IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 6, 2019, accepted July 17, 2019, date of publication July 24, 2019, date of current version August 9, 2019.

*Digital Object Identifier 10.1109/ACCESS.2019.2930753*

# Efficient Similarity Search on Quasi-Metric Graphs

**TIANMING ZHANG[1], YUNJUN GAO[1,3], (Member, IEEE), LU CHEN[2], GUANLIN CHEN[1,3], AND SHILIANG PU[4]**

[1]College of Computer Science, Zhejiang University, Hangzhou 310027, China
[2]Department of Computer Science, Aalborg University, 9100 Aalborg, Denmark
[3]Zhejiang University City College, Hangzhou 310015, China
[4]Hangzhou Hikvision Digital Technology Company Ltd., Hangzhou 310052, China

Corresponding author: Yunjun Gao (gaoyj@zju.edu.cn)

This work was supported in part by the National Key R&D Program of China under Grant No. 2018YFB1004003, the 973 Program of China under Grant No. 2015CB352502, the NSFC under Grants No. 61522208, the NSF-Zhejiang Joint Fund under Grant No. U1609217, the Science & Technology Development Project of Hangzhou (China) under Grant No. 20162013A08, and the ZJU-Hikvision Joint Project.

**ABSTRACT** Similarity search in metric spaces finds similar objects to a given object, which has received much attention as it is able to support various data types and flexible similarity metrics. In real-life applications, metric spaces might be combined with graphs, resulting in geo-social network, citation graph, social image graph, to name but a few. In this paper, we introduce a new notion called *quasi-metric graph* that connects metric data using a graph, and formulate similarity search on quasi-metric graphs based on the combined similarity metric considering both the metric data similarity and graph similarity. We propose two simple efficient approaches, the *best-first* method and the *breadth-first* method, which traverse the quasi-metric graph following the best-first and the breadth-first paradigms, respectively, and utilize the triangle inequality to prune unnecessary evaluation. Extensive experiments with three real datasets demonstrate, compared with several baseline methods, the effectiveness and efficiency of our proposed methods.

**INDEX TERMS** Algorithm, graph, metric space, query processing, similarity search.

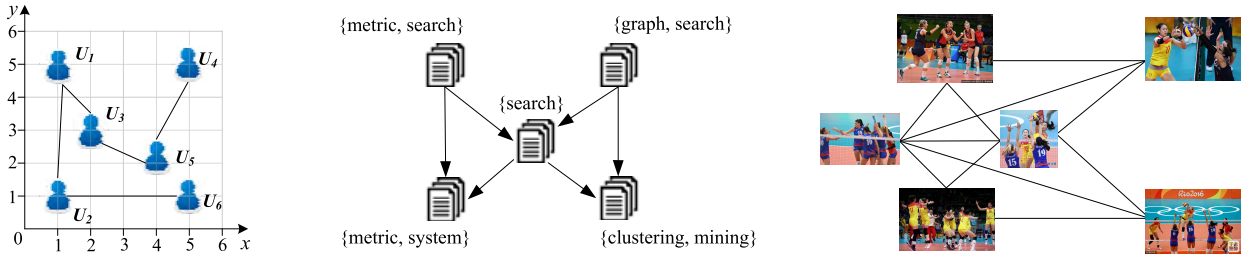
## I. INTRODUCTION

Given a query object $q$ and an object set $S_O$, a similarity query in metric spaces finds objects from $S_O$ similar to $q$ under a certain similarity metric. Considering that metric spaces can support a wide range of data types and similarity metrics, metric similarity queries are useful in GIS, information retrieval, multimedia recommendation, etc. In real-life applications, metric spaces might be combined with graphs, i.e., the relationships between objects in a metric space can be modeled as a graph, resulting in geo-social network, citation graph, social media graph, to name but a few. Motivated by this, we introduce a new notion called *quasi-metric graph* (see Definition 1 for details) that connects metric data using a graph and investigate similarity search (including range query and $k$ nearest neighbor ($k$NN) search) on quasi-metric graphs. Here, we need to consider the metric data similarity and the graph similarity simultaneously. In the following, we give three representative examples.

*Application 1 (Geo-Social Network): As illustrated in Fig. 1(a), a static geo-social network is an undirected graph where each vertex denotes a user and each edge indicates that two connected users are friends. The geo-social network allows users to capture their geographic locations and share them in the social network via an operation called check-in. Here, similarity search can help a user to find candidates who take part in an event. In this case, candidates are friends nearest to the query user, i.e., both the social and the geographic distances are considered.*

*Application 2 (Citation Graph): As depicted in Fig. 1(b), a static citation graph is a directed graph in which each vertex represents a publication and each edge means a citation from the current publication to another. Here, similarity search can help users to find related publications to a specified one. In this case, both the similarity (e.g., Jaccard distance, tf-idf) between the features of the publications and the*

The associate editor coordinating the review of this manuscript and approving it for publication was Muhammad Asif Naeem.




(a) Geo-social network (b) Citation graph (c) Social image graph

**FIGURE 1.** **Example of quasi-metric graphs.**

*shortest path distance in the citation graph are considered for finding related publications.*

*Application 3 (Social Image Graph): As shown in Fig. 1(c), a static social image graph is an undirected graph where each vertex denotes an image and each edge indicates that two connected images are tagged or commented by the same user. Nowadays, the rapid growth of social image websites (e.g., Flickr, Instagram) allows improved image searching to use social information. Here, similarity search can help users to find the images that they might be interested in. In this case, both the similarity (e.g., $L_p$-norm, SIFT) between the features of images and the social distances are considered for image recommendation.*

Most of the existing efforts on similarity search only consider metric spaces [1]–[3] or traditional graphs [4], [5], separately. Nevertheless, they are inefficient to support combined quasi-metric graphs, which is also demonstrated in our experiments. Recently, attribute graphs were introduced [6]–[9], which differ from quasi-metric graphs as follows: (i) Vertices in an attribute graph may have any type of attribute data, while vertices in a quasi-metric graph only associate with metric data, where the triangle inequality can be employed to accelerate the search. (ii) Existing approaches on attribute graphs either transform attributes into parts of graphs (e.g., an edge is added if two vertices have the same attributes [8], [9]) or utilize a probability model for query processing, which cannot be used to solve our studied problem, because we utilize a combined similarity metric (i.e., considering both metric data similarity and graph similarity) with a parameter to control the weights of metric data similarity and graph similarity. In addition, some studies aim at handling specific quasi-metric graphs, e.g., geo-social network, citation graph, and social image graph. Nonetheless, the algorithms designed for those particular graphs cannot tackle generic quasi-metric graphs.

A naïve solution for similarity search on quasi-metric graphs is to compute the metric data similarity and graph similarity between all the vertices in the graph and the query vertex. Unfortunately, it is inefficient due to a huge amount of superfluous metric data and graph similarity computation. To support efficient similarity search on quasi-metric graphs, two challenges have to be addressed. The first challenge is *how to reduce the number of metric data similarity computations.* Distance computation is one of the most expensive operations in metric spaces that we would like to avoid. As a result, we present several filtering techniques based on the triangle inequality. The second challenge is *how to reduce the number of graph similarity computations.* The graph similarity can be calculated as the shortest path distance with the time complexity $O(|V|^2)$ ($|V|$ denotes the number of vertices in the graph), which is costly. To avoid unnecessary graph similarity computations, we traverse the graph in a best-first or breadth-first paradigm, so that we could obtain all the needed graph similarities by traversing graph only once (i.e., only one shortest path distance computation is needed). In brief, the key contributions of this paper are summarized as follows:

- We introduce the notion so-called *quasi-metric graph* that connects metric data using a graph and explore similarity search on quasi-metric graphs with a simple but effective combined similarity metric.
- We propose two efficient approaches that follow the best-first and breadth-first paradigms to answer similarity search on quasi-metric graphs, in which several filtering techniques are developed based on the triangle inequality to boost search.
- We conduct extensive experiments using three real datasets, compared with three baseline algorithms, to verify the effectiveness and efficiency of our proposed algorithms.

