The Partition Bridge (PB) Tree: Efficient Nearest Neighbor Query Processing on Road Networks

Xiangqiang Min[a,b,c,d,e], Dieter Pfoser[c], Andreas Züfle[c], Yehua Sheng[b,d,*] and Yi Huang[b,d]

[a]School of Geomatics, Anhui University of Science and Technology, Huainan 232001, China
[b]Key Laboratory of the Virtual Geographic Environment, Ministry of Education of PRC, Nanjing Normal University, Nanjing 210023, China
[c]Department of Geography and Geoinformation Science, George Mason University, Fairfax 22030, USA
[d]Jiangsu Center for Collaborative Innovation in Geographical Information Resource Development and Application, Nanjing Normal University, Nanjing 210023, China
[e]Coal Mining-induced Disasters of Anhui Higher Education Institutes, Anhui University of Science and Technology, KLAHEI(KLAHEI18015), Huainan 232001, China

ARTICLE INFO

Keywords:
road network, *K* nearest neighbor query, hierarchical network partition, data distribution, spatial database

ABSTRACT

With the rapid development of location-based services, many types of applications in urban areas such as transportation planning, traffic management, and deployment of infrastructure depend on spatial objects that are distributed along the road network. As current place datasets include millions of spatial objects and human mobility datasets capture billions of locations, it is an important challenge to answer *k*-nearest neighbor queries efficiently. However, shortest-path distance calculations are the computational bottleneck for a *k*-nearest neighbor queries on road networks. Existing query algorithms partition the network to mitigate this bottleneck, but they do not take the data distribution into account, which leads to inefficiencies in dense areas and sparse areas. In this paper, an efficient and scalable indexing method, called the Partition Bridge tree (PB-tree), is proposed based on hierarchical network partitions that consider network connectivity and data distribution. The structure of the PB-tree mainly includes distance matrices, union-bridges, *bridges*, and active network nodes component. Based on the PB-tree, a *k* nearest neighbor query processing algorithm is proposed by combining "bottom-up", "top-down", and adjacent extension methods . By using different road networks and datasets, the effectiveness and practicability of PB-tree are evaluated; the experimental results show that PB-tree outperforms the state-of-the-art methods and the classical approach.

1. Introduction

With the continuous evolution of mobile devices and telecommunication technologies, we now witness substantial value in location-based services (LBS). In particular, in metropolitan areas, we observe many users that are content to subscribe to LBS, such as traffic assistance, location-based games, and location-based social networks [1–3]. As LBS became more popular, it has motivated the development of different types of nearest neighbor queries on road networks [4]. For example, for people going on a trip: (1) they want to find the five hotels closest to their conference venue; (2) they want to find the three seafood restaurants closest to their hotel; or (3) they may want to find the locations of their ten closest friends in a location-based social network. To support LBS on road networks, many companies (e.g., Google, Microsoft, and Tencent) provide road network maps enriched with points of interests, reviews, and other spatial objects (SOs) located on the edges of the spatial network. Finding the nearest SOs for a given query location is an important tool in any LBS [5]. When executing a (k) nearest neighbor (kNN) query the shortest network distances (SNDs) between the query point and SOs are computed and the (k) SO(s) that minimize this distance are returned. As shortest path computations are the bottleneck of kNN processing on networks [4], an efficient kNN query processing algorithm requires efficient solutions for shortest path computation.

Many nearest neighbor query processing solutions have been developed for road networks. They can be roughly classified into three categories: (1) Query method based on the Dijkstra algorithm [6–15]: The main idea is to extend the search area along the adjacent edge of the query point; then update the distance between the query point and its adjacent points until the k nearest neighbor (KNN) spatial objects are found. To improve efficiency, different indexing structures and pruning strategies are also proposed. (2) Heuristic expansion methods based on Best-First-Search (BSF): These methods find the most promising candidate objects based on heuristic strategies [16–23]. (3) Adjacent region expansion methods [24] which partition the road network into sub-networks based on set rules. When the kNN query is performed, the sub-network containing the query point are explored first followed by adjacent sub-networks. If no valid object is found, the process is repeated iteratively. All the above-methods have been proven to handle some specific applications. Usually, query methods based on the Dijkstra algorithm are suitable for small or middle-scale road networks with a high density of objects. Heuristic expansion modes have better performance for large range road networks with low-density SOs.The adjacent regions'

This work is partially supported by the Key Fund of the National Natural Science Foundation of China (grant number 41631175), and the National Science Foundation Grant No. 1637541.

*Corresponding author: shengyehua163@163.com
ORCID(s):

expansion modes search the middle range road network with smaller k value [25].

As the popularity of LBS increases, applications now create vast amounts of spatial objects. Rather than capturing hundreds of points of interests, modern LBS may now capture millions of user-generated spatial objects (such as comments or reviews), and spatial databases may capture billions of past user locations. Capturing such very large numbers of spatial objects creates challenges for efficient KNN query processing. Furthermore, the density of spatial objects is very heterogeneous. Some areas of a city that have many popular restaurants may have millions of spatial objects, whereas residential areas may have very few spatial objects. While existing solutions for spatial query processing on road networks often assume a homogeneous distribution of spatial objects, we propose an adaptive spatial kNN processing algorithm based on a novel data structure called the partition bridge tree (PB-tree). The core part of the PB-tree is to recursively partition the road network into sub-networks based on data distributions and connectivity of road edges. This recursive partitioning (i) ensures a balanced distribution of spatial objects among sub-partitions, rather than balancing the size of the partitioned regions and (ii) defines a partition tree structure on top of the underlying road network, with each tree-node representing a sub-network. The PB-tree is a balanced search tree where each partition a similar number of spatial objects. Thus, the PB-tree adapts to non-uniform distributions of spatial objects on a network. The structure of PB-tree is similar to that of HN-tree [26], but there are some differences:(1)The HN-tree just considers the high density of spatial objects, the PB-tree can adapt to high and low density spatial objects;(2) the query processing algorithm is new. In summary, the contributions of this paper are as follows:

(1) PB-tree with an adaptive and efficient tree indexing, which has low space overhead and better performance, is developed.

(2) An efficient query algorithm that combines top-down and bottom-up and adjacent extension methods is devised.

The remainder of this paper is organized as follows. Related works are summarized in Section 2. Section 3 introduces the context for PB-tree by defining road network, line graph, and graph partitioning. The hierarchical graph partitioning is described in section 4. We present the PB-tree index structure and query algorithm in Section 5. Experimental results and analysis are shown in Section 6. Section 7 summarizes the main contributions and highlights future work.

2. Related Work

Various spatial objects KNN algorithms on road networks can be classified into three types. In the following, the performance of some classical and the state-of-the-art KNN algorithms are reviewed in detail.

2.1. Query methods based on Dijkstra algorithm

An incremental network expansion (INE), which is based on the R-tree [6]. The query process, which is a network expansion approach, uses Dijkstra's algorithm to compute the distance between query point q to SOs until the termination condition is satisfied. Moreover, MkNN is presented by improving INE [7]. INE and MkNN can perform well on road networks with the high density of SOs, but for road networks with a low density of SOs, the query cost increases and performance degenerates. Safar proposed PINE performs KNN query by combining R-tree and Voronoi diagram [8]. The main process is to first locate the Voronoi unit containing the query point and to expand the other adjacent Voronoi units to check for other NN objects. Usually, when k is large, PINE performs worse. All distances for SOs with the radius$r$ should be precomputed [9]. The query processes include two steps: first, it checks the "Island" containing the query point; if the number of SOs is greater than k, the K nearest neighbor SOs are found; otherwise, it will use the Dijkstra algorithm to extend other "Islands" until the condition is satisfied. A unique continuous search algorithm (UNICONS), which is similar to "Island", is used to precompute NN SOs of cross-vertices [10]. Island and UNICONS have advantages for the different densities of SOs on road networks, but they can increase the query cost when k is more than the number of NN precomputed SOs.

