An Efficient Edge-Based Index for Processing Collective Spatial Keyword Query on Road Networks

Ye-In Chang[1], Jun-Hong Shen [2,*], and Sheng-Yang Lin[1]

[1] Department of Computer Science and Engineering, National Sun Yat-Sen University,
Kaohsiung 804, Taiwan
changyi@cse.nsysu.edu.tw, allen9910@hotmail.com
[2] Department of Information Management, National United University, Miaoli 360301,
Taiwan
shenjh@nuu.edu.tw

Abstract. Spatial keyword queries find extensive applications in geographic information systems like Facebook and Instagram. The collective spatial keyword query (CSKQ) plays a crucial role among the various types of queries. This query aims to retrieve a set of Points of Interest (POIs) that collectively cover the specified keywords while being in proximity to both the query location and other objects. To evaluate the spatial cost of a set of POIs in CSKQ, we introduce the Edge-Based Collective Nine-Area Tree Index (EBCNA). By incorporating edge information and POIs into the NA-tree structure, the EBCNA offers a comprehensive solution. All edge information, including POIs, is stored in the leaf nodes, and each edge links to its adjacent edges via pointers. This design enables direct retrieval of edge information and POIs without repeatedly trailing back to the root node. Through a comparative analysis, we have demonstrated our proposed method's superior performance compared to the existing one.

Keywords: Collective spatial keyword query, road network, spatial database.

1 Introduction

With the growing popularization of geo-positioning technologies and geo-location services, many spatial-textual objects are used in many applications (e.g., Twitter and Facebook). Those geo-tagging services combine the location information with textual descriptions. Through the aforementioned development, some services efficiently process spatial keywords query (SKQ) that concern textual relevance and spatial closeness between POIs (Points of Interest) and the query location.

The spatial road network belongs to the category of geographic graphs, wherein nodes are situated along road networks [1–3]. A road network can be represented as a graph comprising a collection of vertices (or nodes), edges, and weights (or network distances) assigned to these edges. In this context, each vertex signifies an endpoint or a road intersection within the network, while each edge represents a road segment.

Furthermore, the weight associated with each edge corresponds to the respective road segment's length (or network distance). Additionally, when considering a set of spatial-textual objects, namely points of interest (POIs) located on the road network, each POI possesses both a spatial location and a textual description.

The collective spatial keyword query (CSKQ) is an essential variant of spatial keyword queries. The purpose of CSKQ is to find a set of objects that collectively incorporate the query keywords, and those objects are near the query location and close to each other object [4-5]. Namely, we must evaluate the keyword matching and the spatial proximity of query location and objects. For instance, we issue a query with keywords $\{School, Park, Restaurant\}$, and we have an object $o1$ with keyword *{School}* and an object $o2$ with keywords $\{Park, Restaurant\}$. An example of a keyword matching approach of CSKQ is $\{o1, o2\}$, which collectively covers the query keywords. Besides the keyword-matching approach, spatial proximity also needs to be concerned.

In the literature, Gao et al. introduced a renowned algorithm for addressing the collective spatial keyword query (CSKQ), employing the CCAM index structure for storing points of interest (POIs) on the road network [4]. Building upon the CCAM index structure [6], the authors proposed algorithms to tackle the CSKQ problem. The first algorithm, Network Expansion Based (NEB), identifies the nearest objects encompassing the queried keywords relative to the query location. These objects are subsequently utilized as the result set. The closeness of the result set is evaluated using a cost function. The goal of NEB is to find an upper bound of the cost used by the other algorithms to find a better answer. Their exact algorithm, called Sliding Window (SW) algorithm, is to find the optimal result set with the lowest cost calculated by the cost function. In their proposed algorithms, upon issuing a CSKQ, the process necessitates the traversal of numerous edges. In order to retrieve the requisite edge information, an iterative search of the B+-tree structure from the root node becomes imperative. However, this recurrent search operation significantly escalates the overall search time.