The rest of the paper is organized as follows. Section II reviews related work. Section III formalizes our problem, and presents several pruning and validating lemmas. Section IV elaborates three baseline methods. Section V proposes two efficient approaches for supporting similarity search on quasi-metric graphs. Considerable experimental results and our findings are reported in Section VI. Finally, Section VII concludes the paper with some directions for future work.

## II. RELATED WORK

In this section, we overview the existing work on similarity search in metric spaces and on graphs, respectively.

### A. SIMILARITY SEARCH IN METRIC SPACES

Similarity search (including range query and $k$ nearest neighbor ($k$NN) retrieval) in metric spaces has been surveyed well in the literature [1]–[3]. More specifically,

two broad categories of metric indexes exist that aim to accelerate similarity search in metric spaces, namely, compact partitioning methods and pivot-based approaches. The former methods partition the space as compact as possible, they try to prune unqualified partitions during search. BST [10], [11], GHT [12], [13], SAT [14], M-tree family [15]–[17], D-index [18], eDindex [19], LC family [20]–[22], and BP [23] all belong to this category. Methods of the other category store precomputed distances from every object in the database to a set of pivots, they utilize the distances and triangle inequality to prune unqualified objects during search. BKT [24], AESA [25], [26], EP [27], FQT [28], VPT [29], [30], and Omni-family [31] all belong to this category. Recently, hybrid methods that combine compact partitioning with the use of pivots have presented. The PM-tree [32] utilizes cut-regions defined by pivots to improve query processing on the M-tree. The M-Index [33] generalizes the iDistance technique for metric spaces, which compacts the objects by using precomputed distances to their closest pivots. The SPB-tree [34] integrates the pivot-mapping method with the space-filling curve technique to further improve efficiency.

Since the shortest path distance (i.e., the graph distance) on the graph with non-negative edge weights satisfies the triangle inequality but does not meet symmetry when the graph is directed, the quasi-metric graph using the combined similarity metric (containing metric data similarity and graph similarity) can be regarded as a general quasi-metric space [35]. Note that, techniques in general metric spaces are usually based on the triangle inequality and thus can be used for quasi-metric graphs. However, the above similarity search algorithms designed for metric spaces still need combined similarity computations (including metric data similarity and graph similarity computations) for unpruned objects, incurring lots of unnecessary graph similarity computations, which is also confirmed by our experiments.

### B. SIMILARITY SEARCH ON GRAPHS

Many measurements are presented to define the similarity between two vertices in graphs, e.g., the shortest path distance [36]–[38], SimRank [39], [40], Personalized PageRank (PPR) [41], [42], to name just a few. The shortest path distance is an intuitive graph distance to measure how close one vertex is to another. SimRank is a measurement that says two objects are considered to be similar if they are referenced by similar objects. PPR utilizes random walk to estimate each vertex's similarity score w.r.t. a query vertex. For sake of simplicity, we utilize the shortest path distance to define graph similarity in this paper, while other similarity metrics would be investigated as a direction of our future work.

There are many previous studies on addressing the problem of shortest path distance computation. A landmark-based method [38] for shortest path distance estimation preselects a subset of vertices as landmarks and precomputes the shortest path distances between each vertex in the graph and those landmarks, and thus, the shortest path distance between a pair of vertices can be estimated by combining the precomputed distances. Pruned landmark labeling (PLL) [36] is one of up-to-date exact algorithms. PLL precomputes distance labels for all vertices by performing pruned breadth-first search from every vertex, and then, a shortest path distance query for any pair of vertices can be exactly computed using the distance labels. Nevertheless, either aforementioned algorithms designed for shortest path distance computation or top-$k$ algorithms [4], [5] designed for similarity search on traditional graphs cannot be directly applied for our studied problem, because the metric data similarities between vertices are ignored.

Recently, attributed graphs are proposed [6]–[9], in which every vertex has a set of attributes. They differ from quasi-metric graphs. Specifically, vertices in an attribute graph can have any type of attribute data, whereas vertices in a quasi-metric graph associate with metric data where the triangle inequality can be utilized to accelerate the search. Most existing efforts on attributed graphs either transform the attributes into parts of graphs (e.g., building an additional edge to connect two vertices that have the same attributes [9], and then weighting the edge between two vertices by using the attribute similarity [43]) or use a probability model [7] to model an attribute graph. However, in this paper, we utilize a combined similarity metric, and thus, existing methods used for attribute graphs cannot solve our problem.

In addition, studies on specific quasi-metric graphs [8], [44]–[47], such as geo-social network [44], [45], citation graph [46], and social media graph [47], have also been investigated. Nonetheless, they are designed for particular quasi-metric graphs, i.e., techniques used to improve search utilize the characteristics of the specific quasi-metric graphs, and hence, they cannot be applied for the general case.

## III. PROBLEM FORMULATION

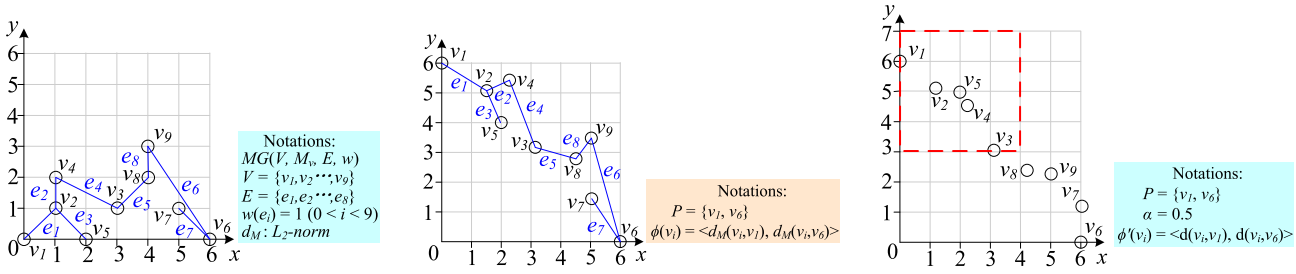
In this section, we first present the definition of quasi-metric graph, and then, we formalize the range query and $k$NN search based on the quasi-metric graph. Finally, we present several pruning and validating lemmas to accelerate similarity search on the quasi-metric graph. Table 1 summarizes the symbols frequently used throughout this paper.

### A. QUASI-METRIC GRAPH

Before defining quasi-metric graph, we first review metric space and graph, respectively.

**Metric space [35].** A metric space is denoted as a tuple $(M, d_M)$, in which $M$ is an object domain and $d_M$ is a metric distance function to measure similarity between two objects in $M$. The metric distance function $d_M$ has four properties: (1) *symmetry*: $d_M(q, o) = d_M(o, q)$; (2) *non-negativity*: $d_M(q, o) \geq 0$; (3) *identity*: $d_M(q, o) = 0$ iff $q = o$; and (4) *triangle inequality*: $d_M(q, o) \leq d_M(q, p) + d_M(p, o)$.