Hu et al. develop the shortest path tree with horizontal edges (SPIE) that converts road networks into interconnected shortest-path trees [11]. SPIE views each SOs as vertices of the road network. The main processes of SPIE are first to find NN SOs in the descendant node of the query point q and then extend to the parent node of q, finally find the NN SOs in other descendant nodes of the parent node. The processes are repeated until the root node is accessed. SPIE can efficiently avoid the query cost of network expansion, but its performance is heavily dependent on the known spatial object sets. Moreover, many extra vertices are added in the process of SPIE construction, which greatly increases the network's scale. Huang et al. proposed the S-GRID indexing, which is based on the hash and grid [12]. The query processes of S-GRID first locate the grid containing the query point and then find the NN SOs within the grid; if the number of found SOs is less than K, the adjacent grids are checked using the Dijkstra algorithm. The S-GRID can quickly find satisfying SOs with efficient encoding technology. However, for a large and sparse road network and larger K, the performance declines greatly. The ROAD is the classical KNN method, its main objective is proposed to improve the Dijkstra algorithm on the hierarchical road network [13, 14]. The ROAD uses route overlay and associate directory to search vertices and SOs, respectively. In the query process, if a sub-network does not contain any SO, the shortcut strategy can skip the sub-network to improve the performance. However, for road networks with a uniform distribution of SOs, the performance of ROAD is the same as that of Dijkstra. HLDB is a novel KNN method

that is implemented completely in SQL [27]. The HLDB contains "forward" and "poilab" tables. In the query process, HLDB uses the bucked-based approach to obtain the results. However, the costs of preprocessing and space overhead are high.

2.2. Heuristic query methods

IER uses the Euclidean distance as the lower boundary to perform the search; the basic idea is based on a heuristic search pattern to extend the NN vertices [6]. When the Euclidean distance between query point and SOs is similar to the network distance between them, IER has good efficiency. Otherwise, the performance can be worse.

Spatially induced linkage cognizance(SILC) is proposed based on the consistency idea of the shortest path on road networks [16]. To improve performance, Distance Browsing(DisBrw) is proposed based on SILC [17]. However, if several SOs are distributed in certain areas, both methods become less efficient. Distance signature(Dist Sign) computes all distances between all vertices and SOs in advance and then encodes all distances based on distance range [18]. The query processes first check the SOs encoded as label 0, if the number of checked SOs is less than k, the objects of label 1 are identified. Although the classified distances are encoded to save the storage space, the costs of preprocessing and space overhead remain high. G-tree is a balanced search tree [19, 20]. First, it hierarchically partitions the whole road network into some sub-network with each sub-network representing a tree-node. Moreover, the distance matrices of union borders in each non-leaf-node are computed in advance, and the distances between all border vertices and vertices in the leaf nodes are be pre-computed. When G-tree performs the query, it first identifies leaf-node that contains the query point q, and then other tree-nodes are checked if the SOs of the leaf-node cannot satisfy the conditions. G-tree just computes the distances between border vertices or some vertices, which can improve efficiency. However, when the road network is a large scale with different densities of spatial objects, the KNN query performance can be improved.

The HN-tree is a hierarchical tree structure consider the data distribution, but it focuses on the range query of high density of spatial objects[26]. When the density of spatial objects is very low, the leaf node of HN-tree may contain many network nodes, thus, in the worst case, its performance is equal to that of the INE [6]. The structure of PB-tree is similar to that of HN-tree, but their main differences can be summarized as follows:(1) the graph partitioning of PB-tree considers the case for the low density of spatial objects, when the number of spatial objects in the partition is less than "fanout", the graph partitioning only considers topology structure, it can ensure that leaf nodes of PB-tree cannot contain many network nodes;(2) PB-tree adds the active network nodes and lowest active nodes to improve the shortcoming of HN-tree that is hard to adapt to the low density of spatial objects;(3) to improve PB-tree construction efficiency and KNN query performance, the distance matrices of PB-tree are improved which can adapt to low/high spatial objects.

2.3. Adjacent region query methods

VN^3 partitions the entire road network into many Voronoi units, the centers of that are each spatial object [24]. Furthermore, the distances between border points and the center point should be precomputed in each Voronoi unit. Besides, the R-tree is used to query each Voronoi unit. VN^3 is suitable for a middle-scale network with a middle or low density of spatial objects, and it performs well, especially for 1NN.

3. Problem Definition

The PB-Tree index structure supports efficient k-Nearest Neihgbour query processing on road networks. This section first provides a formal definition of road networks in Section 3.1 and formalizes the problem of finding the k Nearest spatial objects of a query location in a road network in Section 3.2. The following definitions in this section are equivalent to those in related works [1, 26].

3.1. Definitions

Definition 1 (Road network). *A road network can be represented as $G = \langle V, E \rangle$, V is the set of nodes, E is the set of network links that connect the nodes.*

We assume that spatial objects (such as user locations and points of interest) are located on edges of the road network. We formally define spatial objects as follows.

Definition 2 (Spatial Object (SO)). *Let $G = \langle V, E \rangle$ be a road network. A spatial object $so =< id, e, \alpha >$ is a triple where id is a unique identifier, $e \in E$ is the link identifier the object is located on, and α is the SO's relative location on link $e = (v_i, v_j)$. The (absolute) spatial location of S_{so} can be obtained as follows:*

$$S_{so} = S_{v_i} + \alpha \cdot (S_{v_j} - S_{v_i}), \alpha \in [0, 1].$$

We let SO define the set of all of spatial objects. In addition, all spatial objects are unchanging.

Our proposed approach is based on recursive partitioning of the road network into smaller subnetworks. For the correctness of our algorithms it is paramount that such as partitioning is lossless, that is, it does not discard any nodes or links of the road network. This observation is important, as traditional graph partitioning algorithm (such as the METIS algorithm [28]) partition the set of nodes. While such algorithm minimize the number of links "cut" between nodes of different partitions, such algorithm cannot avoid cutting links. A common strategy used in the literature for lossless partitioning of road networks is based on the concept of the Line Graph of a network. The line graph of a network represents each network link as a line graph node, and connects line graph nodes with edges if the corresponding links in the network share an adjacent node in the network. Intuitively, the line graph of a network allows to partition the links of the network rather than the nodes. The line graph can then be partitioned, and once line graph partitions are mapped back into road network representation, nodes

may appear (redundantly) in multiple partitions and all links are retained. More details on the concept of transforming a a road network into a line graph for lossless partitioning, including examples, can be found in [26]. Here, formally define the line graph of a road network as follows.

Definition 3 (Line Graph). *Given a road network $G = \langle V, E\rangle$, the corresponding line graph $\overline{G} = \langle \overline{V}, \overline{E}\rangle$ is constructed by defining each link $v_i \in V$ as a graph vertex $\overline{v_i} \in \overline{V}$, and defining an undirected edge $\overline{e} = (\overline{v_i}, \overline{v_j}) \in \overline{E}$ between any pair of vertexes $\overline{v_i}$ and $\overline{v_j}$ whose corresponding edges e_i and e_j in G are connected, i.e., share a common vertex. Thus, edges in $\overline{G}$ capture the adjacency relations among links in G. Each vertex $\overline{v}_i$ has a weight w_i that corresponds to the number of SOs on the corresponding link e_i on the road network. Notice that in the network model, the roads are bi-directional.*

3.2. k-Nearest Neighbor Query

Given a road network G enriched with spatial objects SO and a query location q, a k-Nearest Neighbor (kNN) query returns the set of k spatial objects having the smallest network distance from q. We formally define a kNN query as follows.

Definition 4 (k-Nearest Neighbor (kNN) Query). *Let $G = \langle V, E\rangle$ be a road network, SO be a set of spatial objects on G, and let q a query location on the road network (located at a node or link of the network). A k-Nearest Neighbor (kNN) query returns the set of spatial objects:*

$$\{kNN(q) \subseteq SO | \forall so \in kNN(q), so' \notin kNN(q) : dist(q, so) \leq dist(q, so'), |kNN(q)| = k\}$$

4. Hierarchical Graph Partitioning

In what follows, we briefly introduce the hierarchical graph partitioning approach that is used to iteratively partition a road network without loss of information, i.e., without "cutting" links connecting partitions and by at the same time considering the number of spatial objects in each partition. To illustrate this problem, assume a spatial partitioning $\mathcal{S}$ of the network topology created without considering the distribution of spatial objects. Let $s \in \mathcal{S}$ be one partition. In the worst case, all spatial objects may fall into s, thus all other partitions having zero spatial objects.

