Flexible Aggregate Nearest Neighbor Queries and its Keyword-Aware Variant on Road Networks

Zhongpu Chen, Bin Yao, Zhi-jie Wang, Xiaofeng Gao, Shuo Shang, Shuai Ma, and Minyi Guo, *Fellow, IEEE*

Abstract—Aggregate nearest neighbor (ANN) query in both the Euclidean space and road networks has been extensively studied, and the flexible aggregate nearest neighbor (FANN) problem further generalizes ANN by introducing an extra flexibility parameter ϕ that ranges in $(0, 1]$. In this paper, we focus on FANN on road networks, denoted as FANN$_{\mathcal{R}}$, and its keyword-aware variant, denoted as KFANN$_{\mathcal{R}}$. To solve these problems, we propose a series of universal (i.e., suitable for both *max* and *sum*) algorithms, including a Dijkstra-based algorithm that enumerates P instead of $\phi|Q|$-combinations of Q, a queue-based approach that processes data points from-near-to-far, and a framework that combines *incremental Euclidean restriction* (IER) and kNN. We also propose a specific exact solution to *max*-FANN$_{\mathcal{R}}$ and a constant-factor ratio approximate solution to *sum*-FANN$_{\mathcal{R}}$. These specific algorithms are easy to implement and can achieve excellent performance in some scenarios. Besides, we further extend this problem to top-k and multiple FANN$_{\mathcal{R}}$ (resp., KFANN$_{\mathcal{R}}$) queries. We conduct a comprehensive experimental evaluation for the proposed algorithms on real datasets to demonstrate their superior efficiency and high quality.

Index Terms—Road networks, indexing, spatial-keyword databases

1 Introduction

The aggregate nearest neighbor (ANN) query [1], [2], [3], [4], [5], [6], [7] is a classic problem that has a large number of applications (e.g., location-based services) in spatial databases. Given a set Q of query points, ANN finds out a point in a set P of data points, which has the smallest aggregate distance to *all* points in Q. The aggregate function is usually either *max* or *sum*. The ANN problem in both the Euclidean space [1], [2], [3] and road networks [4], [5], [6], [7] has been extensively studied.

We observe that it is more desirable to take *a fraction of* Q into account in many cases. More precisely, a more general query is to allow users to specify a flexibility parameter $\phi \in (0, 1]$, and the goal is to retrieve the best point from P that is the closest to *any* $\phi|Q|$ points in Q. We denote this query as the flexible aggregate nearest neighbor on road networks (FANN$_{\mathcal{R}}$). In addition, sometimes we also need to address this query in the context of keyword-aware road networks, where both the road network distance and text similarity need to be considered. We call this variant the keyword-aware flexible aggregate nearest neighbor on road networks (KFANN$_{\mathcal{R}}$).

- *Z. Chen, B. Yao (corresponding author), X. Gao and M. Guo are with Department of Computer Science and Engineering, Shanghai Jiao Tong University. Email: {chenzhongpu@, yaobin@cs., gao-xf@cs., guo-my@cs.}sjtu.edu.cn.*
- *Z.-J. Wang is with College of Computer Science, Chongqing University, also with School of Data and Computer Science, Sun Yat-Sen University. Email: cszjwang@yahoo.com.*
- *S. Shang is with Computer Science Program, King Abdullah University of Science and Technology. Email: jedi.shang@gmail.com.*
- *S. Ma is with School of Computer Science and Engineering, Beihang University. Email: mashuai@buaa.edu.cn.*

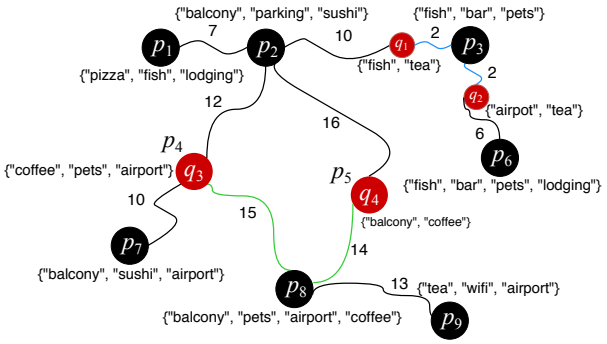


Fig. 1. A road network with textural information

Fig. 1 illustrates an example of FANN and KFANN$_{\mathcal{R}}$, where data points $P = \{p_1, p_2, \ldots, p_8, p_9\}$ (colored in black) are candidates to hold an election meeting for a company, query points $Q = \{q_1, q_2, q_3, q_4\}$ (colored in red) are shareholders with voting rights, and the aggregate function g is *sum*. Each candidate (resp., shareholder) is described by a set of keywords indicating its featured (resp., his favorite) environment. Note that p_4 and q_3, p_5 and q_4 are located at the same node, respectively. Some points are located at edges. For example, q_1 lies on (p_2, p_3). If all points in Q are considered, this can be answered by an ANN query. The result of this ANN query is p_2 with the aggregate road distance of 52 (i.e., $10+14+12+16$). However, if the meeting is legitimate as long as half of members are present, we can find a venue which minimizes the *flexible* aggregate distance to shareholders for the sake of cutting down the traveling cost, and the result of this FANN$_{\mathcal{R}}$ query is p_3 with the aggregate road distance of 4 (i.e., $2 + 2$), and the *flexible* query objects are $\{q_1, q_2\}$. On the other hand, in order to maximize these shareholders' satisfactory, it is preferred to take both the spatial and keyword information

into account. In order to simplify the presentation, we assume the desirable data point must cover all keywords of "flexible" query objects. This KFANN$_\mathcal{R}$ query is p_8 since only p_8 can cover all keywords of any 2 query points, and the *flexible* query objects are $\{q_3, q_4\}$.

To the best of our knowledge, there is no any research on FANN$_\mathcal{R}$ problems before. In the literature, FANN queries [8], [9] are first studied in the Euclidean space. Compared with the Euclidean space, many operations on road networks are much more expensive. The most relevant study is the ANN query on road networks [4], [6], [7], but our study of FANN$_\mathcal{R}$ is not a trivial extension or adaption based on ANN query for two main reasons. Firstly, the *IER* algorithm in [4] can reach the best performance, which uses R-tree to index the data objects, but it could be inefficient when only partial points in Q are considered; this is mainly because the number of all possible answers can achieve the scale of $\binom{|Q|}{\phi|Q|}$. Secondly, our comprehensive study of FANN$_\mathcal{R}$ enables us to answer it efficiently in different scenarios (e.g., having an index structure or not, returning an exact answer or not). Obviously, we can regard ANN query as a special case of FANN$_\mathcal{R}$ query when $\phi = 1$. Another relevant but different type of query is the *optimal meeting point* (OMP) query [5] on road networks, but the set P in OMP query is not determined in advance. As for the KFANN$_\mathcal{R}$ query, which is a keyword-aware variant of FANN$_\mathcal{R}$, has also never been studied. FSNNK [10] investigates the FANN with keywords in the Euclidean space. It is challenging to combine both spatial and textual information to prune candidates efficiently on road networks.

In this paper, we first focus on the universal (i.e., applicable to both *max* and *sum*) methods for the FANN$_\mathcal{R}$ and KFANN$_\mathcal{R}$ query problems. In this regard, we develop three solutions. Firstly, we design a Dijkstra-based algorithm by enumerating P instead of $\phi|Q|$-combinations of Q. Secondly, we successfully modify the *List* algorithm [8], [9] in the context of road networks, which processes data points from-near-to-far. Thirdly, we propose a general algorithm framework by combining *Incremental Euclidean Restriction* (IER) and kNN. These universal algorithms often require complex index structures which usually are infeasible for large dynamic road networks. To this end, we also investigate the specific algorithms which sacrifice some generalities but have other merits. In brief, we develop a specific algorithm for *max*-FANN$_\mathcal{R}$ that can return an exact answer; and design a constant-factor approximation algorithm for *sum*-FANN$_\mathcal{R}$, which is sometimes desirable to retrieve a near-optimal answer as fast as possible.

The main contributions of our work can be summarized as follows:

- We firstly introduce an extra flexibility parameter to the classic ANN problem on road networks, and formulate FANN$_\mathcal{R}$ and its keyword-aware variant KFANN$_\mathcal{R}$ formally.
- We design a Dijkstra-based algorithm to answer FANN$_\mathcal{R}$ and KFANN$_\mathcal{R}$ queries, which is much better than adopting ANN as an independent module directly. After that, we propose a queue-based algorithm. We also combine *IER* with kNN, and then develop a family of algorithms based on the general algorithm framework.
- We propose a specific *Exact-max* (exact *max*-FANN$_\mathcal{R}$) algorithm for *max*-FANN$_\mathcal{R}$ and *APX-sum* (approximate *sum*-FANN$_\mathcal{R}$) algorithm for *sum*-FANN$_\mathcal{R}$ to get the exact and 3-approximation answer, respectively. We also prove that the approximation ratio of *APX-sum* can even reach 2 if Q is the subset of P. These specific algorithms are easy to implement and can achieve excellent performance in some scenarios.
- We further extend FANN$_\mathcal{R}$ and KFANN$_\mathcal{R}$ to top-k and multiple queries, and solve them successfully based on our proposed methods.

The rest of this paper is organized as follows: Section 2 defines the problem, discusses some related work, and shows an outline of the proposed approaches. Section 3 presents universal methods, including a Dijkstra-based algorithm, a queue-based method and a general algorithm framework. Section 4 presents specific methods to answer *sum*-FANN$_\mathcal{R}$ and *max*-FANN$_\mathcal{R}$ queries respectively. Section 5 discusses k-FANN$_\mathcal{R}$ (resp., k-KFANN$_\mathcal{R}$) and m-FANN$_\mathcal{R}$ (resp., m-KFANN$_\mathcal{R}$), which are extensions of FANN$_\mathcal{R}$ (resp., KFANN$_\mathcal{R}$). We present the experimental results in Section 6 and make a conclusion in Section 7.

2 PRELIMINARIES

We first present the road network and keyword related definitions and formulate the FANN$_\mathcal{R}$ and KFANN$_\mathcal{R}$ problem formally. Then, we review some related work. Finally, we present the outline of our proposed methods. We also summarize the frequently used symbols in Table 1.

2.1 Problem Formulation

A road network can be represented as an undirected weighted graph, $G = (V, E)$, where V is the set of vertices, and E is the set of edges. If the textual information is considered, each vertex v contains a set of keywords, denoted as $v.\mathcal{W}$. Each keyword x has a weight $x.w$, and it can be generated by the Language Model [11] based on the inverted file.

We use Q to denote the set of query objects, use P to denote the set of data objects, and use ϕ to denote the flexibility parameter, where ϕ varies in the range of $(0, 1]$. For simplicity, we assume that query and data objects are located at vertices, i.e., $P \subset V$ and $Q \subset V$. If the query (resp., data) object is on an edge, we can use the two vertices on the edge to do FANN$_\mathcal{R}$ search and merge the answer sets of the two vertices to generate the final result. If the query (resp., data) object is outside the whole road network, we can find the closest point to it on the road network and use the closest point to do an FANN$_\mathcal{R}$ search. The similar assumption is also adopted in [12]. In the following, "point", "object", and "vertex" are interchangeably used if the context is clear.