Quasi-metric [35] is similar to metric, the only difference between a metric and a quasi-metric is that a quasi-metric does not possess the symmetry axiom (in the case $d(q, o) \neq d(o, q)$ is allowed).




(a) Quasi-metric graph (b) Mapped metric space (c) Mapped quasi-metric graph

**FIGURE 2.** Illustration of pruning quasi-metric graph.

**TABLE 1.** Symbols and description.

| Notation | Description |
|---|---|
| $MG(V, M_v, E, w)$ | a quasi-metric graph with a set $V$ of vertices, a set $M_v$ of vertex-specific metric data, a set $E$ of edges, and an edge weight function $w$ |
| $\|V\|$ or $\|E\|$ | the number of vertices or edges |
| $r$ | a search radius |
| $q$ | a query vertex |
| $k$ | the number of the objects requested (for $k$NN query) |
| $u, v$ | a vertex |
| $v.data$ | the metric data associated with a vertex $v$ |
| $e$ | an edge |
| $d_G(u, v)$ | the shortest path distance (i.e., graph distance) between vertices $u$ and $v$ |
| $d_M(u, v)$ | the metric distance between $u$ and $v$ |
| $d(u, v)$ | the combined distance between vertices $u$ and $v$ used in the quasi-metric graph |
| $P$ | a set of pivots $p$ |
| $ud_M(u, v)$ or $ld_M(u, v)$ | the upper bound or lower bound of $d_M(u, v)$ |
| $ud(u, v)$ or $ld(u, v)$ | the upper bound or lower bound of $d(u, v)$ |
| $ud_G(u, v)$ | the upper bound of $d_G(u, v)$ |
| $\alpha$ | the parameter for controlling the weights between $d_M$ and $d_G$ |
| $RQ(q, r)$ | the result set of a range query on the quasi-metric graph with a search radius $r$ |
| $kNNQ(q, k)$ | the result set of a $k$ nearest neighbor ($k$NN) query on the quasi-metric graph w.r.t. $q$ |
| $ND_k$ | the combined distance from a query vertex to its $k$-th nearest neighbor |

**Graph.** A graph is denoted as $G(V, E, w)$, where $V$ is a set of vertices, $E$ is a set of edges, and $w$ is an edge weight function denoted as $w : E \rightarrow \mathbb{R}^+$. In particular, $w(e)$ represents the weight of an edge $e$, and $w(e)$ is equal to 1 for unweighted graphs. In this paper, we assume that the graph is static and we use the shortest path distance $d_G(u, v)$ to measure graph similarity between two vertices $u$ and $v$ in the graph. If vertices $u$ and $v$ are disconnected in the graph, then $d_G(u, v) = \infty$. Let $P_s(u, v) = \{u, e_1, t_1, e_2, t_2, \cdots, e_m, t_m, e_{m+1}, v\}$ be the shortest path between $u$ and $v$, where $\{u, t_1, t_2, \cdots, t_m, v\}$ and $\{e_1, e_2, \cdots, e_m, e_{m+1}\}$ are the sequenced vertices and edges respectively in the shortest path, then $d_G(u, v) = \sum_{i=1}^{m+1} w(e_i)$. Note that, $d_G(u, v)$ satisfies all the properties defined in the metric space except for symmetry when the graph is directed, and hence, it is a quasi-metric [35] distance.

By combining the metric space and the graph, we introduce the quasi-metric graph as follows.

*Definition 1 (Quasi-Metric Graph): A quasi-metric graph is denoted as $MG(V, M_v, E, w)$, where $V$ is a set of vertices, $M_v = \{v.data \mid \forall v \in V\}$ is a set of vertex-specific metric data, i.e., each vertex $v$ in $V$ associates with metric data $v.data$, $E$ is a set of edges, and $w$ is an edge weight function. We define a combined distance $d(u, v) = \alpha \times d_G(u, v) + (1-\alpha) \times d_M(u, v)$ to measure the similarity between two vertices $u$ and $v$ in the quasi-metric graph, the parameter $\alpha$ ($0 < \alpha < 1$) is employed to control the weights between graph similarity and metric data similarity. Obviously, the combined distance $d(u, v)$ is a quasi-metric distance, since $d_M(u, v)$ is a metric distance and $d_G(u, v)$ is a quasi-metric distance. Thus, the sum of their linear varieties must be the quasi-metric distance.*

*Example 1: Consider a quasi-metric graph example $MG(V, M_v, E, w)$, i.e., a geo-social network, in Fig. 2(a), where $V = \{v_1, v_2, \cdots, v_9\}$ denotes a set of users, $E = \{e_1, e_2, \cdots, e_8\}$ represents a set of friendships between the users, $w(e)$ is equal to 1 for any edge $e \in E$, and metric data $v.data$ associated with each vertex $v$ denotes the corresponding location of $v$ (e.g., $v_2.data = (1, 1)$). Here, $L_2$-norm is used as $d_M$ to measure the distance between locations of users, and the shortest path distance is used as $d_G$ to measure the relationship between users.*

### B. SIMILARITY SEARCH ON QUASI-METRIC GRAPHS

Based on the quasi-metric graph, we formally define similarity search, including range query and $k$ nearest neighbor ($k$NN) query, as stated below.

*Definition 2 (Range Query on Quasi-Metric Graph): Given a quasi-metric graph $MG(V, M_v, E, w)$, a query vertex $q$, and a search radius $r$, a* range query on quasi-metric graph *finds the vertices $v$ in $V$ that are within the distance $r$ to $q$, i.e., $RQ(q, r) = \{v \mid (v \in V) \wedge (d(q, v) \leq r)\}$.*

*Definition 3 (kNN Query on Quasi-Metric Graph): Given a quasi-metric graph $MG(V, M_v, E, w)$, a query vertex $q$, and an integer $k$, a $k$NN query on the quasi-metric graph finds $k$ vertices in $V$ that are most similar to $q$, sorted in ascending order of their combined distances w.r.t. $q$, i.e., $kNNQ(q, k) = \{U \mid (U \subseteq V) \wedge |U| = k \wedge (\forall u \in U, \forall v \in (V - U), d(q, u) \leq d(q, v))\}$ as well as let $U = \{v_1, v_2, \cdots, v_k\}$, $\forall v_i \in U$, and $\forall v_j \in U$, if $i > j$, then $d(q, v_i) \geq d(q, v_j)$.*

Consider the example depicted in Fig. 2(a). Suppose $\alpha$ is equal to 0.5, a range query on the quasi-metric graph $MG$ retrieves the vertices whose distances to a query vertex $v_5$ are within 2, i.e., $RQ(v_5, 2) = \{v_1, v_2\}$. A 2NN ($k$ = 2) query on the quasi-metric graph $MG$ returns 2 vertices most similar to the query vertex $v_5$, i.e., $kNNQ(v_5, 2) = \{v_2, v_1\}$. It is worth noting that, a $k$NN query can be regarded as a range query if the combined distance from a query vertex $q$ to its $k$-th nearest neighbor, denoted as $ND_k$, is known in advance.

### C. PRUNING QUASI-METRIC GRAPH

Considering that distance calculation in metric spaces are usually complex, the pivot mapping technique is employed to avoid unnecessary distance computations.