This observation implies that traditional partitioning schemes that do not consider the distribution of spatial objects may fail to meet their purpose of partitioning the data into smaller subsets that can be queried efficiently. Such imbalanced indexing may incur substantial computational overhead as many (potentially empty or nearly empty) partitions need to be explored and as such defeat the purpose of an index. Thus, a data driven (rather than a space driven) partitioning of the road network is essential.

Based on above analysis, the partitioning method fully considers both network topology and data distributions. Moreover, for dense SOs regions, such as urban centers, it should have a small extent. However, the partitions should have a large extents for widely scattered SOs, such as the countryside, where SOs are fewer than busy zones. Similarly, R-tree and quadtree are also based on the rule. Furthermore, the SOs of each partition should be closer than other groups; when we perform the query, a few node accesses can get more valid SOs. We design the hierarchical partitioning method based on the connectivity of network links and data distribution.

In this paper, the line graph $\overline{G}$ is partitioned as a hierarchy a series of sub-graph, where large sub-graphs at the upper level contain smaller sub-graphs at lower level. The definition of the hierarchical sub-graphs can be as below:

Definition 5 (Hierarchical sub-graph). *Let $\overline{G} = \langle \overline{V}, \overline{E}\rangle$ be the line graph of a road network $G = (V, E)$. $\overline{G}$ is partitioned into a series of sub-graphs ($sub\overline{G}_{11}$, $sub\overline{G}_{1j}$,..., $sub\overline{G}_{ij}$), where $i > 1, j > 1$, $sub\overline{G}_{ij}$=($\overline{V}_{ij}$, $\overline{E}_{ij}$). The hierarchical sub-graphs have the following properties in each level:*
(1) $\bigcup \overline{V}_{ij(1<j<N)} = \overline{V}$;
(2) For $k \neq s, \overline{V}_{ik} \cap \overline{V}_{is} = \varnothing$;
(3) SO sets of each sub-graph are independent

Graph partitioning is a critical part of indexing construction. Improving graph partitioning is always a challenging task. The main challenge is how to partition the vertex of the line graph into approximately equal parts; for instance, the number of edges in different sub-graphs should be minimized. Moreover, when each vertex has its weight, it should balance the weight of different partitions. It has been proven that the graph partitioning problem is an NP-problem [28, 29]In this paper, we do not try to solve this problem; instead, we adopt a high-quality partition algorithm to achieve our goal. We choose a public implementation METIS [28] to generate partitions.

The processes of the hierarchical graph partition can be described as follows. Given a road network G, a set of SOs, the number of bottom partitions(θ) as a termination criterion. First, the road network(G) is converted into line graph($\overline{G}$), the converting details can see [26]. Then, the ($\overline{G}$) is partitioned as a series of sub-graphs that are added into the list HP (hierarchical partitioning results set) using METIS [28]; Next, to identify the number of SOs in each sub-graph, if the num of SOs is more than *fanout*, the sub-graph is partitioned into partitions having equal number of SOs; otherwise, the sub-graph is partitioned based on the equally sized vertexes. This process is applied recursively when the number of the bottom sub-graphs is less than θ. Because the kNN query is based on the road network, these sub-graphs need to be converted into a network when the partition of the line graph is finished.

An example of a hierarchical graph partition using binary partitions (*fanout*=2) is shown in Figure 1, the termination criterion is that the number of the bottom partitions is not less than 4.For ease of presentation, the results and processes of hierarchical graph partition are shown using

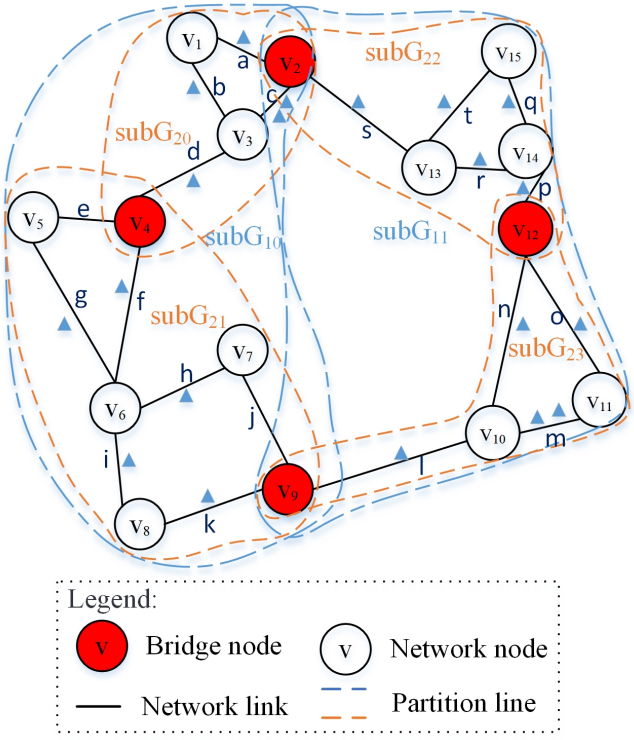


Figure 1: Hierarchical network partition

the road network. In Figure 1, different colors' broken lines represent different levels' partitions. The blue broken lines represent the partitions at the first level, the orange broken lines represent the partitions at the second level. Note that some nodes (highlighted in solid red) appear in multiple partitions. This redundancy is required to avoid loosing (cutting) links between nodes from different partitions. We call such nodes that appear redundantly in multiple partitions bridge nodes.

5. The PB-tree Framework

This section introduces the PB-tree, including both index construction and kNN query processing . The data to be indexed by the PB-tree are SOs that are located on the road network. Rather than considering the unconstrained Euclidean space, our index partitions the network space considering also an existing SO density. The number of SOs on each road links is identified, which is the basis of performing hierarchical network partitioning to balance the number of SOs for each partition. To facilitate hierarchical network partitioning, the road network is converted to a line graph in which nodes become edges and vice versa.

The line graph is divided into a series of sub-graphs. The idea of partitioning the line graph instead of the directly partitioning the road network is a traditional partitioning of the road network would discard edges between partitions. Using the line graph partitions and mapping them back into the road network, we obtain partitions that retain all information (vertices and edges) by potentially duplicating vertices that are adjacent to edges from different partitions. More details and an example of a the line graph transformation, network partitioning on the line graph, and mapping back o the network can be found in [26]. Network nodes that appear in different partitions are denoted as *bridge nodes*. The resulting hierarchical partitioning yields a tree structure of partitions that are connected via bridge nodes. We store distances of bridge nodes to other tree nodes in a distance matrix.

A formal definition of bridge nodes and distance matrices which we leverage for our PB-tree are described in Section 5.1. Based on these definitions, we present our PB-tree construction algorithm in Section 5.2. In Section 5.3, we present our algorithm for kNN query processing based on the PB-tree. Intuitively, our algorithm traverses the PB-tree and the underlying road network by combining both "top-down", "bottom-up", and adjacent extension methods that exploit bridge vertices. This algorithm requires an efficient solution to compute the shortest network distance between two vertices which is presented in detail in [26]. Finally, Section 6 assesses the query performance and varying parameter settings. The above parts are summarized in Figure 2.

5.1. PB-tree Index Definition

This section presents definitions that are used in the remainder of the paper.