Therefore, to reduce the search time, this paper presents the edge-based collective nine-area tree (EBCNA) index structure to shorten the search time in the leaf node and enhance the efficiency of collective spatial keyword query processing on road networks. By incorporating edge information and POIs into the NA-tree structure [7], the EBCNA offers a comprehensive solution. All edge information, including POIs, is stored in the leaf nodes, and each edge links to its adjacent edges via pointers. This design enables direct retrieval of edge information and POIs without repeatedly trailing back to the root node.

The rest of this paper is organized as follows. Section 2 presents the proposed algorithms. Section 3 evaluates the performance efficiency. Section 4 presents the concluding remarks of the study.

2 The Proposed Algorithms

In the proposed algorithms, we use an ECBNA (edge-based collective nine-area tree) index, a revised version of the NA-tree proposed by Chang et al. [7], to build our road network model. We proposed the basic expanding algorithm and the nearest keyword first exact algorithm to process CSKQ.

2.1 Edge-Based Collective Nine-Area Tree

We define the road networks as an undirected graph $G = (V, E)$, where V is a set of vertices and E is a set of edges. The vertex v in V is denoted by $v = (vid, pos)$, where *vid* is the vertex ID number and *pos* coordinates the vertex in 2D space. Every vertex v in V corresponds to the junctions of edges and the endpoints of edges on the road network. The edge e in E is represented as $e = (eid, v_s, v_e, length, obj)$. *eid* is the ID number of the edge. $v_s \in V$ is the start point of the edge. $v_e \in V$ is the endpoint of the edge. *length* is the length of the edge. *obj* is the spatial object which contains both spatial and textual information belonging to the edge.

An NA-tree [7] is a structure according to data location and is organized by spatial numbers. The spatial space is decomposed into four equal-sized regions, and an NA-tree might have nine children according to the decomposition of the spatial space, as shown in Figure 1. Figure 2 shows a running example of the road network. The length of the edge between two vertexes is marked in red color, and the black circle indicates an object with spatial keywords and the corresponding distance to the vertex with the lower vertex number along the edge. For example, object $o3$ is associated with the keyword c, and the distance from vertex $v2$ to $o3$ is 6.

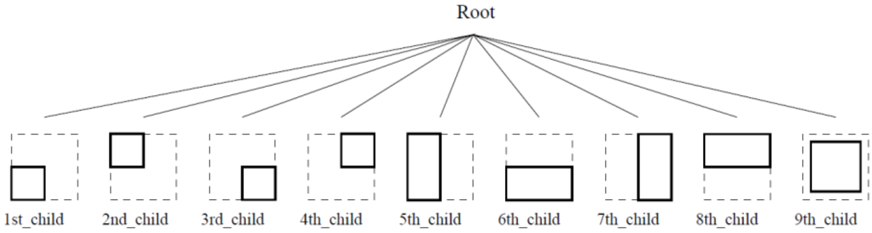


Fig. 1. NA-tree

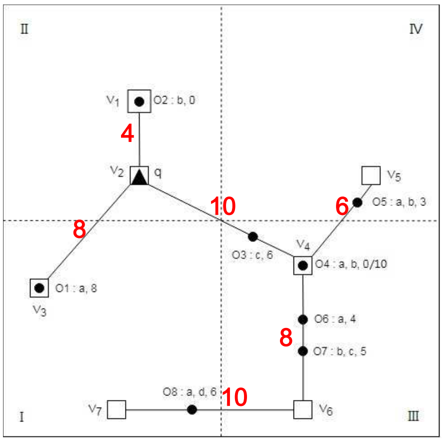
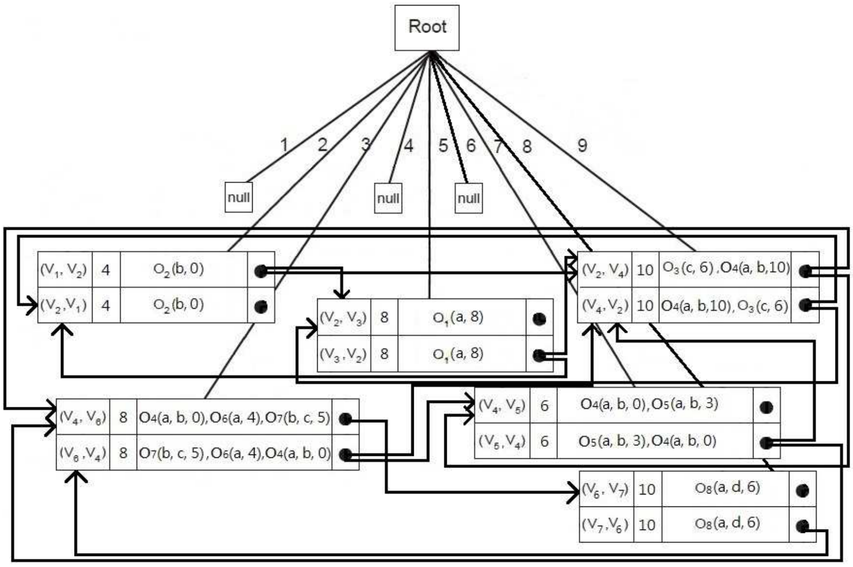