*Pivot Mapping:* Given a pivot set $P = \{p_1, p_2, \cdots, p_l\}$, the objects in a metric space can be mapped to data points in a $l$-dimensional vector space using $P$. Specifically, an object $v$ in the metric space is mapped to a point $\phi(v) = \langle d_M(v, p_1), d_M(v, p_2), \cdots, d_M(v, p_l)\rangle$ in the $l$-dimensional vector space.

Consider the example in Fig. 2 again, where $L_2$-norm is used as $d_M$. If $P = \{v_1, v_6\}$, the original metric space (as illustrated in Fig. 2(a)) is mapped to a two-dimensional vector space (as shown in Fig. 2(b)), in which the $x$-axis denotes $d_M(v_i, v_1)$ and the $y$-axis represents $d_M(v_i, v_6)$ for any vertex $v_i$. For instance, object $v_5$ is mapped to point $\langle 2, 4\rangle$.

Based on the pivot mapping technique, the lower and upper bounds of $d_M$ can be derived as follows.

*Definition 4 (Lower and Upper Bounds): Given a pivot set $P$, the upper bound $ud_M(u, v)$ of metric distance $d_M(u, v)$ is set as $min\{d_M(u, p_i) + d_M(p_i, v) \mid p_i \in P\}$, and the lower bound $ld_M(u, v)$ is set as $max\{|d_M(u, p_i) - d_M(v, p_i)| \mid p_i \in P\}$.*

Back to the example depicted in Fig. 2(a), where $P = \{v_1, v_6\}$. According to Definition 4, $ud_M(v_5, v_4) = \sqrt{5} + 2$ and $ld_M(v_5, v_4) = \sqrt{29} - 4$ with $d_M(v_5, v_4) = \sqrt{5}$. Based on the lower and upper bounds of $d_M$, corresponding pruning and validating lemma is developed below.

*Lemma 1: Given a range query with a query vertex $q$ and a search radius $r$, a vertex $v$ can be pruned if $\alpha \times d_G(q, v) + (1 - \alpha) \times ld_M(q, v) > r$, and the vertex $v$ can be validated if $\alpha \times d_G(q, v) + (1 - \alpha) \times ud_M(q, v) \leq r$.*

*Proof:* Based on the triangle inequality, $|d_M(q, p_i) - d_M(v, p_i)| \leq d_M(q, v) \leq d_M(q, p_i) + d_M(p_i, v)$. Therefore, $ud_M(q, v) = min\{d_M(q, p_i) + d_M(p_i, v) \mid p_i \in P\} \geq d_M(q, v)$, and $ld_M(q, v) = max\{|d_M(q, p_i) - d_M(v, p_i)| \mid p_i \in P\} \leq d_M(q, v)$. If $\alpha \times d_G(q, v) + (1 - \alpha) \times ld_M(q, v) > r$, then $d(q, v) = \alpha \times d_G(q, v) + (1 - \alpha) \times d_M(q, v) \geq \alpha \times d_G(q, v) + (1 - \alpha) \times ld_M(q, v) > r$, and thus, $v$ can be pruned. If $\alpha \times d_G(q, v) + (1 - \alpha) \times ud_M(q, v) \leq r$, then $d(q, v) = \alpha \times d_G(q, v) + (1 - \alpha) \times d_M(q, v) \leq \alpha \times d_G(q, v) + (1 - \alpha) \times ud_M(q, v) \leq r$, and hence, $v$ can be validated. The proof completes. □

Consider a range query with $r = 2$ and $q = v_5$ on the quasi-metric graph shown in Fig. 2(a), where $P = \{v_1, v_6\}$ and $\alpha = 0.5$. Vertices $v_3$ and $v_6$ to $v_9$ can be discarded (i.e., $v_3, v_6 \sim v_9$ are excluded from $RQ(v_5, 2)$) by Lemma 1, since $\alpha \times d_G(v_5, v_i) + (1 - \alpha) \times ld_M(v_5, v_i) > 2 (6 \leq i \leq 9$ or $i = 3)$. Vertex $v_1$ can be validated (i.e., $v_1$ is certainly included in $RQ(v_5, 2)$) by Lemma 1, as $\alpha \times d_G(v_5, v_1) + (1 - \alpha) \times ud_M(v_5, v_1) = 2$.

Note that, the distances $d_M(v, p_i)$ from all the vertices $v$ in the quasi-metric graph to the pivots $p_i$ can be precomputed and stored. Consequently, if we compute the distances $d_M(q, p_i)(p_i \in P)$ once, the lower and upper bound distances (i.e., $ld_M(v, q)$ and $ud_M(v, q)$) from all the vertices $v$ to the query vertex $q$ can be obtained without any further metric distance computation. Nonetheless, for the vertices that cannot be pruned or validated using Lemma 1, we still need to compute their corresponding metric distances to the query vertex $q$ for further verification.

Since the combined distance function $d = \alpha \times d_G + (1 - \alpha) \times d_M$ used for the quasi-metric graph is quasi-metric, it satisfies the triangle inequality. Hence, the whole quasi-metric graph can be mapped to data points in the vector space.

As an example, the quasi-metric graph in Fig. 2(a) with $P = \{v_1, v_6\}$ and $\alpha = 0.5$ can be mapped to data points as depicted in Fig. 2(c) (e.g., vertex $v_5$ can be mapped to point $\langle 2, 5\rangle$). In addition, Definition 4 is also applied by replacing $d_M$ with $d$. Based on the lower and upper bounds of $d$, corresponding pruning and validating lemma is presented below.

*Lemma 2: Given a range query with a query vertex $q$ and a search radius $r$, a vertex $v$ can be pruned if $ld(q, v) > r$, and $v$ can be validated if $ud(q, v) \leq r$. Here, $ld(q, v)$ and $ud(q, v)$ represent the lower bound and the upper bound of the combined distance $d(q, v)$, respectively.*

*Proof:* Since the combined distance $d(u, v)$ is a quasi-metric distance, it satisfies triangle inequality. Hence, $|d(q, p_i) - d(v, p_i)| \leq d(q, v) \leq d(q, p_i) + d(p_i, v)$. Therefore, $ud(q, v) = min\{d(q, p_i) + d(p_i, v) \mid p_i \in P\} \geq d(q, v)$, and $ld(q, v) = max\{|d(q, p_i) - d(v, p_i)| \mid p_i \in P\} \leq d(q, v)$. If $ld(q, v) > r$, then $d(q, v) > r$, and thus, $v$ can be pruned. If $ud(q, v) \leq r$, then $d(q, v) \leq r$, and hence, $v$ can be validated. The proof completes. □

Again consider a range query with $r = 2$ and $q = v_5$ on the quasi-metric graph in Fig. 2(a), where $P = \{v_1, v_6\}$ and $\alpha = 0.5$. Vertices $v_6$ through $v_9$ located outside the red dashed rectangle in Fig. 2(c) can be pruned by Lemma 2 as $ld(v_5, v_i) > 2 (6 \leq i \leq 9)$, and vertex $v_1$ can be validated by Lemma 2 due to $ud(v_5, v_1) = 2$.