Definition 6 (PB-tree). *A PB-tree is a non-overlapping search tree that satisfies the following properties:*

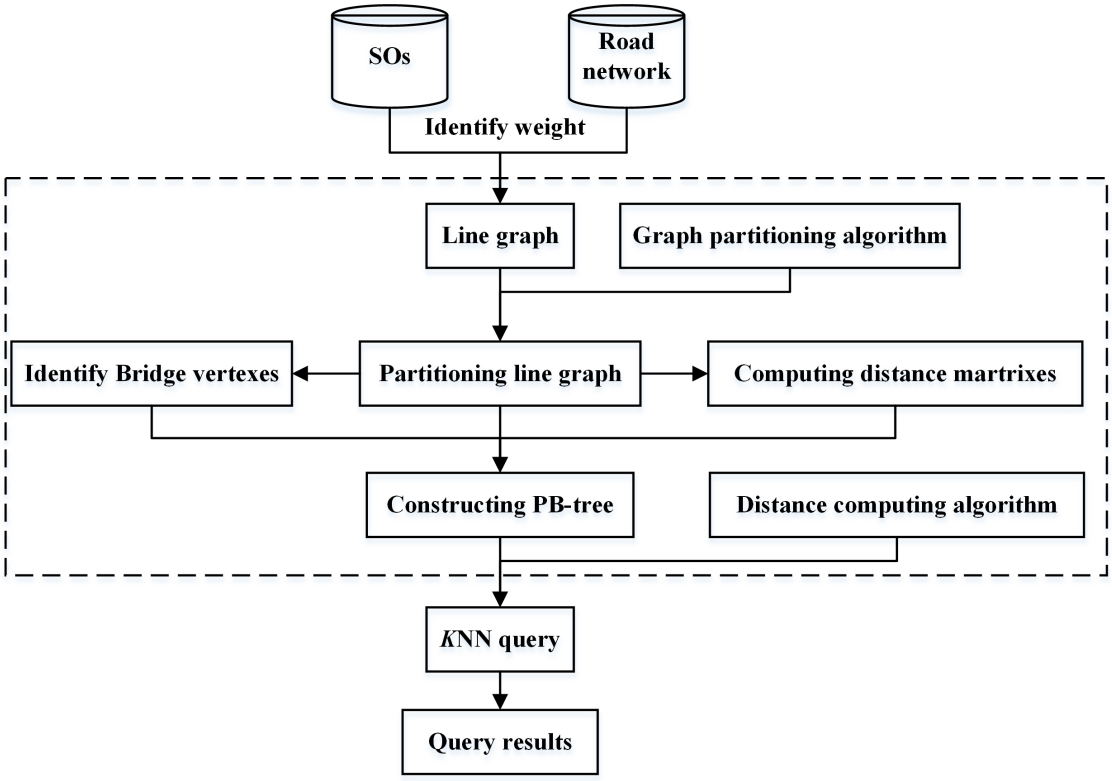


Figure 2: Flow diagram of the proposed approach

(1) Each tree node in the PB-tree is corresponding to a sub-network. The root node covers the entire road network G. The relationships between sub-networks and parent-network are captured in the PB-tree hierarchy;

(2) Each non-leaf node has between m and M child nodes;

(3) Each lowest active node contains at least one SO;

(4) Each leaf node covers at least one network link, and all leaf nodes are at the same level;

(5) Each tree node maintains a distance matrix for bridge nodes.

Definition 7 (Bridge node). *Given a road network* $G = \langle V, E \rangle$*, its sub-networks* $subG_{ik}$ *and* $subG_{ij}$*, a network node is called a bridge node if it is part of* $subG_{ik}$ *and* $subG_{ij}$*. The bridge nodes of* $subG_{ij}$ *are denoted by* $B\left(subG_{ij}\right)$*. Therefore,* $B\left(subG_{ik}\right) \cap B\left(subG_{ij}\right) \neq \emptyset$*; for example, the network nodes of the red solid circles in Figure 1 represent the bridge nodes* $B\left(subG_{11}\right) = \left\{v_2, v_9\right\}$ *and* $B\left(subG_{10}\right) = \left\{v_2, v_9\right\}$*.*

Definition 8 (Union-bridges). *Given a sub-network* $subG_{ij}$ *of G, a set of bridge nodes of all its sub-networks is referred to as union-bridge nodes. The sets of union-bridges of non-bottom partitions are a union of all bridge nodes of its sub-networks. Moreover, the union-bridges of a bottom sub-network are equal to its bridge nodes. The set of union-bridges is denoted by* $UB\left(\mathrm{sub}\, G_{ij}\right)$*. Examples in Figure 1 are* $UB\left(subG_{10}\right) = \left\{v_2, v_4, v_9\right\}$; $UB\left(subG_{11}\right) = \left\{v_2, v_9, v_{12}\right\}$; $UB\left(subG_{21}\right) = \left\{v_4, v_9\right\}$; $UB\left(subG_{20}\right) = \left\{v_2, v_4\right\}$.

Definition 9 (Active and Inactive network nodes). *Given an SO on link* e_i *which comprises the nodes* v_i *and* v_j*, if the closest node is* v_i*, we define* v_i *as the active network node. However, if none of the SOs node* v_j *as their nearest node, it is considered an inactive network node. The active and inactive network node are denoted as* AV *and* IAV*, respectively.*

Definition 10 (Lowest active node). *Given a tree node* TN*, if one of its child nodes does not contain any SO, the tree node* TN *is called the lowest active node. The lowest active nodes are denoted as* LAN*.*

Definition 11 (Distance matrices). *The columns/rows of the distance matrix of non-leaf nodes are its union-bridge nodes, and the value of each entry is the shortest network distance between the two bridge nodes. In the distance matrix of leaf nodes, the rows are all bridge nodes and active network nodes, and the columns are its all network nodes. The value of each entry is the shortest network distance between the bridge and active network nodes and network nodes. If the lowest active node is not leaf node, the distance matrix of this node also records the network distance between its bridge nodes and its active network nodes. Figure 4 gives a distance matrix for the example of Figure 1. For simplicity, the length of each network link is set to 1.*

5.2. PB-tree construction

The PB-tree is a balanced search tree with each partition having a similar number of spatial objects. Thus, the PB-tree adapts to non-uniform distributions of spatial objects on a road network. Figure 3 shows the basic components and structure of the PB-tree. The PB-tree consists of a hierarchy of tree node components, the bridge component, the union-bridge component, the active network nodes component, and the distance matrix component. To efficiently compute the shortest network distances (SNDs) between a query point and the SOs, the distance matrices for each tree node are introduced in Definition 11. Figure 4 shows a PB-tree example for the partitions of Figure 1. The bridge nodes of each tree node are shown under the tree nodes and the distance matrix of each node is displayed next to the tree node.

A sub-network $subG_{ij}$ is called a parent-network of sub-network $subG_{(i+1)k}$, if $subG_{ij}$ is a super-network of $subG_{(i+1)k}$. Therefore, the $subG_{(i+1)k}$ is called a child-network of $subG_{ij}$. The PB-tree construction is based on hierarchical graph partitions. First, the entire road network is taken in at the root level. The sub-network $subG_{ij}$ is a child of the root node. This process is repeated until the sub-network has no more sub-networks, at which point it will become a leaf-node. When the hierarchy of tree nodes is constructed, the bridge nodes of the union-bridge nodes of each tree node are identified; the active network nodes of each leaf node are marked; and the lowest active nodes are found. The distance matrices of each tree node are pre-computed and stored.

5.3. *K*NN query algorithm

Based on the structure of the PB-tree, two important characteristics of PB-tree form the foundation of *K*NN query approach. The bridge node of a parent partition is a bridge node of its child partitions. Since the bridge node connects different partitions at each level, its child partitions should contain these bridge node. More importantly, these bridge nodes in the child partitions have the same function as the bridge node in the parent partition. For example, as shown in Figure 1, $subG_{11}$ is parent partition of $subG_{22}$ and $subG_{23}$, $B(subG_{11}) = \{v_2, v_9\}$, $B(subG_{22}) = \{v_2, v_{12}\}$, $B(subG_{23}) = \{v_9, v_{12}\}$.

Given a tree node $subG_{ij}$ that contains a network node v_k, $subG_{(i-1)k}$ is a parent node of $subG_{ij}$, the SND between v_k and $B(subG_{ij})$ is less than or equal that of $B(subG_{(i-1)k})$, as shown in Eq 1.

$$\min \mathrm{SND}\left(\mathrm{v_k}, \mathrm{B}\left(\mathrm{sub\,G_{ij}}\right)\right) \leq \min \mathrm{SND}\left(\mathrm{v_k}, \mathrm{B}\left(\mathrm{sub\,G_{(i-1)k}}\right)\right) \tag{1}$$

Where $\min \mathrm{SND}\left(v_k, B\left(\mathrm{sub}\, G_{ij}\right)\right)$ represents the minimum distances between v_k and $B(subG_{ij})$, $\min \mathrm{SND}\left(v_k, B\left(subG_{(i-1)k}\right)\right)$ has the same meaning for $subG_{(i-1)k}$.

Since the space of a parent node contains that of child nodes, the path from v_k to $B\left(subG\left(_{i-1}\right)_k\right)$ should pass one of the $B\left(subG_{ij}\right)$, besides, the distances between network nodes are > 0, thus Eq 1 always holds. For instance, in Figure 1, $\min \mathrm{SND}\left(v_5, B\left(subG_{21}\right)\right) = 1$, $\min \mathrm{SND}\left(v_5, B\left(subG_{10}\right)\right) = 3$.