Let δ be the distance function on G, and the network distance $\delta(v_i, v_j)$ between objects v_i and v_j is defined as the minimum sum of weights of any path between them. The textual similarity between v_i and v_j is denoted as $t(v_i, v_j)$. It is the normalized sum of the weights (of the shared keywords in $v_i.\mathcal{W}$ and $v_j.\mathcal{W}$); and is compute [10] as:

$$t(v_i, v_j) = \frac{1}{|v_i.\mathcal{W}|} \sum_{x \in s.\mathcal{W}} \frac{x.w}{w_{max}}, \tag{1}$$

where w_{max} is the maximum keyword weight in the dataset, and $s.\mathcal{W} = v_i.\mathcal{W} \cap v_j.\mathcal{W}$. Note that (1) other similarity measures (e.g., Jaccard and cosine similarity) can also be used, but we do not focus on effective measures in this paper; (2) this similarity is not a *metric* since symmetry does not hold (i.e., $t(v_i, v_j) \neq$

$t(v_j, v_i)$). For ease of presentation and without lose of correctness, we assume $|v_i|$ is always the cardinality of the set of query point's keywords. Under this assumption, $t(v_i, v_j) == t(v_j, v_i)$.

When it comes to the distance between objects v_i and v_j in terms of their spatial distance and textual similarity, the widely used linear interpolation with normalization is adopted here:

$$\delta^K(v_i, v_j) = \alpha \frac{\delta(v_i, v_j)}{M_R} + (1-\alpha)(1 - t(v_i, v_j)), \quad (2)$$

where M_R is the maximum road network distance, and $\alpha \in [0, 1]$ is to adjust the relative importance of road network distance. Note that other cost functions in terms of spatial distance and textual similarity can be also adopted.

Let g be an aggregate function, which can be defined on a single object p and a dataset $S \subset V$, and it can be any monotonic (e.g., *min*) one. In this paper, it is either *sum* or *max*:

$$g(p, S) = g\{\Delta(p, v_1), \Delta(p, v_2), ..., \Delta(p, v_k)\}, \quad (3)$$

where $k = |S|$, $v_i \in S$, for $i = 1, 2, \ldots, k$. Note that Δ can be either δ or δ^K according to problem setting. Then we can define the *flexible aggregate function* g_ϕ, which is the most critical operation for both FANN$_\mathcal{R}$ and KFANN$_\mathcal{R}$ query problems.

Definition 1 (Flexible Aggregate Function)**.** The flexible aggregate function g_ϕ, is a function that takes a point $p \in P$ and the set Q as its input, and returns a pair (d^p, Q^p_ϕ) as the result, i.e., $(d^p, Q^p_\phi) = g_\phi(p, Q)$, which satisfies:

$$\begin{cases} Q^p_\phi = \underset{Q_\phi \subset Q, |Q_\phi| = \phi|Q|}{\operatorname{argmin}} g(p, Q_\phi), \\ d^p = g(p, Q^p_\phi), \end{cases}$$

where Q_ϕ is the subset of Q with $|Q_\phi| = \phi|Q|$. Given a point p, we denote the *optimal flexible subset* of Q as Q^p_ϕ, and denote its *flexible aggregate distance* to Q as d^p.

Remark 1. We usually use $g_\phi(p, Q)$ to denote the *flexible aggregate distance* for simplicity and add "keyword-aware" ahead of these notations if textual information is considered.

Our goal is to retrieve a point p^* in P to minimize the flexible aggregate distance. An FANN$_\mathcal{R}$ (resp., KFANN$_\mathcal{R}$) query can be formalized with the following definition:

Definition 2 (FANN$_\mathcal{R}$ (resp., KFANN$_\mathcal{R}$) query)**.** The input of an FANN$_\mathcal{R}$ (resp., KFANN$_\mathcal{R}$) query is a quintuple (G, P, Q, ϕ, g), which returns a triple (p^*, Q^*_ϕ, d^*) as its answer such that:

$$\begin{cases} (p^*, Q^*_\phi) = \underset{p \in P, Q_\phi \subset Q, |Q_\phi| = \phi|Q|}{\operatorname{argmin}} g(p, Q_\phi), \\ d^* = g(p^*, Q^*_\phi), \end{cases}$$

where p^* is the point in P that minimizes the flexible aggregate distance, Q^*_ϕ is the optimal flexible subset, and d^* is the optimal flexible aggregate distance.

TABLE 1
Frequently used notations

Symbol	Definition
ϕ	flexible parameter
$g_\phi(\cdot, \cdot)$	flexible aggregate function
(K)FANN$_\mathcal{R}$	(keyword-aware) flexible aggregate query on road networks
α	coefficient that adjusts the importance of spatial distance
p^*	the point in P that minimizes $g_\phi(\cdot, \cdot)$
Q^*_ϕ	the subset of Q that minimizes $g_\phi(\cdot, \cdot)$
d^*	the distance of minimized $g_\phi(\cdot, \cdot)$

2.2 Related Work

The Shortest Path Algorithm. The shortest path algorithm is one of the most fundamental operations on road networks, and it has been extensively studied during the past half century. The *Dijkstra* algorithm [13] and its variants (e.g., A^* algorithm [14]) have been widely applied in location-based services. We can use either lower bounds (or other heuristic properties) or materialization techniques to accelerate the shortest path computation. The fully materialization of distances requires high storage cost, while *HiTi* [15] and *HEPV* [16] materialize distances partially to make it feasible for large graph. Currently, PHL [17] could be the fastest method which decomposes a graph into the shortest paths and stores distances from each vertex to the shortest path in its labels.

Indexing Techniques on Road Networks. Indexing techniques [15], [16], [18], [19], [20] are also widely used on road networks. The basic idea is to partition the graph into subgraphs recursively, and precompute some shortcuts within subgraphs. It is usually required to keep the hierarchical structure balanced for better performance. *CH* [19] has a low memory overhead, and it may have to traverse a large number of nodes when objects are relatively dispersed in the graph. The authors in [7], [21] transplanted Voronoi digram to the domain of road networks, while it may cause unbalanced partitions. Lee [20] used *ROAD* to index road networks in a hierarchical way, while it may not perform well if the objects are sparse and road networks are large. G-tree [12], [22] has a superior performance and the cost of building index is acceptable.

kNN, ANN, and FANN in the Euclidean Space. The k-nearest neighbor (kNN) query on road networks has been studied for decades. Many successful approaches have been developed to solve this problem [12], [23], [24]. Abeywickrama et al. [25] studied different in-memory algorithms for kNN queries, and they proved that IER has an excellent performance potential. The ANN problem and its top-k extension in both the Euclidean space [1], [2], [3] and road networks [4], [5] have been studied for decades. The *IER* algorithm [4] can reach the best performance, which uses R-tree to index the data objects. Li et al. [8], [9] first studied FANN in the Euclidean space, which generalized the classical ANN problem and offered it richer semantics. They proposed a series of exact and approximate algorithms to address this problem. Nevertheless, those algorithms cannot work for our problem, since we are interested in road networks (instead of Euclidean space).

Spatial Keyword Queries. In the context of spatial and textual domain [26], [27], [28], top-k problem and its variants have been studied, and IR-tree [29] has been proven to be a highly efficient method. Shi et al. [30] further investigated this problem on RDF data. The flexible group spatial keyword query [10] considers the textual information for FANN problem in the Euclidean space, and it also addresses some variants such as multiple flexible subgroup nearest neighbor with keywords.

None of existing work solves the problem of the flexible aggregate nearest neighbor query on road networks, not to mention its keyword-aware variant. In this paper, we will address these two queries respectively. This paper is an extended version of our previous study [31]. The new contributions are three-fold. (1) We introduced the keyword-aware flexible aggregate nearest neighbor queries on road networks (KFANN$_\mathcal{R}$); (2) we extended FANN$_\mathcal{R}$

(resp., KFANN$_{\mathcal{R}}$) to the multiple flexible aggregate nearest neighbor queries on road networks; (3) we conducted new experiments for newly introduced queries, and more comprehensive evaluations for FANN$_{\mathcal{R}}$ which are not shown in [31].

2.3 An Outline of Proposed Methods

In this paper, we design two kinds of algorithms which are known as the *universal* and *specific* methods, respectively. The universal methods are able to deal with both *max* and *sum*, and the specific methods can only solve the problem when g is either *max* or *sum*, while they are generally superior to universal methods in terms of both efficiency and implementation. To save space, we omit the complexity analysis of proposed algorithms, and readers who are interested in them can refer to our previous work [31].

Universal FANN$_{\mathcal{R}}$**.** As for the universal methods, a naive way is to regard the ANN as an independent module. To be specific, we enumerate $\binom{|Q|}{\phi|Q|}$ options to determine Q_ϕ^p (as the query objects), and then apply the ANN routine directly. However, this method is always infeasible in practice since $\binom{|Q|}{\phi|Q|}$ is often too large to deal with. For example, the $\binom{|Q|}{\phi|Q|}$ can reach 2.39×10^{37} if we set $|Q|$ and ϕ to 128 and 0.5 respectively. To this end, we design a Dijkstra-based algorithm to compute g_ϕ (shown in Section 3.1). Although it also searches in an enumerative way, it is much more efficient than the naive solution since it enumerates P (instead of combinations of Q), which has a much less search space. Inspired by the threshold algorithm [32], *List* [8], [9] is proposed to answer FANN queries in the Euclidean space. We modify this *List* (denoted as *R-List*), and implement it in a "switchable" way to answer FANN queries on road networks (i.e., FANN$_{\mathcal{R}}$), as shown in Section 3.2. The modified implementation also leads to much more efficient solution to *max*-FANN$_{\mathcal{R}}$ problems (shown in Section 4.1). As presented in [25], IER has an excellent potential when retrieving kNN. Hence, we further design a IER-kNN framework to answer FANN$_{\mathcal{R}}$ queries. Based on the general IER-kNN framework, we can obtain a family of algorithms (shown in Section 3.3).

Universal KFANN$_{\mathcal{R}}$**.** The straightforward adaption for universal KFANN$_{\mathcal{R}}$ is to use the maximum textual similarity as a bound while expanding the road path. Note that given a query point q, we cannot determine specific distance order in terms of points in P, so *R-List* is ineffective for the keyword-aware variant. In this paper, we use an IR-tree [29] like structure for IER-kNN framework. Different from other extensions (e.g., [10]), we do not have to store the keywords explicitly in tree nodes, and use a roaring bitmap to encode the textural information which can boost the space efficiency greatly.

Specific FANN$_{\mathcal{R}}$**.** Although the universal methods can generally achieve good performance, they highly rely on sophisticated indexing techniques. The construction cost of index can often be very high especially for frequently changing road networks. Motivate by this, we design a specific algorithm when g is max, which can return an exact answer. We denote it as *Exact-max* (shown in Section 4.1). *Exact-max* follows the basic data structure of *R-List*, but it often outperforms other methods when it is index-free. On the other hand, sometimes it is often desirable to obtain a near-optimal result if the running time can be greatly reduced. To this end, we design an *APX-sum* algorithm to answer *sum*-FANN$_{\mathcal{R}}$ queries. This algorithm can return a 3-approximation result. We further prove that it can even return a 2-approximation result if Q is the subset of P (shown in Section 4.2). It is worth noting that these specific algorithms rely on the spatial properties, so they cannot be applied for KFANN$_{\mathcal{R}}$.

3 UNIVERSAL METHODS

We firstly present a Dijkstra-based algorithm (Section 3.1), and then present a queue-base algorithm (Section 3.2). Finally, we show a framework which is able to generate a family of algorithms (Section 3.3).