The difference between Lemma 1 and Lemma 2 is that, both the metric distance and the graph distance are computed

**TABLE 2.** PLL index for the quasi-metric graph in Fig. 2(a).

| $v$ | $L(v) : \{(u, \delta_{uv})\}$ |
|---|---|
| $v_1$ | $\{\}$ |
| $v_2$ | $\{(v_1, 1)\}$ |
| $v_3$ | $\{(v_1, 3), (v_2, 2)\}$ |
| $v_4$ | $\{(v_1, 2), (v_2, 1), (v_3, 1)\}$ |
| $v_5$ | $\{(v_1, 2), (v_2, 1)\}$ |
| $v_6$ | $\{(v_1, 6), (v_2, 5), (v_3, 3)\}$ |
| $v_7$ | $\{(v_1, 7), (v_2, 6), (v_3, 4), (v_6, 1)\}$ |
| $v_8$ | $\{(v_1, 4), (v_2, 3), (v_3, 1), (v_6, 2)\}$ |
| $v_9$ | $\{(v_1, 5), (v_2, 4), (v_3, 2), (v_6, 1), (v_8, 1)\}$ |

and stored together for Lemma 2, while the metric and graph distances are calculated separately for Lemma 1.

## IV. BASELINE METHODS

To address similarity search on quasi-metric graphs, similarity query approaches on graphs or in metric spaces can be adapted accordingly. In the following, we elaborate three baseline methods, namely, Pruned Landmark Labeling based method, M-tree based method, and SPB-tree based method.

### A. PRUNED LANDMARK LABELING BASED METHOD

Pruned Landmark Labeling (PLL) [36] is the state-of-the-art method to compute the shortest path distance. It provides two functions Landmark_ub and Landmark for computing the upper bound and the exact shortest path distance, respectively. Specifically, PLL precomputes a distance label for every vertex $v$, denoted as $L(v)$, by performing a pruned breadth-first search from each vertex. $L(v)$ is a set of pairs $(u, \delta_{uv})$, where $u$ is a vertex and $\delta_{uv}$ is the shortest path distance from $u$ to $v$. A shortest path distance query between vertices $s$ and $t$ can be computed as $min\{\delta_{sw} + \delta_{wt} \mid (w, \delta_{sw}) \in L(s), (w, \delta_{wt}) \in L(t)\}$. For example, Table 2 shows the PLL index for the quasi-metric graph in Fig. 2(a), then, $d_G(v_5, v_6) = min\{\delta_{v_5v_1} + \delta_{v_1v_6}, \delta_{v_5v_2} + \delta_{v_2v_6}\} = 6$.

To address similarity search on quasi-metric graphs, a pruned landmark labeling based method is developed. It traverses every vertex in the quasi-metric graph in sequel. For every vertex $v$, the method first computes the upper bound of the shortest path distance between a query vertex $q$ and the vertex $v$ by invoking function Landmark_ub. Next, the method validates $v$ using the upper bound of the combined distance by Lemma 2. If $v$ cannot be validated, the method needs to compute the exact shortest path distance from $q$ to $v$ using function Landmark, and then, it prunes or validates $v$ via Lemma 1. If $v$ still cannot be pruned or validated, the metric distance between $q$ and $v$ should be computed for the final verification. The pruned landmark labeling based method includes *PLL based Range query Algorithm* (LRA) and *PLL based kNN query Algorithm* (LNA).

Algorithm 1 presents the pseudo-code of LRA. It takes as inputs a query vertex $q$, a search radius $r$, a parameter $\alpha$, and a quasi-metric graph $MG(V, M_v, E, w)$, and outputs the result set $RQ(q, r)$ of a range query. LRA traverses the whole quasi-metric graph until all answer vertices are found (lines 1-12). For every vertex $v$ in $V$, the algorithm first calls Landmark_ub function to compute the upper bound $ud_G(q, v)$ of the shortest path distance between query vertex $q$ and vertex $v$. If $\alpha \times ud_G(q, v) + (1 - \alpha) \times ud_M(q, v) \leq r$, the vertex $v$ is inserted into the result set $RQ(q, r)$ by Lemma 2 (lines 3-4); otherwise, LRA invokes Landmark function to compute the exact shortest path distance $d_G(q, v)$ (lines 5-6). If $\alpha \times d_G(q, v) + (1 - \alpha) \times ud_M(q, v) \leq r$, the vertex $v$ is added to the result set $RQ(q, r)$ by Lemma 1 (lines 7-8); otherwise, if $\alpha \times d_G(q, v) + (1-\alpha) \times ld_M(q, v) \leq r$, LRA computes $d_M(q, v)$, and inserts $v$ into the result set if $d(q, v) \leq r$ by Lemma 1 (lines 9-12). Finally, the result set $RQ(q, r)$ is returned (line 13).

**Algorithm 1** PLL Based Range Query Algorithm (LRA)

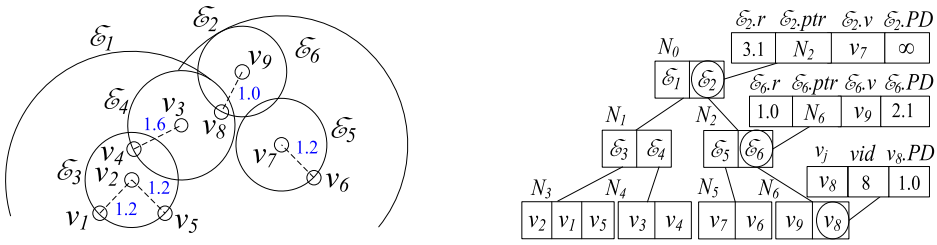
**Input:** a query vertex $q$, a search radius $r$, a quasi-metric graph $MG(V, M_v, E, w)$, a parameter $\alpha$
**Output:** the result set $RQ(q, r)$ of a range query

1: **for** each vertex $v \in V$ **do**
2: $ud_G(q, v) =$ Landmark_ub$(q, v)$
3: **if** $\alpha \times ud_G(q, v) + (1 - \alpha) \times ud_M(q, v) \leq r$ **then** // Validated by Lemma 2
4: insert $v$ into $RQ(q, r)$
5: **else** // $v$ cannot be validated by Lemma 2
6: $d_G(q, v) =$ Landmark$(q, v)$
7: **if** $\alpha \times d_G(q, v) + (1 - \alpha) \times ud_M(q, v) \leq r$ **then** // Validated by Lemma 1
8: insert $v$ into $RQ(q, r)$
9: **else if** $\alpha \times d_G(q, v) + (1 - \alpha) \times ld_M(q, v) \leq r$ **then** // Pruned by Lemma 1
10: compute $d_M(q, v)$
11: **if** $\alpha \times d_G(q, v) + (1 - \alpha) \times d_M(q, v) \leq r$ **then**
12: insert $v$ into $RQ(q, r)$
13: **return** $RQ(q, r)$

For LNA, as $ND_k$ is not known in advance for $k$NN query, $k$NN query is more complex than range query. The differences between LNA and LRA are as follows. (i) LNA first set the current $k$-th NN distance $curND_k$ to infinity, and update the value during the search until it reaches $ND_k$ (i.e., $k$NN vertices are found). (ii) LNA cannot validate vertices, because the search radius $curND_k$ is decreasing during the search, i.e., lines 2-4 and lines 7-8 of Algorithm 1 do not work for LNA. (iii) LNA uses $(1 - \alpha) \times ld_M(q, v) \leq curND_k$ to prune before invoking Landmark function for computing the shortest path distance.

### B. M-TREE BASED METHOD

Since the distance used for a quasi-metric graph is quasi-metric, it also satisfies the triangle inequality, and the metric indexes can be directly employed to tackle similarity search on quasi-metric graphs. M-tree [16] is a typical metric index belonging to the compact partitioning methods, which exploit the ball partitioning technique. Let $\mathscr{E}.v$ be the center vertex




(a) Data distribution (b) M-tree structure

**FIGURE 3.** Example of M-tree.

of a partition region $\mathscr{E}$, and $\mathscr{E}.r$ be the covering radius of $\mathscr{E}$, then the set of vertices $u$ ($\in V$) in the partition region $\mathscr{E}$, obtained via ball partitioning, is defined as $\{u \mid (u \in V) \wedge (d(u, \mathscr{E}.v) \leq \mathscr{E}.r)\}$.

