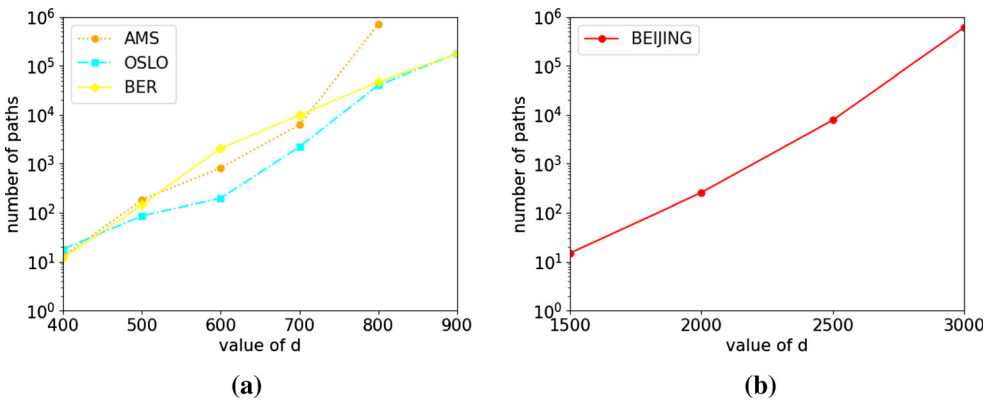

Fig. 8 Number of d-constrained paths

distance to prune. To improve efficiency, we propose a better algorithm called BQ-DFS. Specifically, BQ-DFS reduces the repeated searches because the paths explored can be shared. On the practical side, an extensive empirical study on four real-life graphs shows that BQ-DFS significantly outperformed BC-DFS when the distance constraint is small. In the future, we will boost the query performance by an index. Besides, it is also an interesting work to investigate parallel algorithms for the most relevant point query.

Acknowledgement This article is funded by the Program of Docking Research between Supercomputing and Big Data Management Services in Geospatial of the Department of Natural Resources of Hunan Province.

Author Contributions All authors contributed to the research concept and design. ZZ is responsible for the drafting of the experiment and paper, SY is responsible for the overall architecture and design of the paper, ZY is responsible for the experimental scheme, YH puts forward the optimization of the algorithm, and XZ puts forward ideas and data collection.

Declarations

Competing Interests The authors have no relevant financial or non-financial interests to disclose.