3.1 Dijkstra-based Algorithm

3.1.1 FANN$_{\mathcal{R}}$

The Dijkstra-based algorithm is based on the following observation. Recall how *Dijkstra* routine runs: at every step of its expansion, it chooses an unvisited nearest object of the source object to visit and updates its neighbors' distances to the source object. This *INE* (i.e., incremental network expansion) behavior also makes sense in running $g_\phi(p, Q)$. First, let p be the source object, we call a *Dijkstra*-like routine on it. Then we keep the path expanding until $\phi|Q|$ objects in Q are labeled as visited. Hence, these $\phi|Q|$ points are exactly Q_ϕ^p. In this way, we can enumerate points in P and return the one with the smallest flexible aggregate distance. At a high level, the difference between the naive method mentioned in Section 2.3 and the Dijkstra-based algorithm is that: the former is to construct Q_ϕ first, and then to determine the correct p and its flexible aggregate distance; the latter is to choose p first, and then retrieve the correct Q_ϕ^p and its flexible aggregate distance. Clearly, the complexity of the latter strategy is much smaller.

Intuitively, there are two ways to improve this algorithm: (1) prune objects in P as much as possible (i.e., reduce the number of calling g_ϕ), or (2) improve the implementation of g_ϕ. We will discuss related improvement techniques later (e.g, Sections 3.2 and 3.3).

3.1.2 KFANN$_{\mathcal{R}}$

Different from FANN$_{\mathcal{R}}$, in order to prune unpromising *INE* or *Dijkstra* expansion, we use the textual similarity as a bound while expanding. To be specific, for a data point p, we use a *max*-heap with size of $\phi|Q|$ to maintain the expanding results in terms of road network distance and textual similarity. Suppose we have visited at least $\phi|Q|$ query objects and query object q^* has the largest distance value in the max-heap. We can terminate the expansion at q_x if

$$\alpha\frac{\delta(p, q_x)}{M_R} + (1 - a)(1 - t_{max}) \geq \delta^K(p, q^*). \quad (4)$$

Like FANN$_{\mathcal{R}}$, the whole algorithm is to conduct a keyword-aware expanding procedure for each point in P.

Time complexity. Since g_ϕ has the same time cost with *Dijkstra* (i.e., $O(|E| + |V|\log|V|)$), the total time cost is $O((|E| + |V|\log|V|)|P|)$ in the worst case.

3.2 The R-List Algorithm

The basic idea of *R-List* algorithm is to construct a threshold [32] for early termination, and thus it is able to reduce the number of calling g_ϕ. The *R-List* (road networks' *List* [8], [9]) is actually queue-based. We create $|Q|$ queues. Each queue corresponds to an object in Q, and processes data points in a from-near-to-far way.

Although *R-List* shares the basic idea with *List* [8], [9], we have two contributions here. Firstly, we provide the implementation details for constructing the list of queues in the context of road networks, which are different from that in the Euclidean space. Secondly and most importantly, the modified implementation can lead to a much more efficient solution to *max*-FANN$_\mathcal{R}$ problems. As for the implementation details as well as the time complexity, we will discuss them together with those of *Exact-max* in Section 4.1.

3.3 IER-kNN Framework

3.3.1 FANN$_\mathcal{R}$

In this section, we propose a powerful algorithm framework for the FANN$_\mathcal{R}$ query. Firstly, we adopt the Incremental Euclidean Restriction (*IER*) to prune the unpromising points in P as many as possible. Let e be an entry of an R-tree that indexes P and $e.b$ be its minimum bounding rectangle (MBR). We can calculate the minimum possible distance from an entry e to a point q, denoted as $\delta(e.b, q)$. We denote g^ϵ as the *Euclidean aggregate function*, and we have

$$g^\epsilon(p, Q) = g\{\delta^\epsilon(p, q_1), \delta^\epsilon(p, q_2), \ldots, \delta^\epsilon(p, q_{|Q|})\}. \quad (5)$$

Similarly, $g^\epsilon(e, Q)$ can be defined by $g\{\delta(e.b, q_i), \forall q_i \in Q\}$. Like Definition 1, we can define the *flexible Euclidean aggregate function* (Euclidean FANN) by $g^\epsilon_\phi(p, Q)$ if replacing $g(p, Q_\phi)$ with $g^\epsilon(p, Q_\phi)$. Following the similar way, we also have $g^\epsilon_\phi(e, Q)$ to denote the flexible Euclidean aggregate function with respect to an MBR $e.b$ and query points set Q. For simplicity, we also use g^ϵ_ϕ to denote its flexible Euclidean aggregate distance if the context is clear. Now, we can have the following lemma:

Lemma 1. Let Q be a set of query points and e be an R-tree node entry. For any point p indexed under e, $g^\epsilon_\phi(e, Q)$ cannot be larger than $g_\phi(p, Q)$.

Proof. It follows from the fact that $g^\epsilon_\phi(e, Q) \leq g^\epsilon_\phi(p, Q)$ and $g^\epsilon_\phi(p, Q) \leq g_\phi(p, Q)$. □

Based on Lemma 1, we can solve the FANN$_\mathcal{R}$ query using an R-tree built on P. We show this process in Algorithm 1. Initially, the root of the R-tree is enqueued into a priority queue which is sorted by $g^\epsilon_\phi(e, Q)$ in *ascending* order (line 2). For each iteration, we first check whether $g^\epsilon_\phi(e, Q)$ is larger than or equal to the current best candidate result (line 5). If so, we terminate the algorithm (line 6); otherwise, we check whether the dequeued item is an R-tree node (line 8). If so, we push all entries under this node into the priority queue (lines 9-10); otherwise, we run $g_\phi(e, Q)$ on it and update the result if necessary (lines 12-14). Note that the entry $\hat{e}$ (line 9) is a data point in P if e is a leaf node, and it is an R-tree node if e is a non-leaf node.

A Running Example. Let us see how Algorithm 1 finds the sum-FANN$_\mathcal{R}$ in Fig. 1 whose $\phi = 50\%$. We illustrate this process in Fig. 2, and remove the road segments for better visualization. In the first round of loop, we would push (MBR$_1$, 7), (MBR$_2$, 1.8), and (MBR$_3$, 21) into H. After that, (MBR$_2$, 1.8) will be chosen, and we would check p_3 and p_6 inside MBR$_2$. After that, we can know $p^* = p_3$, $d^* = 4$, and $Q^*_\phi = \{q_1, q_2\}$. We can safely terminate the algorithm since d^* is less than the distance value of head in H.

Revisitation of $g_\phi(p, Q)$. Now we revisit the implementation of $g_\phi(p, Q)$. As implied in Section 3.1, given a point p, Q and ϕ, the flexible aggregate function $g_\phi(p, Q)$ is exactly an *incremental network expansion* (INE), which is also a kNN query where p is a query node, Q is the set of data objects, and $k = \phi|Q|$.

Generally speaking, A* is believed to be superior to Dijkstra (i.e., INE) for computing shortest distance. However, A* is not necessarily better than INE when it comes to the implementation of g_ϕ, as showed in our experiments. This finding is also noted by [25]. Hence, we still adopt INE as one of the implementations of g_ϕ. To boost efficiency of computing the shortest path distance and kNN, we apply two state-of-the-art techniques. The first is G-tree [12], [22], which materializes distance matrix for each tree node. The second is the *pruned highway labeling* (PHL) [17], which accelerates the shortest path distance queries by decomposing a graph into shortest paths and storing distances from each vertex to the shortest path in its labels. We denote the IER-kNN framework combined with the INE algorithm as IER-INE. Similarly, we also have IER-A*, IER-GTree and IER-PHL to represent the IER-kNN framework combined with A*, G-tree and PHL respectively.

As a remark, $g_\phi(p, Q)$ itself can be also implemented with IER and any shortest path distance algorithm when Q is indexed under R-tree. In this way, we denote the IER-kNN method whose g_ϕ is implemented with IER-A* as IER^2-A* (there are double IER routines). The IER-A* here means the g_ϕ method, instead of the FANN$_\mathcal{R}$ approach mentioned earlier. Similarly, if we replace A*

Algorithm 1: IER-kNN Framework

Input: G, P, Q, ϕ, g, R
Output: p^*, Q^*_ϕ, d^*

1 $d^* \leftarrow \infty$, $H \leftarrow$ new priority queue
2 $H.enqueue(R.root, g^\epsilon_\phi(R.root, Q))$
3 **while** H *is not empty* **do**
4 $\quad (e, g^\epsilon_\phi(e, Q)) \leftarrow H.top()$
5 $\quad$ **if** $g^\epsilon_\phi(e, Q) \geq d^*$ **then**
6 $\quad\quad$ break
7 $\quad H.dequeue()$
8 $\quad$ **if** e *is an R-Tree node* **then**
9 $\quad\quad$ **foreach** *R-Tree entry* $\hat{e}$ *under* e **do**
10 $\quad\quad\quad H.enqueue(\hat{e}, g^\epsilon_\phi(\hat{e}, Q))$
11 $\quad$ **else**
12 $\quad\quad (Q^e_\phi, d^e) \leftarrow g_\phi(e, Q)$
13 $\quad\quad$ **if** $d^e < d^*$ **then**
14 $\quad\quad\quad p^* \leftarrow e$, $d^* \leftarrow d^e$, $Q^*_\phi \leftarrow Q^e_\phi$

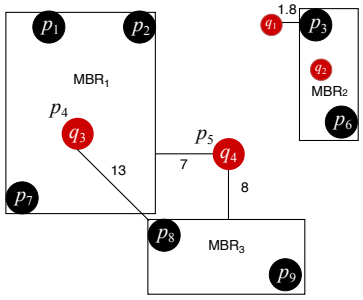


Fig. 2. Example of Algorithm 1

TABLE 2
Road network index of g_ϕ

Algorithm Name	G-tree	PHL	R-tree	Occ
INE	✗	✗	✗	✗
A*	✗	✗	✗	✗
GTree	✓	✗	✗	✓
PHL	✗	✓	✗	✗
IER-A*	✗	✗	✓	✗
IER-GTree	✓	✗	✓	✗
IER-PHL	✗	✓	✓	✗

here with GTree (or PHL), we also have IER^2-GTree (or IER^2-PHL). Note that the "GTree" in IER-GTree is the kNN algorithm presented in [12], [22], which is based on an *occurrence list* (Occ) over Q, and the "GTree" in IER^2-GTree denotes the shortest path distance algorithm based on G-Tree index. In this way, we can have a family of algorithms based on IER-kNN. The index techniques of different implementations for g_ϕ are summarized in Table 2, and they will be further studied in experiments.

3.3.2 KFANN$_\mathcal{R}$

Different from FANN$_\mathcal{R}$, we need to encode textual information into R-tree, so we adopt a IR-tree [29] like structure to index our data points. Since the textual similarity is generated by shared keywords of the query point and data point, we need to compute the weights of keywords in Q, which can be computed off-line by the Language Model [11] based on inverted file. In order to compute the textural similarity between a tree node entry e and a query object q, some studies (e.g., [10]) have to store the union of keywords in the child node entries. Yet, such a storage model could lead to a high space overhead. One possible solution is to use a Bloom filter to encode the keywords on each tree node. Although a bloom filter is space-efficient, it would cause false positives.