*Example 2: Fig. 3 shows an example of M-tree to index the quasi-metric graph in Fig. 2(a). An* intermediate *(i.e., a* non-leaf*) entry (i.e., partition region) $\mathscr{E}$ in a root node (e.g., $N_0$) or a non-leaf node (e.g., $N_1, N_2$) records: (i) A* center vertex *$\mathscr{E}.v$ that is a selected vertex in the subtree $ST_{\mathscr{E}}$ of $\mathscr{E}$. (ii) A* covering radius *$\mathscr{E}.r$ which is the maximal distance between the center vertex $\mathscr{E}.v$ and any vertex in its subtree $ST_{\mathscr{E}}$. (iii) A* parent distance *$\mathscr{E}.PD$ that equals the distance from $\mathscr{E}.v$ to the center vertex of its parent entry. Since a root entry $\mathscr{E}$ (e.g., $\mathscr{E}_2$) has no parent entry, $\mathscr{E}.PD = \infty$. (iv) An* identifier *$\mathscr{E}.ptr$ pointing to the root node of its subtree $ST_{\mathscr{E}}$. In contrast, a leaf entry (i.e., vertex) $v$ in a leaf node (e.g., $N_3, N_6$) records: (i) A* vertex *$v_j$ which stores the detailed information of $v$. (ii) An* identifier *vid representing $v$'s identifier. (iii) A* parent distance *$v.PD$ that equals the distance from $v$ to the center vertex of $v$'s parent entry.*

Based on the M-tree, a new lemma is developed to prune unnecessary entries in the M-tree, as stated below.

*Lemma 3: Given an M-tree, a range query with a query vertex $q$ and a search radius $r$, for a non-leaf entry $\mathscr{E}$ in the M-tree, if $d(q, \mathscr{E}.v) > \mathscr{E}.r + r$, any vertex $u$ in $\mathscr{E}$ cannot be in the final result set $RQ(q, r)$, and thus, $\mathscr{E}$ can be pruned safely.*

*Proof:* For any vertex $u$ in a non-leaf entry $\mathscr{E}$, if $d(q, \mathscr{E}.v) > \mathscr{E}.r + r$, then $d(q, u) \geq d(q, \mathscr{E}.v) - d(u, \mathscr{E}.v) > \mathscr{E}.r + r - d(u, \mathscr{E}.v)$ due to the triangle inequality. According to the definition of M-tree, $d(u, \mathscr{E}.v) \leq \mathscr{E}.r$, and hence, $d(q, u) > r$. Therefore, any vertex $u$ in $\mathscr{E}$ cannot be in the final result set $RQ(q, r)$, i.e., $\mathscr{E}$ can be pruned away safely, which competes the proof. □

Consider the example in Fig. 3, for a range query with $q = v_2$ and $r = 2$, $\mathscr{E}_5$ and $\mathscr{E}_6$ can be pruned by Lemma 3, because $d(v_2, \mathscr{E}_5.v) = 5 > \mathscr{E}_5.r + 2$ and $d(v_2, \mathscr{E}_6.v) = 3.8 > \mathscr{E}_6.r + 2$. To avoid unnecessary distance computations, we can utilize the triangle inequality with the parent distances stored in the M-tree to prune unqualified entries, as stated in Lemma 4.

*Lemma 4: Given an M-tree and let $\mathscr{E}_P$ be the parent entry of entry $\mathscr{E}$, a range query with a query vertex $q$ and a search radius $r$, for the entry $\mathscr{E}$ in the M-tree, if $|d(q, \mathscr{E}_P.v) - \mathscr{E}.PD| > \mathscr{E}.r + r$, then any vertex $v$ in the entry $\mathscr{E}$ cannot be in the final result set $RQ(q, r)$, and hence, $\mathscr{E}$ can be pruned safely.*

*Proof:* According to the triangle inequality, $d(q, \mathscr{E}.v) \geq |d(q, \mathscr{E}_P.v) - d(\mathscr{E}.v, \mathscr{E}_P.v)| = |d(q, \mathscr{E}_P.v) - \mathscr{E}.PD|$. Thus, if $|d(q, \mathscr{E}_P.v) - \mathscr{E}.PD| > \mathscr{E}.r + r$, then $d(q, \mathscr{E}.v) > \mathscr{E}.r + r$. Consequently, any vertex $v$ in the entry $\mathscr{E}$ cannot be in the final result set $RQ(q, r)$, i.e., $\mathscr{E}$ can be discarded by Lemma 3. The proof completes. □

Note that, if $\mathscr{E}$ is a leaf entry, then $\mathscr{E}.r = 0$. Take Fig. 3 as an example again. For a range query with $q = v_2$ and $r = 1.5$, $\mathscr{E}_6$ can be pruned without computing $d(v_2, \mathscr{E}_6.v)$ by Lemma 4, since $|d(v_2, \mathscr{E}_2.v) - \mathscr{E}_6.PD| > \mathscr{E}_6.r + 1.5$.

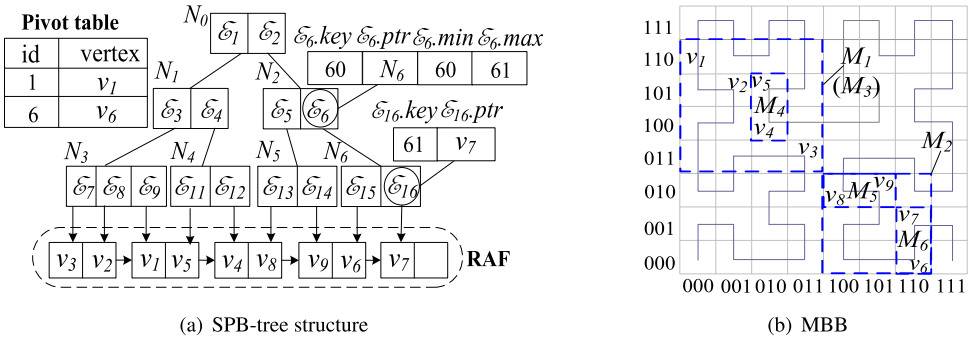
Based on Lemma 3 and Lemma 4, we present *M-tree based Range query Algorithm* (MRA) and *M-tree based kNN query Algorithm* (MNA). Algorithm 2 depicts the pseudo-code of MRA. It takes as inputs a query vertex $q$,

**Algorithm 2** M-Tree Based Range Query Algorithm (MRA)

**Input:** a query vertex $q$, a search radius $r$, a parameter $\alpha$, an M-tree $M$ build on the quasi-metric graph $MG(V, M_v, E, w)$