Given a query $q = \langle v_q, k \rangle$, the kNN query returns the k closest SOs to v_q. For simplicity and ease of presentation, given that the distances between SOs and AVs are much smaller than the distances between network nodes, thus in our algorithm, we assume that both query points and SOs are at network nodes in our algorithm. If the query point falls on a link, we apply the nearest network node to the SO to perform a *K*NN query.

Algorithm 1 shows the skeleton of the PB-tree used to search kNN SOs. First, we initialize the result set R and the priority queue Q as empty and all leaf nodes become the lowest active nodes. Second, to locate the leaf node that contains the query point v_q and identify its lowest active node, the distances between AVs, bridge node in LAN $\left(v_q\right)$ and v_q are computed. These AVs and bridge nodes are enqueued in Q. Next, if the size of R is less than k and Q is not null, the first element e of Q is dequeued, the network node e is checked if is an active network node, the SOs of e are put into R. When e is a bridge node, the lowest active nodes LANs (e) that contain e are identified. For each AV that belongs to LANs(e), the distances between AVs in LANs(e) and v_q are computed. Similarly, the bridge nodes in LANs(e) are also computed. This process is repeated until kNN SOs are found. Finally, all valid SOs are returned.

SND calculation is a critical part of the *K*NN query algorithm and has a significant impact on query performance. According to the relationship between v_i and v_j, we can consider the following two types of computation: (i) v_i and v_j are in different leaf nodes: the dynamic programming algorithm can be applied to effectively compute the SND based on distance matrices. The main process is as follows: first, to identify the lowest common ancestor of v_i and v_j; then, the all bridge nodes of all involved tree node form v_i/ v_j to their lowest common ancestor are combined, the shortest network distance between v_i and v_j can be obtained. Since we precompute and store some distance pairs in variously assembling these pairs and rapidly get the SND; and (ii) v_i and v_j are in the same leaf node: Dijkstra is used to compute the shortest path distance efficiently given the small graph size. Moreover, if one of v_i and v_j is an active network node or a bridge node, the distance can be obtained from the distance matrix. For more details, the reader is referred to [20, 26].

Algorithm 1 *K*NN query

Require: v_q : query point; K: the number of nearest neighbors

Ensure: R : kNN SOs to v_q

Notation: lowest active node (LAN), active network node(AV), tree node(TN);

1: Initialize result set $R = \emptyset$, priority queue $Q = \emptyset$;
2: Find the $LANs$;
3: **if** the $LANs$ do not exist **then**
4: The leaf nodes are $LANs$;
5: **end if**

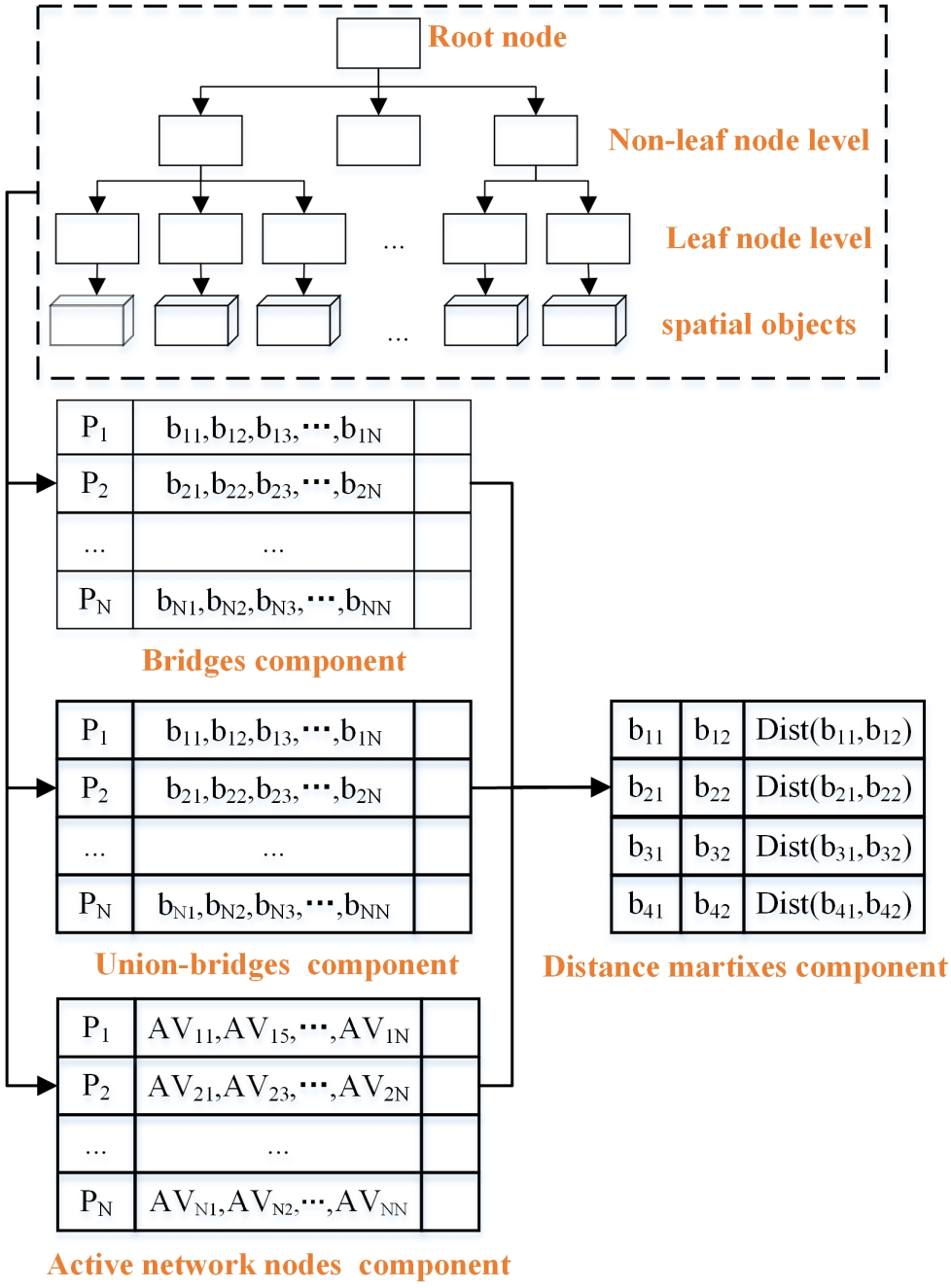


Figure 3: PB-tree structure

Locate leaf node that contains v_q;
Find the LAN that contains v_q;
foreach $AV \in \text{LAN}(v_q)$ **do**
    Q.Enqueue(AV, SND(v_q, AV))
end for
foreach bridge $b \in \text{LAN}(v_q)$ **do**
    Q.Enqueue(b, SND(v_q, b))
end for
while (R.size() $< K$ and $Q \neq \emptyset$) **do**
    $(e, dis) \leftarrow Q$.Dequeue();
    if e is AV **then**
        Insert e. SOs into R
    end if
    if e is *bridge* **then**
        Find the other $LANs(e)$ that contain e;
        foreach tree node $TN \in$ LANs (e) **do**
            foreach bridge node b $\in TN$ **do**
                Q.Enqueue (b, SND (v_q, b))
            end for
            foreach AV $\in TN$ **do**
                Q.Enqueue (AV, SND (v_q, AV))
            end for
        end for