To overcome the above drawbacks, we adopt a roaring bitmap [33], which is characterized by its higher compressed ratio and better performance than other popular bitmap compression schemes. We show the cost of IR-tree with different textual encodings in Section 6.6.2, and it demonstrates the effectiveness of roaring bitmaps. In addition, roaring bitmaps support *union* and *intersection* operations which are necessary when building index and computing the textual similarity $t(e, q)$.

Given $t(e, q)$, the keyword-aware spatial distance between a tree node entry e and a query object q can be defined by:

$$\delta^K(e, q) = \alpha \frac{\delta(e.b, q)}{M_R} + (1 - \alpha)(1 - t(e, q)), \qquad (6)$$

which is an extension of Equation 1. Similarly, keyword-aware $g^\epsilon(e, Q)$ can be defined by $g\{\delta^K(e, q_i), \forall q_i \in Q\}$. Now we can have a lemma below, which is a keyword-aware variant of Lemma 1:

Lemma 2. Let Q be a set of query points and e be an IR-tree node entry. For any point p indexed under e, $g^\epsilon_\phi(e, Q)$ cannot be larger than $g_\phi(p, Q)$.

Proof. We have $t(e, q) \geq t(p, q)$ since $p.\mathcal{W} \subseteq e.\mathcal{W}$. Thus, $\delta^K(e, q)$ is a lower bound of $\delta^K(p, q)$. It is easy to know $g^\epsilon_\phi(e, Q) \leq g^\epsilon_\phi(p, Q)$ and $g^\epsilon_\phi(p, Q) \leq g_\phi(p, Q)$ hold true. □

Therefore, we can use Algorithm 1 to answer KFANN$_\mathcal{R}$ based on Lemma 2, because we can prune all unpromising candidates in the priority queue if $g^\epsilon(e, Q) \geq d^*$ (line 5).

Similarly, we can further boost the efficiency $g_\phi(p, Q)$ here, which is a keyword-aware kNN query [12], [22]. Note that even though p is supposed to be a query point by definition, we shall still regard $q_i \in Q$ as a query point when computing textual similarity due to its asymmetry (recall Section 2.1).

Time complexity. In the worst case, we still have to visit each point in P. Take the IER-GTree for an example. The worst time of kNN search is $O(|V| \log |V|)$, so the total time complexity is $O(|P||V| \log |V|)$. In practice, the time cost is much smaller than worst complexity [12].

4 SPECIFIC METHODS FOR FANN$_\mathcal{R}$

Although the universal algorithms for FANN$_\mathcal{R}$ queries in Section 3 can achieve good performance, they highly depend on the sophisticated indexing techniques (e.g., G-tree or PHL). In some scenarios (e.g., maps in online games), it is difficult to build indexes for underlying road networks, incurring these general methods ineffective, to some extent. To this end, in this section, we design two specific algorithms to solve *sum*-FANN$_\mathcal{R}$ and *max*-FANN$_\mathcal{R}$ queries respectively, and these specific methods can achieve excellent performance even if the underlying road networks are index-free.

4.1 The Exact-max Algorithm

We present this method in Algorithm 2, and call it *Exact-max* (exact *max*-FANN$_\mathcal{R}$), which shares the similar idea and data structure with those of *R-List*. The main difference is that we add a counter for every point in P. Initially, these counters are set to 0 (line 2). During every iteration, we get the head point with the smallest distance (line 4), and then increase the counter associated with the head node by one (line 5). If the counter associated with the head node reaches $\phi|Q|$, the head node is exactly p^*, and then we can terminate the algorithm safely (lines 6-9). Hence, we can run the time-consuming g_ϕ only once (line 8). This is why *Exact-max* can be efficient. Besides, this also indicates that **different implementations of g_ϕ have little influence on *Exact-max***. In other words, *Exact-max* can still achieve a good performance even if we do not build a road network index over the whole road network. This property is appealing when underlying road networks change frequently, since we do not need to rebuild or readjust the index any more, which is usually time-consuming as shown in our experiments.

Algorithm 2: The *Exact-max* Algorithm

Input: G, P, Q, ϕ, δ, g
Output: p^*, Q^*_ϕ, d^*

1 **foreach** $p \in P$ **do**
2 $count[p] \leftarrow 0$
3 **while** *true* **do**
4 $L_{min} \leftarrow$ the queue whose head has the smallest distance
5 $count[L_{min}.top()] \leftarrow count[L_{min}.top()] + 1$
6 **if** $count[L_{min}.top()] \geq \phi|Q|$ **then**
7 $p^* \leftarrow L_{min}.top()$
8 $(Q^*_\phi, d^*) \leftarrow g_\phi(L_{min}.top(), Q)$
9 break
10 $L_{min}.dequeue()$

Implementation Details. Now we discuss the implementation details of *R-List* and *Exact-max*. The *list* of queues is constructed in an implicit way, otherwise it will violate the memory limit if $O(|P||Q|)$ is large enough. To be specific, we set the query objects as the multiple sources initially, and then execute *Dijkstra*-like routine on them simultaneously and independently. As shown in Algorithm 2, the queue operations are alternately performed on different queues, which implies that we can implement this *multi-source Dijkstra* procedure in a "switchable" way. All data structures associated with different queues should be well preserved when the *Dijkstra*-like routine switches away, thus the interrupted search process can be reloaded and resumed when it switches back.

A Running Example. Let us see how *Exact-max* finds the max-FANN$_\mathcal{R}$ in Fig. 1 whose $\phi = 50\%$. The expanding paths of $\{q_1, q_2, q_3, q_4\}$ are $\{p_3, \dots\}$, $\{p_3, \dots\}$, $\{p_4, \dots\}$ and $\{p_5, \dots\}$ respectively. It is obvious that the counter of p_3 will be the first to reach $\phi \times 4 = 2$. Hence the result of this max-FANN$_\mathcal{R}$ query is $p^* = p_3$, $d^* = 2$ and $Q^*_\phi = \{q_1, q_2\}$.

The correctness of this algorithm is easy to validate: the nature of *Dijkstra* algorithm guarantees that the closer a point to the source is, the earlier it will be visited. When a point is the first to be visited by exact $\phi|Q|$ sources, it means that this point is the closest one to these $\phi|Q|$ sources. Thus Algorithm 2 can answer *max*-FANN$_\mathcal{R}$ queries correctly.

As analyzed above, both *R-List* and *Exact-max* require road network expansions, and the order of data points in terms of road network distance can be easily guaranteed due to the nature of *Dijkstra* (or INE procedure). However, given a query object, we cannot guarantee the order of data points in terms of both spatial and textual distance while expanding. Therefore, both *R-List* and *Exact-max* are ineffective for KFANN$_\mathcal{R}$.

Note that the basic idea of *Exact-max* is different from g_ϕ when g_ϕ is implemented with *Dijkstra* or *INE*. The latter is to regard P as sources and then obtain the kNN (i.e., the objects in Q are destinations). The expansion direction of *Exact-max* is quite the reverse: we regard the nodes in Q as sources and then expand them to P (i.e., the objects in P are destinations). It is not hard to understand that *Exact-max* is very efficient when P is dense and $\phi|Q|$ is relatively small. In most real word scenarios, this is true because $|Q|$ is much smaller than $|P|$.

TABLE 3
A counter example of sum-FANN$_\mathcal{R}$

Source	Expanding		
q_1	$(4, p_2)$	$(12, p_3)$	-
q_2	$(2, p_1)$	$(10, p_2)$	-
q_3	$(11, p_1)$	-	-
q_4	$(14, p_4)$	-	-
q_5	$(15, p_2)$	-	-

It is worth noting that the idea behind *Exact-max* cannot be used to answer *sum*-FANN$_\mathcal{R}$ queries. Table 3 illustrates a counter example (note that it has noting to do with the example in Fig. 1). Suppose query points set Q is $\{q_1, q_2, q_3, q_4, q_5\}$ (any query point does not belong to $\{p_1, p_2, p_3, p_4, p_5\}$), and $\phi = 40\%$. Hence, $\phi|Q| = 2$. If we follow the idea of Algorithm 2, we would conduct the following four steps: (1) We examine the queue of q_2 first, and visit data point p_1; (2) we examine the queue of q_1, and visit data point p_2; (3) we examine the queue of q_2 again, and visit data point p_2 again; (4) at this moment, the counter of p_2 reaches 2, and thus we can terminate the algorithm. In this way, we can have $p^* = p_2$, $d^* = 4 + 10 = 14$, and $Q^*_\phi = \{q_1, q_2\}$. However, the correct answer is $p^* = p_1$, $d^* = 2 + 11 = 13$, and $Q^*_\phi = \{q_2, q_3\}$.

Algorithm 3: The *APX-sum* Algorithm

Input: G, P, Q, ϕ, δ, g
Output: p^α, Q^α_ϕ, d^α

```
1 candidate ← ∅
2 foreach q ∈ Q do
3     p ← the nearest neighbor of q in P
4     candidate.insert(p)
5 FANN_R (G, candidate, Q, φ, sum)
```

Time complexity. Assume that g_ϕ is implemented by Dijkstra-like method. For R-List algorithm, every point in P will be visited in the worst case. Thus, the time cost is $O((|E| + |V| \log |V|)|P|)$. In practice, the time complexity is often smaller than it due to the lower bound. Similarly, the time cost of *Exact-max* is $O(|E| + |V| \log |V|)$.

4.2 The APX-sum Algorithm

For the *sum*-FANN$_\mathcal{R}$ problem on road networks, we present an approximate approach *APX-sum* (approximate *sum*-FANN$_\mathcal{R}$) in Algorithm 3. This algorithm is extremely simple, but it has a constant approximation ratio. Instead of considering the whole P, we only examine those data points which are the nearest neighbors of those query points in Q (lines 2-4). We can regard the candidate set as P, and run the FANN$_\mathcal{R}$ algorithm (whose g is sum). In this way, we reduce the number of candidate data points to $|Q|$, which is usually much smaller than $|P|$. This is why it can remarkably improve the search efficiency. In fact, it is even possible that the size of candidate set is smaller than $|Q|$, since some different query points may have the same nearest data point neighbors. One of the most appealing properties of *APX-sum* is the stability when varying P, because it is only affected by Q generally. We can prove that the approximation ratio d^α/d^* of this algorithm is no more than 3.

Theorem 1. Algorithm 3 returns a 3-approximation answer to any *sum*-FANN$_\mathcal{R}$ query on road networks.