References

1. Liu B, Yuan L, Lin, X, Qin L, Zhang W, Zhou J (2019) Efficient (α, β)-core computation: An index-based approach. In: The World Wide Web Conference. WWW '19, pp 1130–1141. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3308558.3313522
2. Greco S, Molinaro C, Pulice C (2018) Efficient maintenance of shortest distances in dynamic graphs. IEEE Trans Knowl Data Eng 30(3):474–487. https://doi.org/10.1109/TKDE.2017.2772233
3. Li L, Zhang M, Hua W, Zhou X (2020) Fast query decomposition for batch shortest path processing in road networks. In: 2020 IEEE 36th international conference on data engineering (ICDE)
4. Chang L, Lin X, Qin L, Yu JX, Pei J (2015) Efficiently computing top-k shortest path join. In: EDBT
5. Chondrogiannis T, Bouros P, Gamper J, Leser U, Blumenthal DB (2020) Finding k-shortest paths with limited overlap. VLDB J (2)
6. Zhong R, Li G, Tan KL, Zhou L (2013) G-tree: An efficient index for knn search on road networks. In: Proceedings of the 22nd ACM international conference on conference on information and knowledge management
7. Zhang S, Li X, Zong M, Zhu X, Wang R (2018) Efficient knn classification with different numbers of nearest neighbors. IEEE Trans Neural Netw Learn Syst 29(5):1774–1785. https://doi.org/10.1109/TNNLS.2017.2673241
8. Gao Y, Chen L, Li X, Yao B, Chen G (2015) Efficient k-closest pair queries in general metric spaces. VLDB J 24(3):415–439. https://doi.org/10.1007/s00778-015-0383-4
9. Ahmadi E, Nascimento MA (2016) K-closest pairs queries in road networks. In: 2016 17th IEEE international conference on mobile data management (MDM), vol 1, pp 232–241. https://doi.org/10.1109/MDM.2016.44
10. Ferreira R, Grossi R, Marino A, Pisanti N, Rizzi R, Sacomoto G (2012) Optimal listing of cycles and st-paths in undirected graphs. Proc Ann ACM SIAM Symp Discrete Algorithms. https://doi.org/10.1137/1.9781611973105.134
11. Leser U (2005) A query language for biological networks. Bioinformatics (Oxford, England) 21(Suppl 2):33–9. https://doi.org/10.1093/bioinformatics/bti1105
12. Qiu X, Cen W, Qian Z, Peng Y, Zhang Y, Lin X, Zhou J (2018) Real-time constrained cycle detection in large dynamic graphs. Proc VLDB Endowment 11(12):1876–1888
13. Sun S, Chen Y, He B, Hooi B (2021) Pathenum: towards real-time hop-constrained s-t path enumeration
14. Peng Y, Zhang Y, Lin X, Zhang W, Qin L, Zhou J (2019) Hop-constrained s-t simple path enumeration: Towards bridging theory and practice. Proc VLDB Endow 13(4):463–476
15. Lu S, He B, Li Y, Fu H (2020) Accelerating exact constrained shortest paths on gpus. Proc VLDB Endow 14(4):547–559. https://doi.org/10.14778/3436905.3436914
16. Selvakkumaran N, Karypis G (1995) Analysis of multilevel graph partitioning. In: supercomputing, IEEE/ACM Sc95 conference
17. Barrientos RJ, Riquelme JA, Hernández-García R, Navarro CA, Soto-Silva W (2021) Fast knn query processing over a multi-node gpu environment
18. Naim A, Bowkett J, Karumanchi S, Tavallali P, Kennedy B (2021) Deterministic iteratively built kd-tree with knn search for exact applications
19. Vadiraja P, Balada CP (2021) Leveraging reinforcement learning for evaluating robustness of knn search algorithms
20. Corral A, Manolopoulos Y, Theodoridis Y, Vassilakopoulos M (2004) Algorithms for processing k-closest-pair queries in spatial databases. Data Knowl Eng 49(1):67–104
21. Roumelis G, Vassilakopoulos M, Corral A, Manolopoulos Y (2014) A new plane-sweep algorithm for the k-closest-pairs query. In: International conference on current trends in theory and practice of informatics
22. Karypis G, Kumar V (1998) A fast and high quality multilevel scheme for partitioning irregular graphs. Siam J Sci Comput 20(1):359
23. Wu F, Xie X, Shi J (2021) Top-k Closest Pair Queries over Spatial Knowledge Graph. Database systems for advanced applications, 26th international conference, DASFAA 2021, Taipei, Taiwan, 11–14 Apr 2021. Proceedings, Part I
24. Hu H, Lee DL, Xu J (2006) Fast nearest neighbor search on road networks. In: Advances in database technology—EDBT 2006, 10th international conference on extending database technology, Munich, Germany, 26–31 Mar 2006, Proceedings
25. Gotthilf Z, Lewenstein M (2009) Improved algorithms for the k simple shortest paths and the replacement paths problems. Inf Proc Lett 109(7):352–355. https://doi.org/10.1016/j.ipl.2008.12.015
26. Hershberger J, Maxel M, Suri S (2007) Finding the k shortest simple paths: a new algorithm and its implementation. ACM Trans Algorithms 3(4)
27. Jin YY (1971) Finding the k shortest loopless paths in a network. Manage Sci 17(11):712–716
28. Knuth DonaldErvin (2011) The art of computer programming. Volume 4A: Combinatorial Algorithms,Addison-Wesley Professional
29. Roberts B, Kroese DP (2007) Estimating the number of s-t paths in a graph. J Graph Algo Appl 11(1):195–214

30. Rizzi R, Sacomoto G, Sagot MF (2014) Efficiently listing bounded length st-paths. Eprint Arxiv 8986:318–329
31. Grossi R, Marino A, Versari L (2018) Efficient algorithms for listing k disjoint st-paths in graphs. Latin Am Symp Theor Inf
32. Lai Z, Peng Y, Yang S, Lin X, Zhang W (2020) Pefp: Efficient k-hop constrained s-t simple path enumeration on fpga
33. Peng Y, Lin X, Zhang Y, Zhang W, Zhou J (2021) Efficient hop-constrained s-t simple path enumeration. VLDB J (2)
34. Ahmadi E, Nascimento MA (2017) Datasets of roads, public transportation and points-of-interest in Amsterdam, Berlin and Oslo. https://sites.google.com/ualberta.ca/nascimentodatasets/