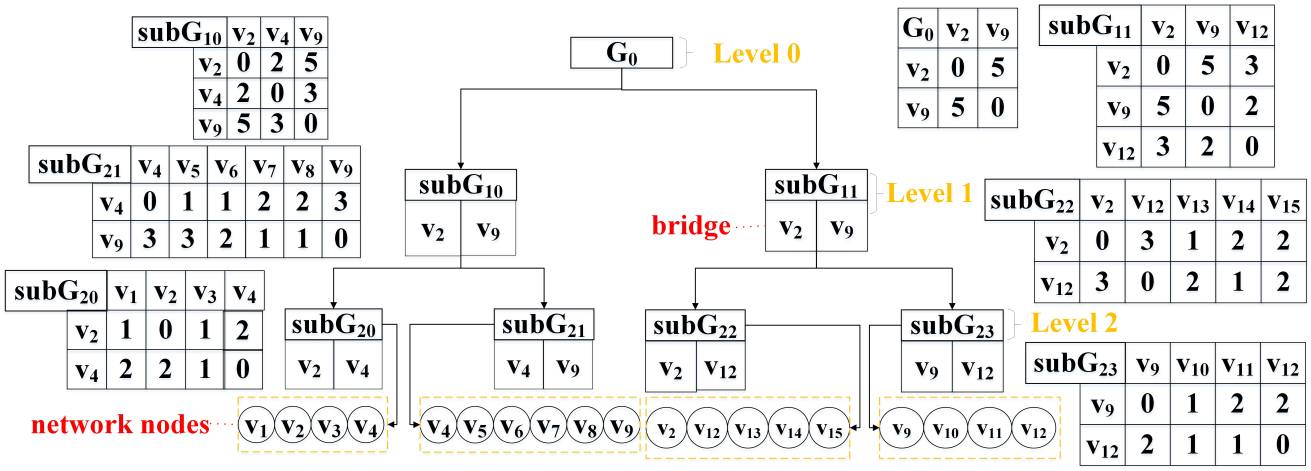


Figure 4: PB-tree example

Table 1
Datasets

Data	Description	# network nodes	# links
OL	Oldenburg(undirected)	6105	7034
SJ	San Joaquin(undirected)	18496	24123
SF	San Francisco(undirected)	175343	223606
SZ	Shenzhen(undirected)	36968	50149
BJ	Beijing(undirected)	83059	11014

29: **end if**
30: **end while**
31: **return** R

6. Experimental Evaluation and Discussion

In this section, we experimentally evaluate our proposed method using synthetic datasets by comparing our approach to state-of-the-art and classical KNN methods.

6.1. Data sets and settings

Five real-world road networks, namely, Oldenburg (Germany), San Joaquin (America), San Francisco (America), Shenzhen (China), and Beijing (China), are used to test the performance of the PB-tree. These road networks consist of streets, roads, and beltways. Moreover, they have different sizes from 7,000 links to 220,000 links, and various spatial patterns. The "Brinkhoff" generator [30] is used to generate SO datasets for those road networks. The main reason for using the synthetic data sets is that the size and distributions of SOs are flexible. The dataset characteristics are shown in Table 1. To better evaluate the KNN query performance, we compare our PB-tree with two state-of-the-art methods, G-tree[20] and G*tree[21], and the classical approach, INE[6]. The implementations of G-Tree and G*-tree are provided by the authors, and INE is implemented by ourselves. For the PB-tree, the default fanout is 4, since the G-tree and G*-tree have the optimal performance with this fanout. To evaluate the efficiency of the KNN query, we randomly chose 20 query points to test. For SOs, we randomly generate the 0.002,0.02, 0.2, and 2 of network links as the datasets (default is 0.2). For k, we use 1, 5, 10, 20, and 50.

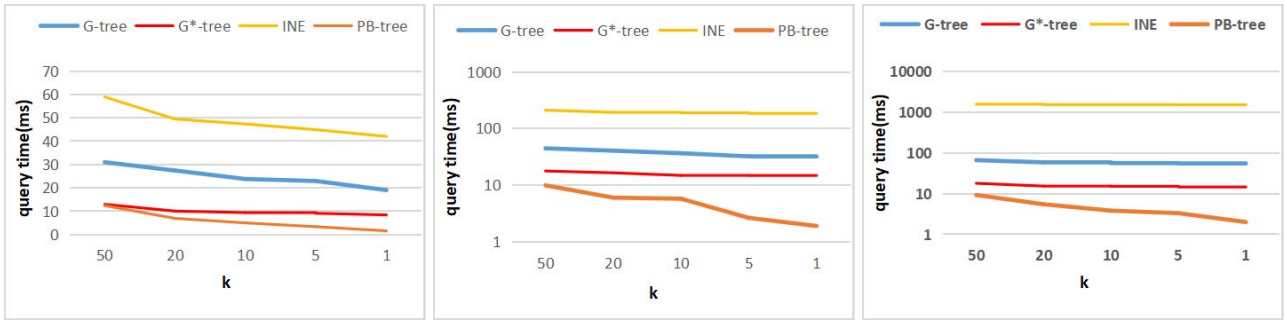
All experiments are conducted in C++ and run on an Intel i7, 2.6 GHz CPU, 16 GB RAM, and Windows 64-bit operating system.

6.2. Evaluation on query efficiency

The KNN search performance of PB-tree is tested for different ks. The number of test SOs is about 20 percent of the network links The SOs are distributed at random on the road network, which can be uniform, random, or clustered. The performance indexes include the query time and the number of computed network nodes. Since the computational bottleneck for KNN is distance calculations, it is essential to compare the number of computed network nodes.

Query time by varying k: The results for query times and different datasets are shown in Figure 5. The PB-tree always outperforms G-tree and INE. Moreover, the PB-tree is better than G*-tree when k is less than 40, and G*-tree and PB-tree have similar performance when k is 50. The main reasons are that the (1) PB-tree partitions the road network based on the network topology and SOs distributions, it not only ensures the spatial proximity, but each partition has the same number of SOs. Therefore, we can quickly find the valid SOs by accessing fewer tree nodes; (2) The significant step of the query process is based on the adjacent tree nodes; we combine expanding the adjacent areas to quickly find the nearest nodes.

The number of computed network nodes by varying k is shown in Figure 6. PB-tree requires a fewer computed network nodes than G-tree and G*tree because of three main factors. First, when the distribution of SOs is uneven, each partition has different network nodes, the tree nodes of dense



(a) OL (b) SJ (c) SF

(d) SZ (e) BJ

Figure 5: Performance comparison on query time

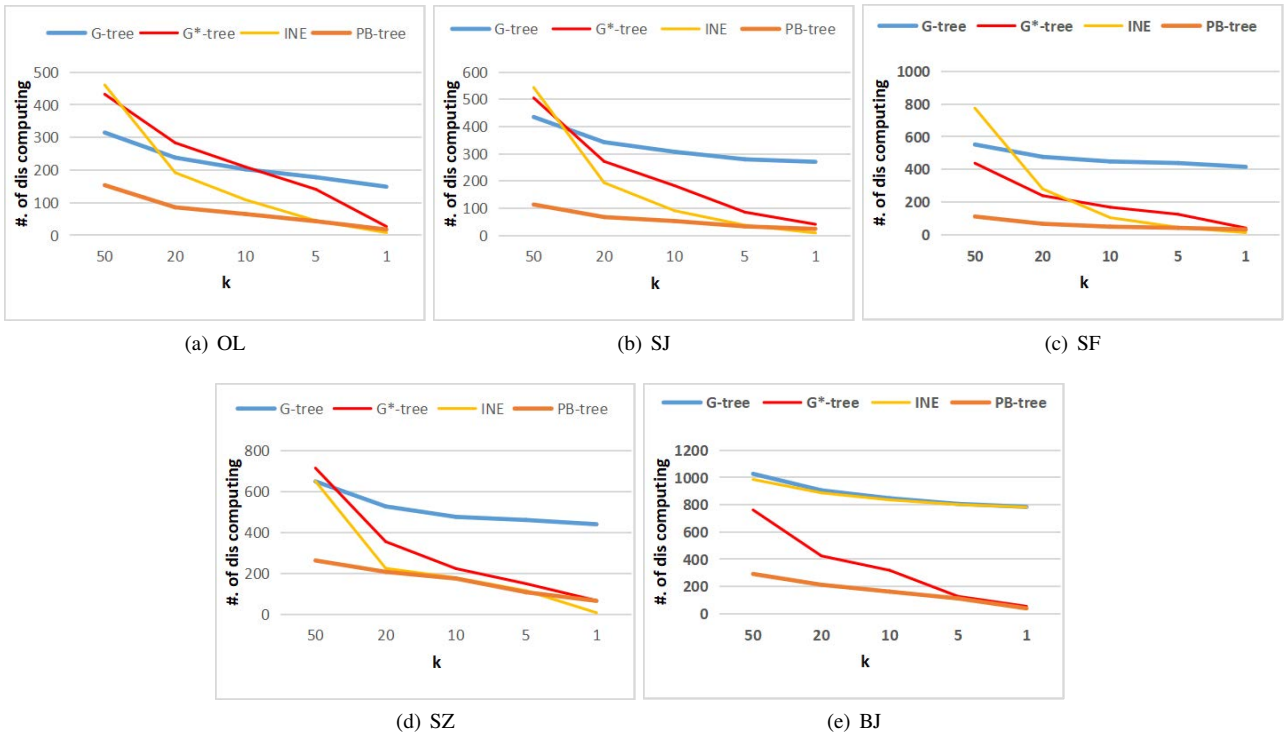
areas have fewer network nodes, thus the bridge nodes are also fewer. Second, the bridge nodes are at least shared by two nodes of the PB-tree, thus fewer bridge nodes are computed. Third, the PB-tree generally does not need to compute some bridge nodes of non-leaf nodes that do not contain valid SOs. PB-tree achieves comparable performance with INE when k is 1. The main reason is that INE uses the Dijkstra search algorithm to find valid objects, and PB-tree may involve brother leaf nodes. However, we can find that PB-tree achieves excellent performance as the k increases.