Fig. 2. A running example of the road network

Inspired by the NA-tree representation approach, we use two endpoints of a line segment to represent line segments. Figure 3 shows the corresponding EBCNA index structure. For example, the edge $v2v3$ is located at the fifth child of the NA-tree, according to Figures 1 and 2. In Figure 3, the fifth leaf node stores the information about the edge from vertex $v2$ to vertex $v3$ and the edge from vertex $v3$ to vertex $v2$. The second column indicates the length of the edge. The third column indicates the object lists along the edge. The last column contains a reference to the other leaf node that stores information about the adjacent edge connected to the endpoint of the current edge.



Root
1
2
3
4
5
6
7
8
9
null
null
null
(V1, V2) | 4 | O2(b, 0)
(V2, V1) | 4 | O2(b, 0)
(V2, V3) | 8 | O1(a, 8)
(V3, V2) | 8 | O1(a, 8)
(V2, V4) | 10 | O3(c, 6), O4(a, b,10)
(V4, V2) | 10 | O4(a, b,10), O3(c, 6)
(V4, V6) | 8 | O4(a, b, 0), O6(a, 4), O7(b, c, 5)
(V6, V4) | 8 | O7(b, c, 5), O6(a, 4), O4(a, b, 0)
(V4, V5) | 6 | O4(a, b, 0), O5(a, b, 3)
(V5, V4) | 6 | O5(a, b, 3), O4(a, b, 0)
(V6, V7) | 10 | O8(a, d, 6)
(V7, V6) | 10 | O8(a, d, 6)

Fig. 3. **The EBCNA index structure**

2.2 The Basic Expanding Algorithm

We present the basic expanding algorithm to find a result set with a reasonable cost function value and to serve as the upper bound of the following exact algorithm. The cost function is as follows [4]: Cost(R) = $(1 - \alpha) \times max_{o \in V}\, d(q, o) + \alpha \times max_{oi,oj \in V}\, d(o_i, o_j)$, where R denotes the set of the result objects. $max_{o \in V}\, d(q, o)$ represents the maximal distance between the query point and any object in R. $max_{oi,oj \in V}\, d(o_i, o_j)$ represents the maximal distance between every two of objects in R. α is a user-defined parameter between 0 and 1 to determine the importance between $max_{o \in V}\, d(q, o)$ and $max_{oi,oj \in V}\, d(o_i, o_j)$. The Cost($R$) function evaluates the spatial cost between the query point and the result set R. α is set to 0.5.