Proof: Given a *sum*-FANN$_\mathcal{R}$ query, Algorithm 3 returns an approximate answer $(p^\alpha, Q^\alpha_\phi, d^\alpha)$, and suppose that the true optimal answer is (p^*, Q^*_ϕ, d^*). Let q^τ be the nearest object in Q^*_ϕ to p^*, and p^τ be the nearest object in P to q^τ. We have:

$$\delta(p^\tau, q^\tau) \le \delta(p^*, q^\tau) \tag{7}$$

And for any $q \in Q^*_\phi$, we have:

$$\delta(p^*, q^\tau) \le \delta(p^*, q) \tag{8}$$

$$\Rightarrow \phi M \cdot \delta(p^*, q^\tau) \le \sum_{q \in Q^*_\phi} \delta(p^*, q) = d^* \tag{9}$$

It is obvious that p^τ cannot be "better" than p^α. If the result of $g_\phi(p^\tau, Q)$ is (Q^τ_ϕ, d^τ), we have:

$$\begin{aligned} d^\alpha &\leq d^\tau \\ &= \sum_{q\in Q^\tau_\phi} \delta(p^\tau, q) \\ &\leq \sum_{q\in Q^*_\phi} \delta(p^\tau, q) \\ &\leq \sum_{q\in Q^*_\phi} (\delta(p^\tau, p^*) + \delta(p^*, q)) \qquad \text{(by Triangle Inequality)} \\ &= \sum_{q\in Q^*_\phi} \delta(p^\tau, p^*) + d^* \\ &= \phi M \cdot \delta(p^\tau, p^*) + d^* \\ &\leq \phi M \cdot (\delta(p^\tau, q^\tau) + \delta(q^\tau, p^*)) + d^* \\ &\leq 2\phi M \cdot \delta(q^\tau, p^*) + d^* \qquad \text{(by Equation 7)} \\ &\leq 2d^* + d^* \qquad \text{(by Equation 9)} \\ &= 3d^* \end{aligned}$$

□

A Running Example. Let us see how *APX-sum* finds the sum-FANN$_\mathcal{R}$ in Fig. 1 whose $\phi = 50\%$. We can easily obtain the candidates of data points $\{p_3, p_4, p_5\}$, and *APX-sum* returns $p^* = p_3$, $d^* = 4$ and $Q^*_\phi = \{q_1, q_2\}$ as the final result, which is coincidently an exact answer.

It should be pointed out that, the approximation ratio 3 is only a theoretical bound. As shown in our experiments, the approximation ratio of *APX-sum* is far less than 3 in practice: it never exceeds 1.2 in our experiments. Furthermore, an interesting result can be derived from Theorem 1 when Q is the subset of P.

Theorem 2. Algorithm 3 returns a 2-approximation answer to any *sum*-FANN$_\mathcal{R}$ query if Q is a subset of P.

Proof: The proof of the bound is similar to the proof of Theorem 1. If Q is the subset of P, then p^τ is q^τ itself in the proof of Theorem 1, so $\delta(p^\tau, q^\tau) = 0$ holds. Clearly, $d^\alpha \leq 2d^*$ is true once replacing $\delta(p^\tau, q^\tau)$ with 0 in the proof of Theorem 1. □

On the top of a better approximation ratio, we can easily find that it will also avoid the cost of expanding to $|Q|$ nearest neighbors, and thus it is more efficient than Algorithm 3.

Time complexity. If the g_ϕ is implemented as Dijkstra or INE, the time cost is $O((|E| + |V| \log |V|)|Q|)$.

5 Extension to top-k and multiple FANN$_\mathcal{R}$ (resp., KFANN$_\mathcal{R}$)

5.1 The Top-k FANN$_\mathcal{R}$ (resp., KFANN$_\mathcal{R}$)

5.1.1 k-FANN$_\mathcal{R}$

We define the k-FANN$_\mathcal{R}$ query, which is a further extension of FANN$_\mathcal{R}$. We can regard FANN$_\mathcal{R}$ as a special case of k-FANN$_\mathcal{R}$ when k is 1.

Definition 3 (k-FANN$_\mathcal{R}$ query)**.** The input of a k-FANN$_\mathcal{R}$ query is a six-tuple (G, P, Q, ϕ, g, k), which returns a vector X of size k as its answer, and each element of X is in the form of (p_i, Q^i_ϕ, r_i), where $p_i \in P$ and $(r_i, Q^i_\phi) = g_\phi(p_i, Q)$, such that for any data point $p_0 \in P \setminus \{p_1, p_2, \cdots, p_k\}$, its flexible aggregate distance to Q is greater than or equal to $\max\{r_1, r_2, \cdots, r_k\}$.

Algorithm 4: IER-kNN Framework for m-FANN$_\mathcal{R}$

Input: G, P, Q, ϕ_{low}, ϕ_{high}, g, R
Output: A vector of $(p^*_i, Q^*_{\phi_i}, d^*_i)$, $i \in [1, m]$

```
d*_i, ..., d*_m ← ∞, H ← new priority queue
H.enqueue(R.root, {g^ε_{φ_1}(R.root, Q), ... g^ε_{φ_m}(R.root, Q)})
while H is not empty do
    (e, {g^ε_{φ_1}(e, Q), ... g^ε_{φ_m}(e, Q)}) ← H.top()
    H.dequeue()
    if ∀i ∈ [1, m] : g^ε_{φ_i}(e, Q) ≥ d*_i then
        continue
    if e is an R-Tree node then
        foreach R-Tree entry ê under e do
            H.enqueue(ê, {g^ε_{φ_1}(ê, Q), ... g^ε_{φ_m}(ê, Q)})
    else
        for i = 1 → m do
            (Q^e_{φ_i}, d^e_i) ← g_{φ_i}(e, Q)
            if d^e < d* then
                p*_i ← e, d*_i ← d^e, Q*_{φ_i} ← Q^e_{φ_i}
```

Obviously, it is unnecessary for $Q^{p_i}_\phi$ to be the same, where $i = 1, 2, \cdots, k$. With some minor modifications, most of the algorithms presented above can be easily adapted to answer the k-FANN$_\mathcal{R}$ query. Now we take the *Exact-max* algorithm as an example to illustrate how to answer a k-FANN$_\mathcal{R}$ (more extension examples, such as *R-List* algorithm and IER-kNN framework, can be found in [31]).

The *Exact-max* Algorithm. For the k-FANN$_\mathcal{R}$ problem, we should expand the paths until k different counters reach $\phi|Q|$. Then, we can terminate the search routine and return the k corresponding query nodes and their flexible distances as the final answer.

Remark 2. As for the *APX-max* algorithm, it is unclear how to guarantee the approximation bound for the k-FANN$_\mathcal{R}$, and we leave it as an open problem in our future work.

5.1.2 k-KFANN$_\mathcal{R}$

Like k-FANN$_\mathcal{R}$ query, we can regard k-KFANN$_\mathcal{R}$ as a special case of k-KFANN$_\mathcal{R}$ when k is 1, so it is trivial to define k-KFANN$_\mathcal{R}$ based on Definition 3. Note that only the Dijkstra-based algorithm and IER-kNN framework are able to solve KFANN$_\mathcal{R}$, we now take the IER-kNN framework as an example to show the extension.

The IER-*k*NN Framework. We maintain a priority queue which is sorted by the flexible aggregate distance in ascending order. Instead of comparing $g^\epsilon_\phi(e, Q)$ with the smallest flexible aggregate distance, we compare it with the k-th smallest flexible aggregate distance d in the priority queue. If $g^\epsilon_\phi(e, Q)$ is larger than or equal to d, we can terminate the algorithm and return the priority queue as the k-KFANN$_\mathcal{R}$ answer.

5.2 The Multiple FANN$_\mathcal{R}$ (resp., KFANN$_\mathcal{R}$)

Different from the FANN$_\mathcal{R}$'s top-k extension, m-FANN$_\mathcal{R}$ means the *multiple* flexible aggregate nearest neighbor queries on road networks. This definition is mainly motivated by the consensus query [34]. Given a positive integer m ($\leq |Q|$), the consensus

query returns objects in P that minimize the aggregate distance for query subsets with sizes in the range $[m, |Q|]$. In fact, the consensus query is a special case of our multiple FANN$_\mathcal{R}$ because we can specify both upper and lower bounds of the size of *flexible* subset of Q while only the lower bound can be specified in the consensus query.

Definition 4 (m-FANN$_\mathcal{R}$ query)**.** The input of an m-FANN$_\mathcal{R}$ query is a six-tuple $(G, P, Q, \phi_{low}, \phi_{high}, g)$, where $0 < \phi_{low} \leq \phi_{high} \leq 1$, which returns a vector X of size $(\phi_{high}|Q| - \phi_{low}|Q| + 1)$ as its answer, and each element in X is the result of an FANN$_\mathcal{R}$ query whose flexibility parameter is ϕ_i, where $\phi_i = \frac{\phi_{low}|Q|+i-1}{|Q|}$, and $i = 1, 2, \cdots, \phi_{high}|Q| - \phi_{low}|Q| + 1$.

Remark 3. It is trivial to define m-KFANN$_\mathcal{R}$ if we add the keyword-aware distance based on m-FANN$_\mathcal{R}$, and their modifications on implementation are quite similar to each other. Therefore, we only discuss m-FANN$_\mathcal{R}$ in the following.

A most straightforward approach to the m-FANN$_\mathcal{R}$ query is to run $(\phi_{high}|Q| - \phi_{low}|Q| + 1)$ times FANN$_\mathcal{R}$ with different flexibility parameters. In other words, it has to access P through $(\phi_{high}|Q| - \phi_{low}|Q| + 1)$-passes. Fortunately, with a few modifications, all proposed methods can answer it through only one-pass over P. In the following, we use the IER-*k*NN framework to illustrate it.

The IER-*k*NN Framework. Let $m = \phi_{high}|Q| - \phi_{low}|Q| + 1$. We maintain an m-element vector of tuples and each tuple is in the form of $(p_i^*, Q_{\phi_i}^*, d_i^*)$ as final results. The element of priority queue H is in the form of $(e, \{g_{\phi_1}^\epsilon(e, Q), \ldots, g_{\phi_m}^\epsilon(e, Q)\})$. Note that it is difficult to determine the best way to sort H here. In this paper, we adopt a heuristic sorting method. To be specific, H is sorted by $g_{\phi_m}^\epsilon(e, Q)$ in ascending order. Algorithm 4 shows the modified implementation, and it shares the similar procedure with Algorithm 1. If for every $i \in [1, m]$, $g_{\phi_i}^\epsilon(e, Q) \geq d_i^*$ holds true, we can prune the entry e (lines 6-7) safely.

Note that Algorithm 4 will not add extra overhead for g_ϕ^ϵ (line 10) because the previous g_ϕ^ϵ in Algorithm 1 will also traverse the Q. Similarly, the for-loop (lines 12-15) can also be achieved by one-pass over Q (or just more expansions if g_ϕ is incremental).

6 EXPERIMENTS

6.1 Setup

We implemented all the algorithms mentioned in this paper by standard C++, and executed our experiments on a Linux machine with dual 6-core Intel Xeon E5-2620 processors at 2.00 GHz and 64 GB DDR3 RAM. All of our road network datasets come from the real word[1]. Note that the original datasets have some errors, such as unconnected components or self-loops, and we have cleaned it up at the preprocessing stage. We show them in Table 4, and use the NW as our default road network.

Given a road network, we focus on several key factors that generate P, and Q, as well as key parameters that affect FANN$_\mathcal{R}$, KFANN$_\mathcal{R}$ and their extensions. The default values are in bold. Note that for simplicity, we set ϕ_{low} to 0.1 for m-FANN$_\mathcal{R}$ (resp., m-KFANN$_\mathcal{R}$), and thus use different values to vary m, which is bijective to a ϕ_{high}.

Parameters for P and Q.

TABLE 4
Road network datasets

Name	Description	# nodes	# edges
DE	Delaware	48,812	119,004
ME	Maine	187,315	412,352
COL	Colorado	435,666	1042,400
NW	Northwest USA	1,089,933	2,545,844
E	Eastern USA	3,598,623	8,708,058
CTR	Central USA	14,081,816	33,866,826
USA	Full USA	23,947,347	57,708,624

- d (the density of P): [0.0001, **0.001**, 0.01, 0.1, 1.0];
- A (the coverage ratio of Q): [1%, 5%, **10**%, 15%, 20%];
- M (the size of Q) [64, **128**, 256, 512, 1024];
- C (the number of clusters of Q): [**1**, 2, 4, 6, 8].