We evaluated the performance for fanouts of 2, 4, 8, 16 in Figure 7. The number of test SOs is about 20% of the total number of network links, and the road network is SJ (San Joaquin). Twenty random queries are used to compare an average value. We set $k = 10$ and the number of partitions is 256. The results show that the query time and the number of computed network nodes become larger as fanout increases. The reasons are simply that (1) when the fanout is larger, a tree node has more child nodes, and the height of the PB-tree will decrease. (2) when the query is performed, although the bridge nodes of fewer non-leafs are computed, the bridge nodes of brother leaf nodes will be involved. Also, index construction takes more time as the fanout increases. Third, as the fanout is increased, the number of bridge nodes decreases. For the second and third characteristics, the primary reasons are that (1) there are two main parts to indexing construction, for index construction most of the time is spent calculating the distance matrices. When the fanout is larger, the union bridges of non-leaf nodes are larger, thus, it will take more time. (2) regarding the number of bridge nodes, a smaller fanout increases the height of the PB-tree, the number of tree nodes increases, and so does the number of bridge nodes. Besides, a larger fanout will produce more bridge nodes for leaf nodes.

Figure 8 presents the results of different SOs dataset sizes, which are from 0.2% to 2 of the number of network links. We test on the SJ road network by default and set $k = 10$, $f = 4$. The results show that (1) for query time, PB-tree consistently performs the best when SOs are different numbers. (2) For the number of computed network nodes, the PB-tree always outperforms the G-tree and G*-tree. Compared with INE, PB-tree achieves comparable performance with INE when SOs dataset size is twice the number of network links, and in other cases, PB-tree consistently demonstrates the best performance.

6.3. Summary of the experiments

The first part of the evaluation focused on testing the performance indexes for query time and the number of computed network nodes for different k. The experimental results show that the PB-tree outperforms all the others for different road networks and number of SOs. The second part of the evaluation addresses fanout and we analyze fanout relates to query performances. The third part primarily evaluates the various SOs dataset sizes and distributions. The experimental results show that our method outperforms the state-of-the-art methods and classical approach. From all results, we can see that the PB-tree has good robustness and scalability.



(a) OL (b) SJ (c) SF

(d) SZ (e) BJ

Figure 6: performance comparison on the number of computed network nodes

7. Conclusion and future works

This work proposes a novel access method called PB-tree in support of kNN queries on road networks. The index is based on road network partitioning considering the spatial objects (SOs) distribution in combination with the network topology, which can be suitable for different cases. This work also introduces a kNN query algorithm that leverages the hierarchical nature of the PB-tree and provides superior query performance as established in experiments using a range of road networks and varying numbers of spatial objects created by a data generator. The experimental results show that the PB-tree outperforms the state-of-the-art methods and the classical approach.

Directions for future work are as follows. An important aspect will be to create an index that considers updates to the SOs as well as the road network. Also, we are working on leveraging the PB-tree for other kinds of queries, such as continuous KNN and reverse NN queries. It will also be interesting to study how to use our method for real-time data, such as moving objects streams.

Data Availability Statement

The road network and experimental datasets can be obtained from the "Brinkhoff" generator (https://iapg.jade-hs.de/personen/brinkhoff/generator/).

Acknowledgement(s)

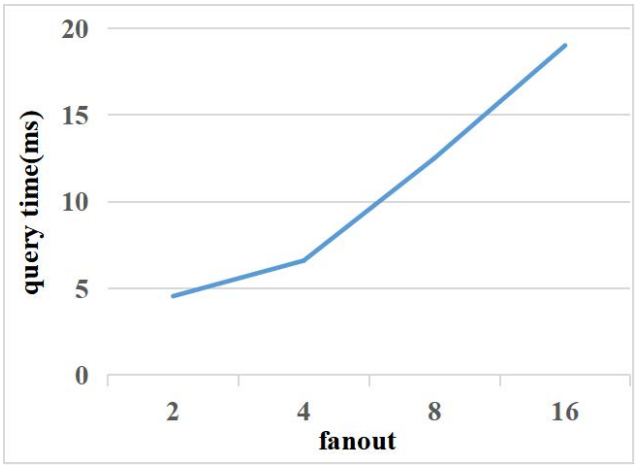
We appreciate the detailed suggestions and constructive comments from the editor and the anonymous reviewers.

Disclosure statement

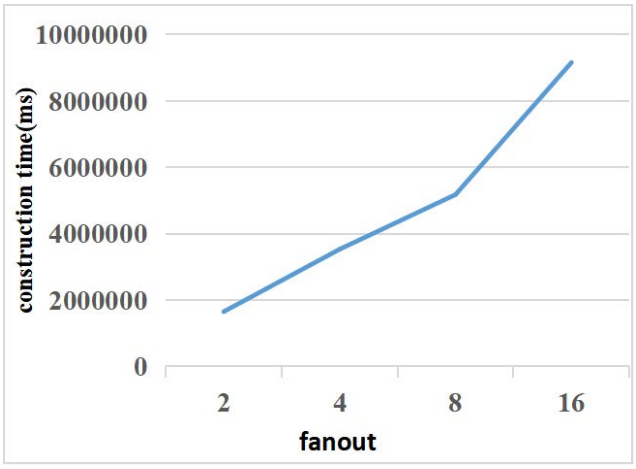
The author has declared no conflicts of interest.

References

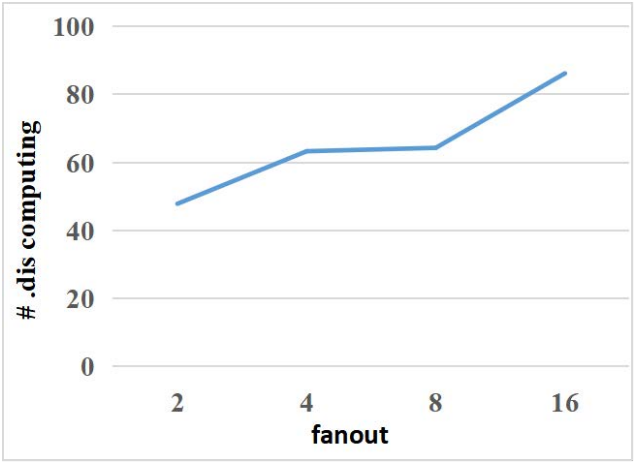
[1] I. S. Popa, K. Zeitouni, V. Oria, D. Barth, S. Vial, Indexing in-network trajectory flows, The VLDB Journal 20 (2011) 643–669.
[2] H. Huang, G. Gartner, Current trends and challenges in location-based services, 2018.
[3] J.-S. Kim, H. Jin, H. Kavak, O. C. Rouly, A. Crooks, D. Pfoser, C. Wenk, A. Züfle, Location-based social network data generation based on patterns of life, in: 2020 21st IEEE International Conference on Mobile Data Management (MDM), IEEE, 2020, pp. 158–167.
[4] T. Abeywickrama, M. A. Cheema, D. Taniar, K-nearest neighbors on road networks: a journey in experimentation and in-memory implementation, arXiv preprint arXiv:1601.01549 (2016).
[5] B. Borhanuddin, B. Solemon, Spatial network k-nearest neighbor: A survey and future directives, Journal of Telecommunication, Electronic and Computer Engineering (JTEC) 9 (2017) 1–6.
[6] D. Papadias, J. Zhang, N. Mamoulis, Y. Tao, Query processing in spatial network databases, in: Proceedings 2003 VLDB Conference, Elsevier, 2003, pp. 802–813.
[7] X. Huang, C. S. Jensen, S. Šaltenis, Multiple k nearest neighbor query processing in spatial network databases, in: East European Conference on Advances in Databases and Information Systems, Springer, 2006, pp. 266–281.
[8] M. Safar, K nearest neighbor search in navigation systems, Mobile Information Systems 1 (2005) 207–224.