In Figure 2, a triangle on node $v2$ denotes the CSKQ query q with keywords $\{a, b, c\}$. *Min_Q* is a minimal priority queue that keeps objects in the queue based on their distances to the query location. *Find_Key* is a set that records the query keywords. Based on the EBCNA index structure, we expand the road network from query q and find the first shortest edge $v1v2$. The edge $v1v2$ has one object $o2$ containing its keyword b, and the distance = 0 from object $o2$ to the node with a smaller ID, *i.e.,* $v1$. Thus, we have *Find_Key* = $\{a, b, c\} - \{b\} = \{a, c\}$ and edge $v1v2$

is dequeued from queue *Min_Q*. We keep the same way to expand the road network and find the next shortest edge, *v*2*v*3, in queue *Min_Q*. On edge *v*2*v*3, object *o*1 with keyword *a* is found, and the distance = 8 from object *o*1 to the node with a smaller ID, *i.e., v*2. Thus, we have *Find_Key* = {*a*, *c*} - {*a*} = {*c*} and edge *v*2*v*3 is dequeued from queue *Min_Q*. Since there is still one keyword *c* in set *Find_Key*, we keep expanding the road network and find the next shortest edge *v*2*v*4 in queue *Min_Q*. On edge *v*2*v*4, object *o*3 with keyword *c* is found, and the distance = 6 from object *o*3 to the node with a smaller ID, *i.e.,* vertex *v*2. Thus, we have *Find_Key* = {*c*} - {*c*} = {}. Then, we have the result set *R* = {*o*1, *o*2, *o*3} and the function Cost(*R*) = (1-0.5) × *d*(*q*, *o*1) + 0.5 × *d*(*o*1, *o*3) = 0.5 × 8 + 0.5 × 14 = 11, which can be the upper bound of the following exact algorithm.

2.3 The Nearest Keyword First Exact Algorithm

We propose the nearest keyword first exact algorithm, *NKF*, to find the optimal result set with the lowest function cost. In the first phase, we expand the road network to identify objects containing query keywords from the query location. These objects are inserted into a minimal priority queue. We then select the first object in the queue as the new query point and employ the basic expanding algorithm to obtain the result set. The cost function of this initial result set is used as the upper bound. Subsequently, we continue to expand the road network, selecting the next object in the queue and finding a new result set. We calculate the cost function for this new result set. If the calculated cost function is lower than the cost of the basic expanding algorithm, we update the current lowest cost function. We repeat this process of expanding the road network and finding different result sets until the distance between the discovered object and the query location exceeds the current lowest cost function. In the second phase, we record the objects with the exact query keywords in the minimal priority queue for each query keyword. Next, we find the rest of the combinations and calculate their function cost. Finally, the result set with the lowest function cost becomes optimal.

For the same example in Figure 2, a CSKQ *q* is issued at vertex *v*2. Figure 4 shows the procedure of the first phase. We expand the road network starting from vertex *v*2, the query location. Initially, we trace the shortest edge, *v*1*v*2, and discover an object, *o*2, with the keyword *b* on this edge. Object *o*2 is inserted into the minimal priority queue *Min_Q*, and the remaining keywords to be found are updated as *Find_Key* = {*a*, *b*, *c*} - *b* = {*a*, *c*}. Next, we employ the basic expanding algorithm from object *o*2 to identify objects close to *o*2 and collectively contain the keywords {*a*, *c*}. Consequently, we obtain the first result set, *R* = {*o*1, *o*2, *o*3}, and compute its function cost, which is 0.5 * *d*(*q*, *o*1) + 0.5 * *d*(*o*1, *o*3) = 0.5 * 8 + 0.5 * (8 + 6) = 11 (i.e., the upper bound), as listed in Step 1 of Table 1.

Moving forward, we expand the road network again and reset *Find_Key* to {*a*, *b*, *c*}. On the next shortest edge, *v*2*v*3, we find object *o*1 with the keyword *a*. Object *o*1 is added to the *Min_Q* queue, and *Find_Key* is updated as {*a*, *b*, *c*} - *a* = {*b*, *c*}, as listed in Step 2 of Table 1. We apply the basic expanding algorithm from object *o*1 and obtain the result set {*o*1, *o*2, *o*3} with a function cost of 0.5 * *d*(*q*, *o*1) + 0.5 * *d*(*o*1, *o*3) = 0.5 * 8 + 0.5 * (8 + 6) = 11. Since the function cost is not lower than the current lowest function cost, we continue expanding the road network and reset *Find_Key* to {*a*, *b*, *c*}.