Parameters for Queries.

- ϕ (the flexibility parameter): [0.1, 0.3, **0.5**, 0.7, 1.0];
- α (the weight of road network distance): [0.1, 0.3, **0.5**, 0.7, 1.0];
- k (the top-k parameter of k-FANN$_\mathcal{R}$ and k-KFANN$_\mathcal{R}$): [**1**, 5, 10, 15, 20];
- m (the *multiple* parameter of m-FANN$_\mathcal{R}$ and m-KFANN$_\mathcal{R}$): [**1**, 25, 50, 75, 95].

Although all of our underlying road networks are from the real world, we can use d, A, M and C to generate synthetic P and Q. Besides the synthetic datasets, we also use some real-word POIs as P and Q directly. In what follows, we summarize the generation methods of P and Q.

- **Synthetic uniform** P: Parameter d reflects the ratio of the size of P to the size of nodes in the whole graph (i.e., $|P|/|V|$). Given a density, P is generated randomly on the road network.
- **Synthetic uniform** Q: Parameter A reflects aggregation degree of the query nodes in Q. We first randomly select a node in V as a source node (i.e., *seed* node), and calculate the shortest path distances from it to all other nodes in V. We denote the maximum one as the *radius* of G. Next, we randomly choose M nodes from G, whose distances to the seed node are no more than $A \times radius$. We assume the size of nodes in such region is always larger than or equal to M, and if there is no enough objects in the region, we simply expand outward until the size reaches M.
- **Synthetic clustered** Q: In some scenarios, the query points are not uniformly distributed. Some locations, such as schools, often occur in clusters. After we determine a region by A on the road network, we select C central nodes as seeds in the selected region, and choose M/C nodes in the vicinity of each seed by expanding from it.
- **Real-world POIs**: We use the real-word data from [25], which is extracted from OpenStreetMap (OSM)[2]. We only show the POIs within NW since our default road network is NW (see Table 5). We choose FF and PO POIs as our P since their densities are equal to our default density (i.e., 0.001). Note that there is no any type of POIs whose size is equal to our default size (i.e., 128), and hence we select the POIs whose sizes are near to 128. In this way, we choose HOS and UNI POIs as our Q.

1. http://www.dis.uniroma1.it/challenge9/download.shtml

2. http://www.openstreetmap.org

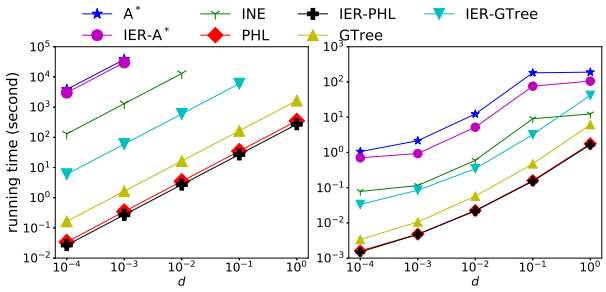
TABLE 5
Real world POIs in NW

Name	Description	# nodes	Density
PA	Parks	5,098	0.005
SC	Schools	4,441	0.004
FF	Fast Food	1,328	0.001
PO	Post Offices	1,403	0.001
HOT	Hotels	460	0.0004
HOS	Hospitals	258	0.0002
UNI	Universities	95	0.00009
CH	Courthouses	49	0.00005

As for the keyword information of P and Q, we embed textual descriptions extracted from OSM into POIs in Table 5, and each description has been simplified by the bag-of-words model.

By default, both P and Q are generated uniformly. In order to minimize the randomness, we average the results of algorithms over 100 queries. In addition, the performances of R-tree and G-tree are dependent on the value of fanout f. In our experiments, we set f to 4. As for G-tree, its performance also depends on the maximum number of points in a leaf node τ; we set τ to 64 (DE), 128 (ME, COL), 256 (NW, E), and 512 (CTR, USA) respectively.

As verified in Section 6.6.3, the running time of *sum*-FANN$_\mathcal{R}$ (resp., *sum*-KFANN$_\mathcal{R}$) is very close to that of *max*-FANN$_\mathcal{R}$ (resp., *max*-KFANN$_\mathcal{R}$) given the same input. Therefore, for universal methods, we only show the results of *max*-FANN$_\mathcal{R}$ and *max*-KFANN$_\mathcal{R}$ in terms of query efficiency.



(a) *GD* with different g_ϕ implementations (b) IER-*k*NN with different g_ϕ implementations

Fig. 3. Efficiency of algorithms implemented by different g_ϕ

6.2 Evaluation for Different Implementations of g_ϕ

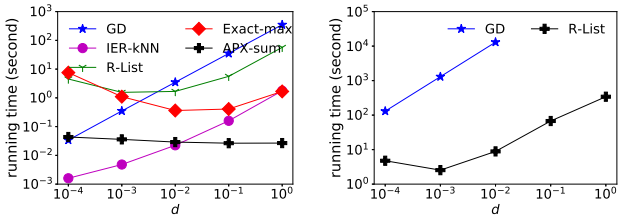
Recall Section 3.3, we mentioned a family of algorithms with different implementations of g_ϕ when discussing the IER-kNN framework. In fact, we can also generate generalized Dijkstra-based algorithms (denoted as *GD*) derived from the algorithm in Section 3.1, since the *Dijkstra* here can be regarded as INE and it can be replaced with other g_ϕ implementations.

We vary the density of P (i.e., d), and present the experimental results of *GD* and IER-kNN which are implemented by different g_ϕ routines. An important finding from Fig. 3(a) is that there is a linear (or sub-linear) relationship between the running time and density d. In addition, it can be found that GD with some implementations (e.g., A*, IER-A*, and INE) could be infeasible, if the underlying road network index is unavailable. Furthermore, by comparing Fig. 3(a) and Fig. 3(b), it can be seen that, given the same implementation of g_ϕ, IER-kNN outperforms *GD* by 1-3 orders of magnitude, demonstrating that IER-kNN framework is much more efficient. It can also be seen from Fig. 3 that PHL and IER-PHL always preform best while A* and IER-A* are the worst among these different implementations. Since PHL (or IER-PHL) is the most efficient implementation of g_ϕ, we choose PHL as the default implementation of g_ϕ in the latter experiments.

6.3 Evaluation for Query Efficiency

6.3.1 FANN$_\mathcal{R}$

We vary parameters d and ϕ, and the experimental results of different algorithms' running time (excluding the construction time of index) are shown in the following.



(a) All FANN$_\mathcal{R}$ algorithms (b) *R-List* and *GD*

Fig. 4. Efficiency when varying d

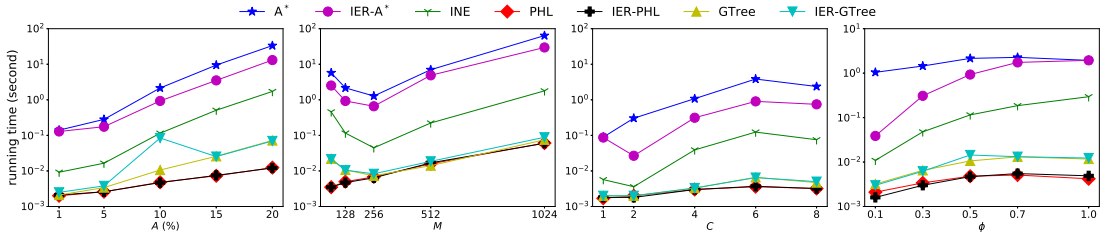
Varying d. We report the results by varying the density of P. Fig. 4(a) shows the efficiency of all FANN$_\mathcal{R}$ algorithms presented in this paper. With the increase of d, IER-PHL performs best at first, and *APX-sum* outperforms any other method when d is larger than 0.01. We can validate the stability of *APX-sum* when varying d, and this is because *APX-sum* depends much on Q instead of P. Both *APX-sum* and *Exact-max* have the tendency that they cost less time when d is larger. This is because the expanding routine from Q to P is faster when the data points is denser. The reason why *Exact-max* decreases first and then increases is due to the trade-off between the expanding overhead and the convenience brought by d. To be specific, a larger d will lead to a higher expanding overhead in general, while it will also make the termination condition to be fulfilled earlier.

On the other hand, we can also observe that *GD* performs even better than *R*-List and *Exact-max* when d is relatively small. This is mainly because default g_ϕ (i.e., PHL) shadows the advantages of *R*-List and *Exact-max*. In what follows, we will use two more evaluations to investigate it.

To begin with, we evaluate the performance of *GD* and *R*-List when g_ϕ is implemented with INE. Fig. 4(b) reports the results. We can see that *R*-List always outperforms *GD*, and the latter cannot be finished within a reasonable time if d is larger than 10^{-2}. There is not doubt that *GD* is infeasible when the road network index is unavailable (e.g., road networks in online games maps).

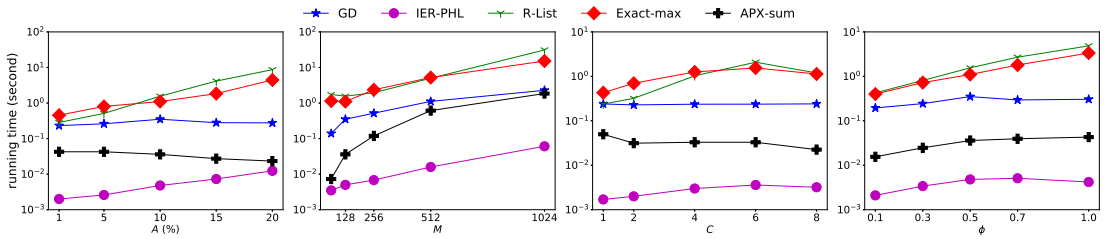
Next, we show the experimental results of *Exact-max* implemented by different g_ϕ routines in Table 6. We can see that different g_ϕ implementations have little influences on the efficiency of *Exact-max*. In other words, *Exact-max* performs quite well even if there is no underlying road network index. For example, if g_ϕ is implemented by A* and $d = 0.0001$, *Exact-max* outperforms *GD* by 2 orders of magnitude.

Varying ϕ. Now we study the effect of ϕ, i.e., the flexibility parameter. There is an obvious positive correlation with ϕ. This is reasonable because the larger ϕ means that more destinations



(a) Varying A (b) Varying M (c) Varying C (d) Varying ϕ

Fig. 5. Efficiency of IER-kNN when varying A, M, C and ϕ

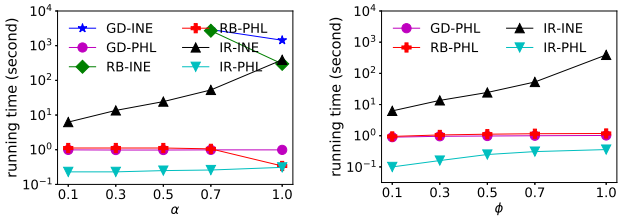


(a) Varying A (b) Varying M (c) Varying C (d) Varying ϕ

Fig. 6. Efficiency of all FANN$_\mathcal{R}$ algorithms when varying A, M, C and ϕ

TABLE 6
Efficiency of *Exact-max* with different g_ϕ (second)

g_ϕ \ d	0.0001	0.001	0.01	0.1	1
A*	7.26	1.24	0.65	0.69	2.05
IER-A*	7.07	1.05	0.47	0.50	1.79
INE	6.81	0.94	0.35	0.38	1.62
PHL	7.56	1.10	0.36	0.41	1.67
IER-PHL	7.59	1.07	0.36	0.40	1.68
GTree	6.77	0.92	0.34	0.37	1.60
IER-GTree	6.78	0.93	0.34	0.38	1.64



(a) Varying α (b) Varying ϕ

Fig. 7. Efficiency of KFANN$_\mathcal{R}$ when varying α and ϕ

need to be visited. For IER-kNN, we can find that A* benefits a lot from R-tree when ϕ is relatively small as shown in Fig. 5(d). This implies that larger ϕ incurs less pruning ability in terms of R-tree. We can also find that *R-List* and *Exact-max* are more effected by ϕ as shown in Fig. 6(d). In addition, the reason why *GD* performs better than *R-List* and *Exact-max* can be found above.