**Output:** the result set $RQ(q, r)$ of a range query

```
1: push all root entries of M into a queue H
2: while H ≠ ∅ do
3:     pop the top entry 𝓔 from H
4:     if 𝓔 points to a non-leaf node then
5:         for each subentry 𝓔_S in 𝓔 do
6:             if |d(q, 𝓔.v) − 𝓔_S.PD| ≤ 𝓔_S.r + r then // Pruned by Lemma 4
7:                 if d(q, 𝓔_S.v) ≤ 𝓔_S.r + r then // Pruned by Lemma 3
8:                     push 𝓔_S into H
9:     else // 𝓔 points to a leaf node
10:        for each subentry 𝓔_S in 𝓔 do
11:            if |d(q, 𝓔.v) − 𝓔_S.PD| ≤ r then // Pruned by Lemma 4
12:                compute d(q, 𝓔_S) = α × d_G(q, 𝓔_S) + (1 − α) × d_M(q, 𝓔_S)
13:                if d(q, 𝓔_S) ≤ r then
14:                    insert 𝓔_S into RQ(q, r)
15: return RQ(q, r)
```




(a) SPB-tree structure (b) MBB

**FIGURE 4.** Example of SPB-tree.

a search radius $r$, an M-tree $M$ build on the quasi-metric graph $MG(V, M_v, E, w)$, and a parameter $\alpha$, and outputs the result set $RQ(q, r)$. Initially, MRA pushes all the root entries of $M$ into a queue $H$ (line 1). Thereafter, a while-loop is performed until $H$ is empty (lines 2-14). In each iteration, the algorithm first pops the top entry $\mathscr{E}$ from $H$ (line 3). Next, if $\mathscr{E}$ points to a non-leaf node, it inserts the subentries $\mathscr{E}_S$ that cannot be pruned by Lemma 3 and Lemma 4 into $H$ (lines 4-8). Otherwise, if $\mathscr{E}$ points to a leaf node, it prunes subentries (i.e., vertices) $\mathscr{E}_S$ by Lemma 4 (lines 9-11). For the unpruned subentries $\mathscr{E}_S$, MRA computes the combined distance $d(q, \mathscr{E}_S)$ (line 12). If $d(q, \mathscr{E}_S) \leq r$, $\mathscr{E}_S$ is inserted into $RQ(q, r)$ (lines 13-14). Finally, the algorithm returns the result set $RQ(q, r)$ (line 15).

The differences between MNA and MRA are as follows. (i) MNA uses a current $k$-th NN distance $curND_k$ for pruning, and updates the corresponding value using unpruned vertices. (ii) MNA visits the entries $\mathscr{E}$ in the M-tree in ascending order of their minimal distances to the query vertex $q$ (denoted as $MIND(q, \mathscr{E})$). Thus, $\mathscr{E}$ can be safely pruned if $MIND(q, \mathscr{E}) \geq curND_k$, i.e., for MNA, $H$ is a priority queue, in which all entries $\mathscr{E}$ are sorted in ascending order of $MIND(q, \mathscr{E})$.

### C. SPB-TREE BASED METHOD

SPB-tree [34] is the state-of-the-art metric index belonging to the hybrid methods, which combines compact partitioning with the use of pivots. The SPB-tree can be directly built on the quasi-metric graph to answer similarity search.

The SPB-tree utilizes the two-stage mapping, i.e., pivot mapping (as discussed in Section III-C) and space-filling curve (SFC) mapping, to map vertices in the vector space to SFC values (i.e., integers) in a one-dimensional space while maintaining spatial proximity. Then, a $B^+$-tree with the minimum bounding box (MBB) information is used to index the SFC values.

*Example 3: Fig. 4 depicts an example of SPB-tree, where Fig. 4(a) illustrates an SPB-tree to index the quasi-metric graph in Fig. 2(a) and Fig. 4(b) shows the space-filling curve (SFC) mapping after the pivot mapping depicted in Fig. 2(c). For instance, $\phi(v_5) = \langle 2, 5\rangle$ after the pivot mapping, and $SFC(\phi(v_5)) = 29$ after the SFC (i.e., Hilbert curve) mapping. An SPB-tree shown in Fig. 4(a) contains three parts, i.e., the pivot table that stores selected vertices (e.g., $v_1$ and $v_6$) to map a metric space to a vector space, the $B^+$-tree, and the RAF which is sorted to store the vertices in ascending order of SFC values as they appear in the $B^+$-tree. Note that, each leaf entry $\mathscr{E}$ in a leaf node (e.g., $N_3$, $N_4$, $N_5$, and $N_6$) of the $B^+$-tree records (i) the* SFC value *$\mathscr{E}.key$, and (ii) an* identifier *$\mathscr{E}.ptr$ to the actual object in the RAF. Each non-leaf entry $\mathscr{E}$ in the root or an intermediate node (e.g., $N_0$, $N_1$, and $N_2$) of the $B^+$-tree records (i) the* minimal SFC value key *$\mathscr{E}.key$ in its subtree, (ii) an* identifier *$\mathscr{E}.ptr$ to the root node of its subtree, and (iii) the* SFC values *min and max for $\langle a_1, a_2, \ldots, a_l\rangle$ and $\langle b_1, b_2, \ldots, b_l\rangle$ to represent the minimum bounding box $\mathscr{E}.MBB = \{[a_i, b_i] \mid 1 \leq i \leq l\}$. Here, $\mathscr{E}.MBB$ is the axis aligned minimum bounding box to include all $\phi(v)$ with $SFC(\phi(v)) \in \mathscr{E}$, e.g., the non-leaf entry $\mathscr{E}_6$ uses min $(= 60)$ and max $(= 61)$ to denote the $M_6$ of $N_6$.*

*Definition 5 (Lower Bound Distance of Entry): Given a pivot set $P$, the lower bound distance $ld(\mathscr{E}, v)$ between a vertex $v$ and a non-leaf entry $\mathscr{E}$ is set as $max\{a_i - d(v, p_i), d(v, p_i) - b_i \mid p_i \in P\}$. Here, $a_i$ and $b_i$ can be obtained by $\mathscr{E}.MBB$.*

Consider the example depicted in Fig. 4, where $P = \{v_1, v_6\}$. According to Definition 5, $ld(\mathscr{E}_6, v_5) = 4$ as $\mathscr{E}_6.MBB = \{[6, 6], [0, 1]\}$. Based on the newly defined lower bound distance, we develop Lemma 5 to prune unnecessary entries.

*Lemma 5: Given an SPB-tree, a range query with a query vertex $q$ and a search radius $r$, for a non-leaf entry $\mathscr{E}$ in the SPB-tree, if $ld(\mathscr{E}, q) > r$, then $\mathscr{E}$ can be pruned safely.*

*Proof:* $\forall\ u \in \mathscr{E}$, we can get that $a_i \leq d(u, p_i) \leq b_i$. According to Definition 4, $ld(u, q) = \max\{|d(u, p_i) - d(q, p_i)| \mid p_i \in P\} \leq d(u, q) (u \in \mathscr{E})$. Hence, $ld(\mathscr{E}, q) = \max\{a_i - d(q, p_i), d(q, p_i) - b_i \mid p_i \in P\} \leq d(u, q)\ (\forall u \in \mathscr{E})$. If $ld(\mathscr{E}, q) > r$, then $d(u, q) > r$ for any $u \in \mathscr{E}$, and thus, $\mathscr{E}$ can be pruned safely. □

Back to the example illustrated in Fig. 4. For a range query with $q = v_5$ and $r = 2$, the non-leaf entry $\mathscr{E}_6$ can be pruned by Lemma 5, because $ld(\mathscr{E}_6, v_5) = 4 > 2$.