Following the same approach as above, on edge *v*2*v*4, we find object *o*3 with the keyword *c* and object *o*4 with the keywords {*a*, *b*}, as listed in Step 3 of Table 1. Initially, we insert object *o*3 into the *Min_Q* queue, and *Find_Key* is updated as {*a*, *b*, *c*} - *c* = {*a*, *b*}. Employing the basic expanding algorithm from object *o*3, we find the result set {*o*3, *o*4} with a function cost of 0.5 * *d*(*q*, *o*4) + 0.5 * *d*(*o*3, *o*4) = 5 + 2 = 7. Since the function cost of 7 is lower than the current lowest function cost of 11, we update the current lowest function cost to 7. Finally, we reset *Find_Key* to {*a*, *b*, *c*} and insert object *o*4 into the *Min_Q* queue, resulting in *Find_Key* being updated as {*a*, *b*, *c*} - {*a*, *b*} = {*c*}. In step 4, *d*(*q*, *o*4) = 10 is also not lower than the current lowest function cost = 7. The first phase terminates once the distance between the object and the query location is equal to or larger than the current lowest function cost.

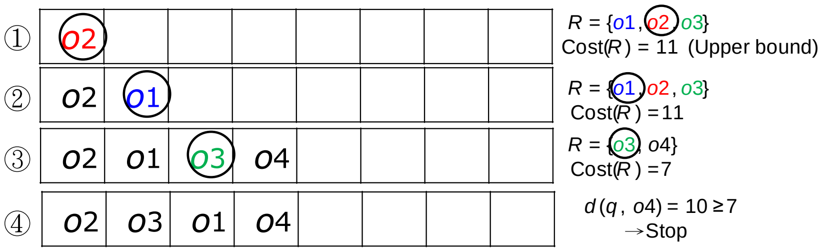


Fig. 4. The first phase

Table 1 The steps of the network expansion

Step	**Edge**	**Object**	**Keywords**	***Find_Key***	**Result Set *R***
1	*v*1*v*2	*o*2	*b*	{*a*, *c*}	{*o*1, *o*2, *o*3} Cost(*R*) = 11
2	*v*2*v*3	*o*1	*a*	{*b*, *c*}	{*o*1, *o*2, *o*3} Cost(*R*) = 11
3	*v*2*v*4	*o*3	*c*	{*a*, *b*}	{*o*3, *o*4} Cost(*R*) = 7
4	*v*2*v*4	*o*4	{*a*, *b*}	*c*	d(*q*,*o*4) = 10 ≥ 7 → Stop

In the second phase, we store objects with each query keyword in the *Min_Q* queue. Specifically, objects with keyword *a* are {*o*1, *o*4}, objects with keyword *b* are {*o*2, *o*4}, and the object with keyword *c* is {*o*3}. We then generate combinations of these objects to create a set encompassing all query keywords. Ultimately, we identify the result set {*o*3, *o*4} with the lowest function cost 7. Hence, the optimal result set is {*o*3, *o*4}.

3 Performance Evaluation

We evaluate the performance efficiency of query time for the collective spatial keyword query processing on road networks using the proposed EBCNA index structure and the CCAM index structure [4]. We perform our experiment on the Oldenburg real road network dataset, which contains 6,105 vertices and 7,035 edges at 10,000 * 10,000 (https://users.cs.utah.edu/~lifeifei/SpatialDataset.htm). Each data object contains three keywords randomly selected from the datasets with 50 keywords. The density of data objects with keywords is set to 0.3. The number of data objects is set to 2110. The threshold value of the leaf node in the B+-tree for the CCAM index structure is set to 4096 bytes. The threshold value of the leaf node in the NA-tree is set to 200. α is set to 0.5 for the cost function.

First, we compare the query time performance for the proposed expanding algorithm utilizing the EBCNA index structure and the NEB algorithm employing the CCAM index structure across varying query keywords, ranging from 3 to 5. Table 2 lists this comparison of the query time. On average, the proposed basic expanding algorithm exhibits a 64.6% improvement over the NEB algorithm. It is observed that the execution time increases as the number of query keywords increases. Notably, the performance of our basic expanding algorithm surpasses that of the NEB algorithm, primarily attributable to the distinct data structures employed for real data. This discrepancy arises because our EBCNA index structure is an edge-based indexing approach, allowing for direct linkage to other edges. Conversely, the CCAM index structure is a node-based indexing structure, necessitating repeated returns to the root node of the B+-tree during road network expansion.