6.3.2 KFANN$_\mathcal{R}$

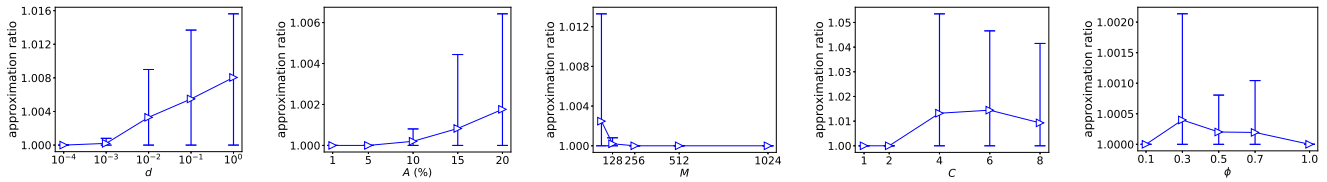
We use use default d and M to generate P and Q from the real word POIs in Table 5. As for the implementation of g_ϕ, we adopt the keyword-aware *INE* (mentioned in Section 3.1.2) and PHL, respectively. Beside the *GD*, we evaluate another baseline algorithm. It uses pure R-tree in IER-kNN framework, denoted as *RB* (R-tree baseline), and only considers spatial information (i.e., let textual similarity be 1) while accessing tree nodes. Note that we denote the keyword-aware IER-kNN framework as *IR* for simplicity. Hence, we have six possible combinations: *GD*-INE, *GD*-PHL; *RB*-INE; *RB*-PHL; *IR*-INE, *IR*-PHL. We show the results in Fig. 7.

Varying α. We report the results by varying the weight of road networks distances (recall Equation 2). As shown in Fig. 7(a), *IR*-PHL performs best among these algorithms. Note that *GD*-INE and *RB*-INE cannot be finished within a reasonable time (generally, they take more than one hour), and only *IR*-INE is feasible for any α without PHL index. We can also verify that R-tree has a quite poor pruning ability if $\alpha \neq 1$. With the increase of α, the weight of road network distance gains more importance, so the running time of *GD*-INE and *RB*-INE drops accordingly. On the other hand, we can find that any PHL version is insensitive to α. When $\alpha = 1.0$, the running time of *IR*-PHL (*IR*-INE) equals to that of *RB*-PHL (*RB*-INE), and this is because IR-tree and R-tree have the same pruning ability if only the spatial information is considered.

Varying ϕ. Since *GD*-INE and *RB*-INE cannot return the answer within a reasonable time (given the default $\alpha = 0.5$), we do not display them in Fig. 7(b). Like varying α, *IR*-INE is only feasible algorithm which can deal with any ϕ without PHL index. We can also find that R-tree would degenerate into *baseline* due to its ineffective pruning ability for textual information. With the increase of ϕ, the running time of *GD*-PHL, *RB*-PHL and *IR*-PHL is stable but has a slight increase. It is not surprising that *IR*-INE has an apparent increase since larger ϕ means more expansions.

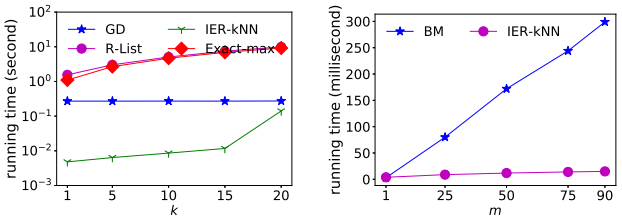
6.3.3 *Top-k and Multiple* FANN$_\mathcal{R}$

Due to the space constraint, we only show the results of k-FANN$_\mathcal{R}$ and m-FANN$_\mathcal{R}$ here.



(a) Varying d (b) Varying A (c) Varying M (d) Varying C (e) Varying ϕ

Fig. 8. Approximation quality of *APX-sum*



(a) Varying k (b) Varying m

Fig. 9. Efficiency of k-FANN$_{\mathcal{R}}$ and m-FANN$_{\mathcal{R}}$

k-FANN$_{\mathcal{R}}$. Fig. 9(a) shows the results. It is obvious that the query time will increase when k grows except for the *GD* algorithm. The stability of *GD* is owned to the fact that it mainly depends on P and the choice of g_ϕ's implementation, while other factors rarely have influence on it. In addition, similar to Fig. 4(b), this figure also shows us that *GD* is even better than *R-List* and *Exact-max*. The underlying reason is the same as to that mentioned in Section 6.3.1.

m-FANN$_{\mathcal{R}}$. The experimental results are shown in Fig. 9(b). The baseline algorithm is the m times FANN$_{\mathcal{R}}$ queries, and we denote it as *BM*. Note that IER-kNN here means the implementation of Algorithm 4. It is obvious that *BM* is nearly linear proportional to m since it requires m-passes over P. In contrast, we can find that IER-kNN is quite stable and this is because the advantage of *one-pass* over P can boost the query efficiency.

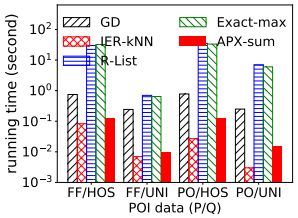
6.4 Approximation Quality of APX-sum

Fig. 8 reports the approximation quality of *APX-sum* algorithm. It can be seen that the approximation ratio of *APX-sum* is always less than 1.2 in all experimental cases (i.e., vary d, A, M, C and ϕ). The y-error in error bars is the standard deviation. Given its excellent approximate quality and other features (e.g., efficient even if index-free), *APX-sum* is a promising algorithm when an approximate answer is acceptable or it is difficult to build a road network index.

6.5 Real World Data of FANN$_{\mathcal{R}}$

Query Efficiency. As shown in Fig. 10(a), the performance of different FANN$_{\mathcal{R}}$ methods on real world POIs has the similar characteristics with the evaluation using synthetic data. For example, IER-kNN performs best among all these methods, and *GD* is even better than *R-List* and *Exact-max*. The reasons for these phenomena are consistent to that for synthetic data.

Approximation Quality of *APX-sum*. Fig. 10(b) reports the results. We can find that the *APX-sum* also has a very good approximation quality on the real world POIs. The approximation ratio is always less than 1.1 for all cases. This further demonstrates the effectiveness of our *APX-sum* algorithm.



(a) Efficiency

	Approximation ratio
FF/HOS	1.003
FF/UNI	1.0
PO/HOS	1.004
PO/UNI	1.076

(b) Approximation quality

Fig. 10. Evaluation of FANN$_{\mathcal{R}}$ on real world data

6.6 Other Results

6.6.1 Query Efficiency When Varying A, M, C

Varying A. Now we study the efficiency with different *coverage ratios* of Q. We show the results in Fig. 5(a) and Fig. 6(a). Clearly, *APX-sum* is stable when we vary A. This validates that *APX-sum* has little dependence on Q's sparsity. As for *R-List* and *Exact-max* algorithms, their performances are bad if Q is sparse. This is because a sparer Q often leads to a slower expanding from Q to P. We can also find that the *GD* algorithm is stable for different coverage ratios. This is not surprising, as the *GD* method depends much on P. Finally, *R-List* and *Exact-max* are not better than *GD* here, and this is also verified by Fig. 4(a) when $d = 0.001$.

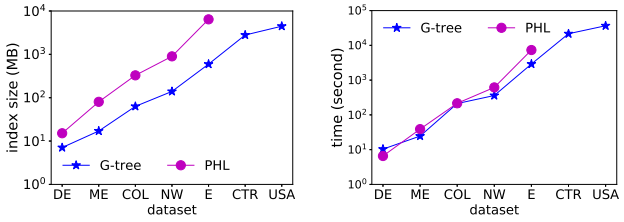
Varying M. We study the efficiency when varying M (i.e., the size of Q). The results are shown in Fig. 5(b) and Fig. 6(b). We can clearly find that *APX-sum* increases with the increase of M, and this further verifies that *APX-sum* depends much on the size of Q. As for IER-kNN methods shown in Fig. 5(b), they have the tendency that the larger M leads to the worse efficiency. Note that the running time decreases first (from $M = 64$ to $M = 256$) for most IER-kNN methods. This is due to the trade-off between M and the sparsity of Q. To be specific, given a coverage ratio of Q, the smaller M incurs a larger sparsity.

Varying C. We evaluate the effect of C (i.e., the size of clusters). Clearly, the larger C leads to the worse performance in general, and this tendency is more obvious for "expanding-based" methods in Fig. 5(c). When C is larger enough, the performance will be stable and the running time will approximate to the value when Q is generated uniformly. For example, the running time of IER-A* is 2.16 seconds if Q is generated uniformly, while the cost is 2.37 seconds when $C = 8$. In addition, as shown in Fig. 6(c), *R-List* and *Exact-max* are more affected by C due to their similar mechanisms.

6.6.2 Index Cost

Here we measure the construction time and size of the road networks' index structures used in our algorithms. The different index techniques have been presented in Table 2.

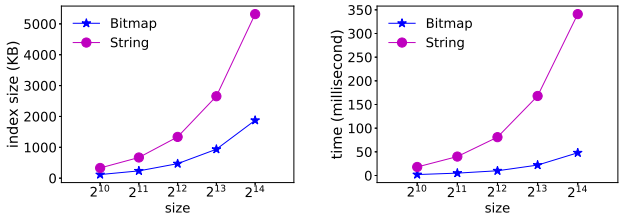
G-tree and PHL. Fig. 11 shows the index size and the construction time of G-tree and PHL for different datasets. Generally, G-tree costs less storage than PHL. Note that PHL can only build indexes for the first 5 datasets, before exceeding the memory capacity. This experimental result indicates that for these two indexes, G-tree could be much suitable for very large road networks (e.g. USA), since PHL fails to build index for CTR and USA in a single commodity machine. One can conclude that, as for choosing between G-tree and PHL to build a road network index, the major considerations are the running efficiency and memory capacity rather than the index construction time. This finding is essentially consistent to that in [25].



(a) Index size of G (b) Construction time of G

Fig. 11. Index cost of different road networks

Different Keyword Encodings. We use the string set itself and the roaring bitmap to encode the textual information, and denote them as *String* and *Bitmap* respectively. The comparison results are shown in Fig. 12. Firstly, we can find both the index size and the construction time are proportional to the dataset size. Secondly, it is not surprising that *Bitmap* costs less space than *String* in terms of index size. Finally, *Bitmap* is also superior to *String* in terms of construction time, and this is because the *union* operation in the former is more efficient than that in the latter, when constructing the corresponding index.