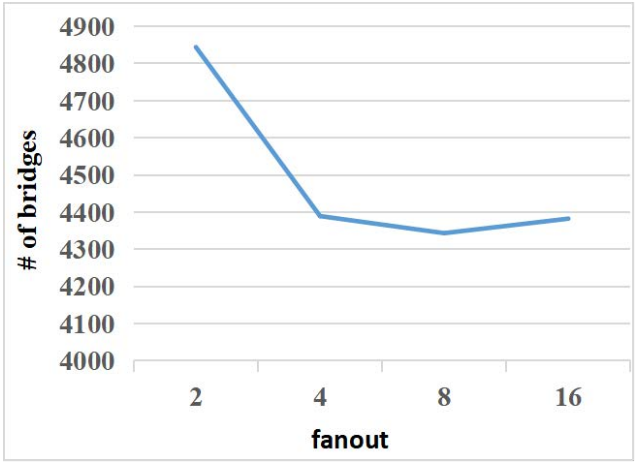
(a) query time



(b) construction time



(c) the number of computed network nodes



(d) the number of bridges

Figure 7: Evaluation on parameter(fanout)

[9] X. Huang, C. S. Jensen, S. Šaltenis, The islands approach to nearest neighbor querying in spatial networks, in: International Symposium on Spatial and Temporal Databases, Springer, 2005, pp. 73–90.

[10] H.-J. Cho, C.-W. Chung, An efficient and scalable approach to cnn queries in a road network, in: International Conference on VLDB, volume 2, International Conference on VLDB, 2005, pp. 865–876.

[11] H. Hu, D. L. Lee, J. Xu, Fast nearest neighbor search on road networks, in: International Conference on Extending Database Technology, Springer, 2006, pp. 186–203.

[12] X. Huang, C. S. Jensen, H. Lu, S. Šaltenis, S-grid: A versatile approach to efficient query processing in spatial networks, in: International Symposium on Spatial and Temporal Databases, Springer, 2007, pp. 93–111.

[13] K. C. Lee, W.-C. Lee, B. Zheng, Fast object search on road networks, in: Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, 2009, pp. 1018–1029.

[14] K. C. Lee, W.-C. Lee, B. Zheng, Y. Tian, Road: A new spatial object search framework for road networks, IEEE transactions on knowledge and data engineering 24 (2010) 547–560.

[15] V. T. De Almeida, R. H. Güting, Using dijkstra's algorithm to incrementally find the k-nearest neighbors in spatial network databases, in: Proceedings of the 2006 ACM symposium on Applied computing, 2006, pp. 58–62.

[16] J. Sankaranarayanan, H. Alborzi, H. Samet, Efficient query processing on spatial networks, in: Proceedings of the 13th annual ACM international workshop on Geographic information systems, 2005, pp. 200–209.

[17] H. Samet, J. Sankaranarayanan, H. Alborzi, Scalable network distance browsing in spatial databases, in: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, 2008, pp. 43–54.

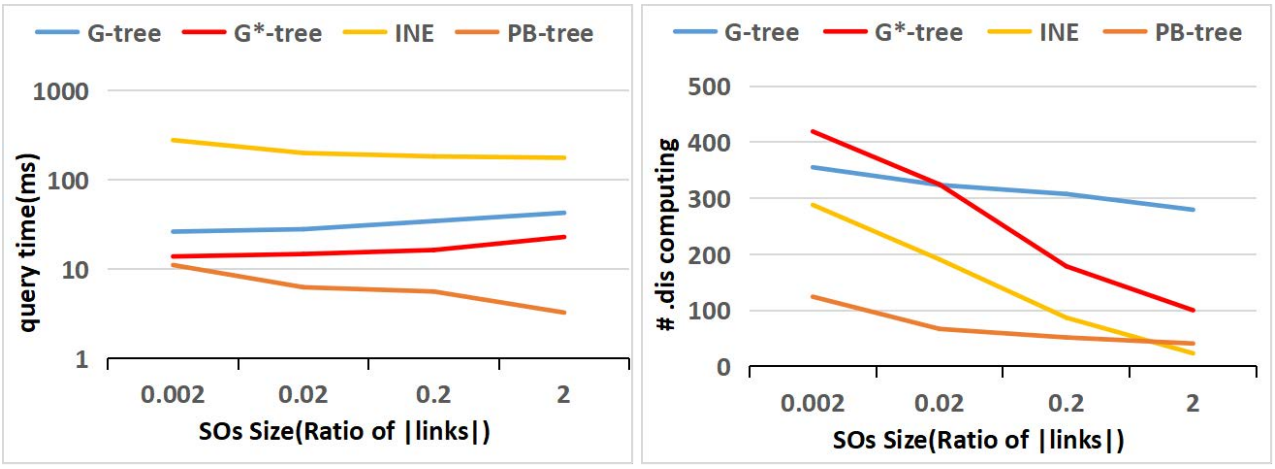
[18] H. Hu, D. L. Lee, V. C. Lee, Distance indexing on road networks, in: Proceedings of the 32nd international conference on Very large data bases, 2006, pp. 894–905.

[19] R. Zhong, G. Li, K.-L. Tan, L. Zhou, G-tree: An efficient index for knn search on road networks, in: Proceedings of the 22nd ACM international conference on Information & Knowledge Management, 2013, pp. 39–48.

[20] R. Zhong, G. Li, K.-L. Tan, L. Zhou, Z. Gong, G-tree: An efficient and scalable index for spatial search on road networks, IEEE Transactions on Knowledge and Data Engineering 27 (2015) 2175–2189.

[21] Z. Li, L. Chen, Y. Wang, G*-tree: An efficient spatial index on road networks, in: 2019 IEEE 35th International Conference on Data Engineering (ICDE), IEEE, 2019, pp. 268–279.

[22] X. Teng, J. Yang, J.-S. Kim, G. Trajcevski, A. Züfle, M. A. Nascimento, Fine-grained diversification of proximity constrained queries on road networks, in: Proceedings of the 16th International Symposium on Spatial and Temporal Databases, 2019, pp. 51–60.



(a) query time

(b) the number of computed network nodes

Figure 8: Evaluation on varying numbers of SOs

[23] X. Teng, G. Trajcevski, A. Züfle, Semantically diverse paths with range and origin constraints, in: Proceedings of the 29th International Conference on Advances in Geographic Information Systems, 2021, pp. 375–378.

[24] M. Kolahdouzan, C. Shahabi, Voronoi-based k nearest neighbor search for spatial network databases, in: Proceedings of the Thirtieth international conference on Very large data bases-Volume 30, 2004, pp. 840–851.

[25] J.-L. BAO, B. Wang, X. C. YANG, H. jie ZHU, Nearest neighbor query in road networks, Journal of Software 29 (2018) 642–662.

[26] X. Min, D. Pfoser, A. Züfle, Y. Sheng, A hierarchical spatial network index for arbitrarily distributed spatial objects, ISPRS International Journal of Geo-Information 10 (2021) 814.

[27] I. Abraham, D. Delling, A. Fiat, A. V. Goldberg, R. F. Werneck, Hldb: Location-based services in databases, in: Proceedings of the 20th International Conference on Advances in Geographic Information Systems, 2012, pp. 339–348.

[28] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM Journal on scientific Computing 20 (1998) 359–392.

[29] T. Anwar, C. Liu, H. L. Vu, C. Leckie, Partitioning road networks using density peak graphs: Efficiency vs. accuracy, Information Systems 64 (2017) 22–40.

[30] T. Brinkhoff, A framework for generating network-based moving objects, Geoinformatica 6 (2002) 153–180.