Table 2 The comparison of the query time (sec) between the proposed basic expanding algorithm and the NEB algorithm

Number of Keywords	**3**	**4**	**5**
BASIC	0.076	0.077	0.083
NEB	0.206	0.225	0.230
% Improvement	63.1%	66.8%	63.9%

Second, we compare the query time performance for the proposed NKF algorithm utilizing the EBCNA index structure and the SW algorithm employing the CCAM index structure across varying query keywords, ranging from 3 to 5. Table 3 lists this comparison of the query time. On average, the proposed NKF algorithm exhibits a 94.4% improvement over the SW algorithm. The performance of the proposed NKF algorithm is better than the SW algorithm since our NKF algorithm considers not only the distance between the query location and objects but also the distance between any two objects. The method can reduce the number of objects in the minimal priority queue in the first phase. Thus, we can reduce the calculation of combinations in the second phase.

Table 3 The comparison of the query time (sec) between the proposed NFK algorithm and the SW algorithm

Number of Keywords	**3**	**4**	**5**
NFK	0.904	4.590	16.489
SW	5.555	1959.956	3619.980
% Improvement	83.7%	99.8%	99.6%

4 Conclusions

This paper introduces the EBCNA index structure to process collective spatial keyword queries (CSKQ) efficiently. This edge-based approach divides the road network into nine distinct areas and employs two spatial numbers to represent each edge. Unlike the node-based approach proposed by Gao et al. [4], the EBCNA enables efficient access to spatial objects during the expansion of the road network, eliminating the need for repeated returns to the root node. We introduce the basic expanding algorithm, which aims to find a result set with an acceptable cost function value and is the upper bound for the subsequent exact algorithm. We also propose an exact algorithm to deal with the CSKQ problem with the lowest function cost. The performance evaluation confirms the superiority of the proposed algorithms over existing ones.

Acknowledgments. This research was supported by grants MOST 105-2221-E-110-084, MOST 107-2221-E-110-064, and NSTC110-2410-H-239-019 from the National Science and Technology Council, Taiwan.

References

1. Chang, Y.-I., Tsai, M.-H., Wu, X.-L.: An edge-based algorithm for spatial query processing in real-life road networks. Int. Journal of Modeling and Optimization, vol. 5, no. 4, pp. 308–312 (2015)
2. Fang, H., Zhao, P., Sheng, V. S., Wu, J., Xu, J., Liu, A., Cui, Z.: Effective spatial keyword query processing on road networks Databases Theory and Applications, Lecture Notes in Computer Science, vol. 9093, pp. 194–206 (2015)
3. Kuang, X., Zhao, P., Sheng, V. S., Wu, J., Li, Z., Liu, G., Cui, Z.: TK-SK: textual-restricted k spatial keyword query on road networks. Databases Theory and Applications, Lecture Notes in Computer Science, vol. 9093, pp. 167–179 (2015)
4. Gao, Y., Zhao, J., Zheng, B., Chen, G.: Efficient collective spatial keyword query processing on road networks. IEEE Trans. on Intelligent Transportation Systems, vol. 17, no. 2, pp. 469–480 (2016)
5. Xue, J., Wu, C., Zhao B., Hu, Y.: Collective spatial keyword query on time dependent road networks. In: Proceedings of Tenth International Conference on Advanced Cloud and Big Data, pp. 7–12 (2022)
6. Shekhar, S., Liu, D.-R.: CCAM: a connectivity-clustered access method for networks and network computations. IEEE Trans. on Knowledge and Data Engineering, vol. 9, no. 1, pp. 102–119 (1997)
7. Chang, Y.-I., Liao, C.-H., Chen, H.-L.: NA-trees: a dynamic index for spatial data. Journal of Information Science and Engineering, vol. 19, no. 1, pp. 103–139 (2003)