(a) Index size (b) Construction time

Fig. 12. IR-tree cost w.r.t different textual encodings

6.6.3 Evaluation for Different Aggregate Functions

TABLE 7
Running time of FANN$_{\mathcal{R}}$ (second)

max-FANN$_{\mathcal{R}}$	3	7	4	6	4	4	20	23	5	5
sum-FANN$_{\mathcal{R}}$	3	7	3	7	5	5	22	25	6	5

TABLE 8
Running time of KFANN$_{\mathcal{R}}$ (second)

max-KFANN$_{\mathcal{R}}$	0.31	0.29	0.28	0.3	0.31	0.29	0.31	0.3	0.29	0.29
sum-KFANN$_{\mathcal{R}}$	0.25	0.27	0.18	0.21	0.27	0.31	0.21	0.23	0.28	0.29

As noted in Section 6.1, given d, A, M, C, ϕ and α, as for universal methods, we only report results for $g = max$. Instead, in Table 7, given the default inputs, we display the first 10 raw running time records of IER-kNN for FANN$_{\mathcal{R}}$. Similarly, we also show the first 10 raw running time records of IER-kNN for KFANN$_{\mathcal{R}}$ in Table 8. It can be seen from these two evaluations (i.e., Tables 7 and 8) that there is a trivial difference between the corresponding results. This essentially implies that different aggregate functions (i.e., g) have little influences on running time.

7 Conclusion

In this paper, we studied an interesting problem of flexible aggregate nearest neighbor queries on road networks (FANN$_{\mathcal{R}}$) and its keyword-aware variant (KFANN$_{\mathcal{R}}$). We proposed a series of universal methods to solve this problem, including a Dijkstra-based algorithm, *R-List* and IER-kNN algorithm framework. Combined with the state-of-the-art shortest path and keyword encoding techniques, the proposed algorithms can achieve excellent performance. On the other hand, our specific approaches for FANN$_{\mathcal{R}}$ (*Exact-max* and *APX-sum*) sacrifice some generalities, but they are easier to implement and much more efficient than those universal approaches especially when the road networks are index-free. Also, we successfully adapted most of the proposed algorithms to answer the extended k-FANN$_{\mathcal{R}}$ (k-KFANN$_{\mathcal{R}}$) and m-FANN$_{\mathcal{R}}$ (m-KFANN$_{\mathcal{R}}$) queries. Finally, we conducted a comprehensive experiments, validating the efficiency and effectiveness of our proposed approaches.

Acknowledgments

This work (Bin Yao) was supported by the NSFC (U1636210, 61729202, 61922054, 61872235, 61832017), and The National Key Research and Development Program of China (2018YFC1504504). Ma was supported in part by the NSFC (61925203, 61421003) and Beijing Advanced Innovation Center for Big Data and Brain Computing. Gao was supported by NSFC (61872238) and the Shanghai Science and Technology Fund (17510740200). This work was also supported by the NSFC (U1811264, 61972425, 61772570), Pearl River S&T Nova Program of Guangzhou (201806010056), and Guangdong NSF for Distinguished Young Scholar (2018B030306025).

References

[1] D. Papadias, Q. Shen, Y. Tao, and K. Mouratidis, "Group nearest neighbor queries," in *ICDE*. IEEE, 2004, pp. 301–312.
[2] D. Papadias, Y. Tao, K. Mouratidis, and C. K. Hui, "Aggregate nearest neighbor queries in spatial databases," *ACM TODS*, vol. 30, no. 2, pp. 529–576, 2005.
[3] F. Li, B. Yao, and P. Kumar, "Group enclosing queries," *IEEE TKDE*, vol. 23, no. 10, pp. 1526–1540, 2011.
[4] M. L. Yiu, N. Mamoulis, and D. Papadias, "Aggregate nearest neighbor queries in road networks," *IEEE TKDE*, vol. 17, no. 6, pp. 820–833, 2005.
[5] D. Yan, Z. Zhao, and W. Ng, "Efficient algorithms for finding optimal meeting point on road networks," *PVLDB*, vol. 4, no. 11, 2011.
[6] M. Safar, "Group k-nearest neighbors queries in spatial network databases," *JGS*, vol. 10, no. 4, pp. 407–416, 2008.

[7] L. Zhu, Y. Jing, W. Sun, D. Mao, and P. Liu, "Voronoi-based aggregate nearest neighbor query processing in road networks," in *SIGSPATIAL*. ACM, 2010, pp. 518–521.

[8] Y. Li, F. Li, K. Yi, B. Yao, and M. Wang, "Flexible aggregate similarity search," in *SIGMOD*. ACM, 2011, pp. 1009–1020.

[9] F. Li, K. Yi, Y. Tao, B. Yao, Y. Li, D. Xie, and M. Wang, "Exact and approximate flexible aggregate similarity search," *VLDBJ*, vol. 25, no. 3, pp. 317–338, 2016.

[10] S. Ahmad, R. Kamal, M. E. Ali, J. Qi, P. Scheuermann, and E. Tanin, "The flexible group spatial keyword query," in *Australasian Database Conference*. Springer, 2017, pp. 3–16.

[11] J. M. Ponte and W. B. Croft, "A language modeling approach to information retrieval," in *SIGIR*. ACM, 1998, pp. 275–281.

[12] R. Zhong, G. Li, K.-L. Tan, and L. Zhou, "G-tree: An efficient index for knn search on road networks," in *CIKM*. ACM, 2013, pp. 39–48.

[13] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[14] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, 1968.

[15] N. Jing, Y.-W. Huang, and E. A. Rundensteiner, "Hierarchical encoded path views for path query processing: An optimal model and its performance evaluation," *IEEE TKDE*, vol. 10, no. 3, pp. 409–432, 1998.

[16] S. Jung and S. Pramanik, "An efficient path computation model for hierarchically structured topographical road maps," *IEEE TKDE*, vol. 14, no. 5, pp. 1029–1046, 2002.

[17] T. Akiba, Y. Iwata, K.-i. Kawarabayashi, and Y. Kawata, "Fast shortest-path distance queries on road networks by pruned highway labeling," in *ALENEX*. SIAM, 2014, pp. 147–154.

[18] H. Bast, S. Funke, and D. Matijević, "Transit: ultrafast shortest-path queries with linear-time preprocessing," in *9th DIMACS Implementation Challenge—Shortest Path*, 2006.

[19] R. Geisberger, P. Sanders, D. Schultes, and D. Delling, "Contraction hierarchies: Faster and simpler hierarchical routing in road networks," in *Experimental Algorithms*. Springer, 2008, pp. 319–333.

[20] K. C. Lee, W.-C. Lee, B. Zheng, and Y. Tian, "Road: A new spatial object search framework for road networks," *IEEE TKDE*, vol. 24, no. 3, pp. 547–560, 2012.

[21] M. Kolahdouzan and C. Shahabi, "Voronoi-based k nearest neighbor search for spatial network databases," in *VLDB*, 2004, pp. 840–851.

[22] R. Zhong, G. Li, K. Tan, L. Zhou, and Z. Gong, "G-tree: An efficient and scalable index for spatial search on road networks," *IEEE TKDE*, vol. 27, no. 8, pp. 2175–2189, 2015.

[23] G. R. Hjaltason and H. Samet, "Distance browsing in spatial databases," *ACM TODS*, vol. 24, no. 2, pp. 265–318, 1999.

[24] N. Roussopoulos, S. Kelley, and F. Vincent, "Nearest neighbor queries," in *SIGMOD record*, vol. 24, no. 2. ACM, 1995, pp. 71–79.

[25] T. Abeywickrama, M. A. Cheema, and D. Taniar, "K-nearest neighbors on road networks: a journey in experimentation and in-memory implementation," *PVLDB*, vol. 9, no. 6, pp. 492–503, 2016.

[26] M. Qiao, L. Qin, H. Cheng, J. X. Yu, and W. Tian, "Top-k nearest keyword search on large graphs," *PVLDB*, vol. 6, no. 10, pp. 901–912, 2013.

[27] T. Guo, X. Cao, and G. Cong, "Efficient algorithms for answering the m-closest keywords query," in *SIGMOD*. ACM, 2015, pp. 405–418.

[28] D.-W. Choi, J. Pei, and X. Lin, "Finding the minimum spatial keyword cover," in *ICDE*. IEEE, 2016, pp. 685–696.

[29] G. Cong, C. S. Jensen, and D. Wu, "Efficient retrieval of the top-k most relevant spatial web objects," *PVLDB*, vol. 2, no. 1, pp. 337–348, 2009.

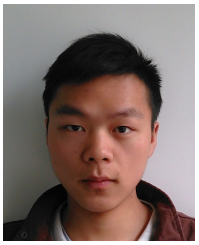
[30] J. Shi, D. Wu, and N. Mamoulis, "Top-k relevant semantic place retrieval on spatial rdf data," in *SIGMOD*. ACM, 2016, pp. 1977–1990.

[31] B. Yao, Z. Chen, X. Gao, S. Shang, S. Ma, and M. Guo, "Flexible aggregate nearest neighbor queries in road networks," in *ICDE*. IEEE, 2018, pp. 761–772.

[32] N. Bruno and H. Wang, "The threshold algorithm: From middleware systems to the relational engine," *IEEE TKDE*, vol. 19, no. 4, pp. 523–537, 2007.

[33] D. Xie, F. Li, and J. M. Phillips, "Distributed trajectory similarity search," *PVLDB*, vol. 10, no. 11, pp. 1478–1489, 2017.

[34] M. E. Ali, E. Tanin, P. Scheuermann, S. Nutanong, and L. Kulik, "Spatial consensus queries in a collaborative environment," *TSAS*, vol. 2, no. 1, p. 3, 2016.



Zhongpu Chen is currently a PhD candidate in Department of Computer Science and Engineering at Shanghai Jiao Tong University, China. His research interests include distributed analytics systems, big data, spatial databases, and indexing techniques for multi-dimensional data.

Bin Yao is an associate professor in Department of Computer Science and Engineering at Shanghai Jiao Tong University. He obtained his PhD in computer science from the Department of Computer Science at Florida State University in 2011. His research interests include management and indexing of large databases, query processing in spatial and multimedia databases, string and keyword search, and scalable data analytics.

Zhi-Jie Wang received the PhD degree in computer science from Shanghai Jiao Tong University, and did a postdoc at The Hong Kong Polytechnic University. He is currently an Associate Professor at the College of Computer Science, Chongqing University (CQU). Before joining CQU, he was a Research Associate Professor at the Sun Yat-Sen University. His research interests include big data, artificial intelligence, data mining, and distributed systems. He is a member of CCF, IEEE, and ACM.

Xiaofeng Gao is an associate professor of Computer Science and Engineering, Shanghai Jiao Tong University. She received the PhD degree from the University of Texas at Dallas, USA, in 2010. Her research areas are data engineering, database management, wireless network, and optimization algorithms.

Shuo Shang is a research scientist at Extreme Computing Research Center, King Abdullah University of Science and Technology. He obtained his Ph.D. from The University of Queensland in 2012. His research interests include big data management, spatial-temporal databases, urban computing, spatial trajectory computing, and location based services.

Shuai Ma is a professor at School of Computer Science and Engineering, Beihang University. He obtained his PhD degree from Peking University in 2004 and The University of Edinburgh in 2010. His research interests include database theory and systems, graph and social data analysis, and data cleaning.

Minyi Guo is a Zhiyuan Chair Professor at Shanghai Jiao Tong University. He received the PhD degree in information science from University of Tsukuba in 1998. His research interests include parallel and distributed processing, parallelizing compilers, cloud computing, pervasive computing; software engineering, embedded systems, green computing, and wireless sensor networks.