Based on the SPB-tree, we propose *SPB-tree based Range query Algorithm* (SRA) and *SPB-tree based kNN query*

**Algorithm 3** SPB-Tree Based Range Query Algorithm (SRA)

**Input:** a query vertex $q$, a search radius $r$, a parameter $\alpha$, an SPB-tree $S$ build on the quasi-metric graph $MG(V, M_v, E, w)$
**Output:** the result set $RQ(q, r)$ of a range query

1: push all root entries of $S$ into a queue $H$
2: **while** $H \neq \varnothing$ **do**
3: pop the top entry $\mathscr{E}$ from $H$
4: **if** $\mathscr{E}$ points to a non-leaf node **then**
5: **for** each subentry $\mathscr{E}_S$ in $\mathscr{E}$ **do**
6: **if** $ld(\mathscr{E}_S, q) \leq r$ **then** // Pruned by Lemma 5
7: push $\mathscr{E}_S$ into $H$
8: **else** //$\mathscr{E}$ points to a leaf node
9: **for** each subentry $\mathscr{E}_S$ in $\mathscr{E}$ **do**
10: **if** $ud(\mathscr{E}_S, q) \leq r$ **then** // Validated by Lemma 2
11: insert $\mathscr{E}_S$ into $RQ(q, r)$
12: **if** $ld(\mathscr{E}_S, q) \leq r$ **then** // Pruned by Lemma 2
13: compute $d(\mathscr{E}_S, q) = \alpha \times d_G(\mathscr{E}_S, q) + (1 - \alpha) \times d_M(\mathscr{E}_S, q)$
14: **if** $d(\mathscr{E}_S, q) \leq r$ **then**
15: insert $\mathscr{E}_S$ into $RQ(q, r)$
16: **return** $RQ(q, r)$

*Algorithm* (SNA). Algorithm 3 depicts the pseudo-code of SRA. It takes as inputs a query vertex $q$, a search radius $r$, a parameter $\alpha$, and an SPB-tree $S$ build on the quasi-metric graph $MG(V, M_v, E, w)$, and outputs the result set $RQ(q, r)$ of a range query. First of all, SRA pushes all the root entries of $S$ into a queue $H$ (line 1). Thereafter, a while-loop is performed until $H$ is empty (lines 2-15). In every iteration, the algorithm first pops the top entry $\mathscr{E}$ from $H$. Next, if $\mathscr{E}$ points to a non-leaf node, it inserts the subentries $\mathscr{E}_S$ that cannot be pruned by Lemma 5 into $H$ (lines 4-7). Otherwise, if $\mathscr{E}$ points to a leaf node, SRA validates or prunes subentries (i.e., vertices) $\mathscr{E}_S$ by Lemma 2, and inserts the unpruned subentries $\mathscr{E}_S$ into the result set $RQ(q, r)$ if $d(q, \mathscr{E}_S) \leq r$ (lines 8-15). Finally, the algorithm returns $RQ(q, r)$ (line 16).

The differences between SNA and SRA are as follows. (i) SNA uses a current $k$-th NN distance $curND_k$ for pruning, and updates the corresponding value using unpruned vertices. (ii) SNA visits entries $\mathscr{E}$ in the SPB-tree in ascending order of their lower bound distances $ld(\mathscr{E}, q)$ to the query vertex $q$, i.e., for SNA, all the entries $\mathscr{E}$ in the queue $H$ are sorted in ascending order of $ld(\mathscr{E}, q)$. (iii) SNA cannot validate the vertices since the search radius $curND_k$ is decreasing during the search, i.e., lines 10-11 of Algorithm 3 do not work for SNA.

### D. DISCUSSION

In this subsection, we analyze query processing costs for all baseline methods/algorithms.

In general, shortest path distance computation and metric distance computation are main operations in query processing, and thus, their costs dominate the query cost. For pruned landmark labeling based method, it needs to traverse the whole quasi-metric graph to find the final result. As analyzed in [36], each shortest path distance computation between a pair of vertices $s$ and $t$ can be answered in $O(|L_{BP}(s)| + |L_{BP}(t)|)$ time using the bit-parallel labels technique. Here, $L_{BP}(s)$ (resp. $L_{BP}(t)$) denotes bit-parallel labels of $s$ (resp. $t$), and $|L_{BP}(s)|$ (resp. $|L_{BP}(t)|$) represents the corresponding cardinality. Hence, in total, Landmark function or Landmark_ub function used to compute the shortest path distance takes $O(|V| \times (|L_{BP}(s)| + |L_{BP}(t)|))$ time. For an unpruned vertex $u$, pruned landmark labeling based method calculates the metric distance between vertices $q$ and $u$. Let $|V_u|$ be the number of unpruned vertices and $f(m)$ be the cost of metric distance computation, pruned landmark labeling based method needs $O(|V_u| \times f(m))$ time for metric distance computation. Thus, the total cost for pruned landmark labeling based method is $O(|V| \times (|L_{BP}(s)| + |L_{BP}(t)|) + |V_u| \times f(m))$.

For M-tree based method and SPB-tree based method, they can prune unqualified vertices using Lemmas 2 through 5. For every unpruned vertex $u$, M-tree based or SPB-tree based method first needs to traverse the M-tree or the SPB-tree to locate the vertex $u$, which takes $log(|V|)$ time. Next, shortest path distance and metric distance between $u$ and a query vertex $q$ are evaluated, and the corresponding time complexities are $O(|E| + |V|log|V|)$ and $f(m)$, respectively. Hence, the total cost for M-tree based or SPB-tree based method is $O(|V_u|log|V| \times (|E| + |V|log|V| + f(m)))$. Clearly, the cost of M-tree based or SPB-tree based method is more expensive than pruned landmark labeling based method, due to the high cost for computing the shortest path distances (i.e., traversing the entire quasi-metric graph to compute every shortest path distance). In addition, the SPB-tree achieves better pruning ability than the M-tree, and hence, the SPB-tree based method is more efficient, which is also verified in Section VI.

## V. GRAPH TRAVERSING METHODS

In this section, we propose two simple but efficient methods for answering similarity search on quasi-metric graphs, i.e., best-first method and breadth-first method. These methods visit the vertices in ascending order of the shortest path distances w.r.t. a query vertex $q$ in order to terminate computation earlier and utilize the triangle inequality to filter unnecessary verification.

### A. BEST-FIRST METHOD

To avoid traversing the whole quasi-metric graph multiple times (i.e., every shortest path distance computation needs to traverse the quasi-metric graph once) and avoid the index construction cost, we propose a simple yet robust best-first traversal method. It visits the vertices in ascending order of their shortest path distances to a query vertex $q$, i.e., the smaller the shortest path distance from the query vertex $q$ to a vertex $v$ is, the earlier verification that whether $v$ is an

**Algorithm 4** Best-First Range Query Algorithm (BeRA)

**Input:** a query vertex $q$, a search radius $r$, a quasi-metric graph $MG(V, M_v, E, w)$, a parameter $\alpha$
**Output:** the result set $RQ(q, r)$ of a range query

1: **for** each vertex $v \in V$ **do**
2: $v.flag$ = false; $d_G(q, v) = \infty$
3: $q.flag$ = true; $d_G(q, q) = 0$
4: push $q$ into a queue $H_1$
5: **while** $H_1 \neq \varnothing$ **do**
6: pop the top vertex $v$ from $H_1$
7: **for** each adjacent vertex $u$ of $v$ **do**
8: **if** $u.flag$ = false and $d_G(q, v) + w(v, u) < d_G(q, u)$ **then**
9: $d_G(q, u) = d_G(q, v) + w(v, u)$
10: push $u$ into a priority queue $H_2$ sorted in ascending order of the current shortest path distance $d_G(q, u)$
11: **if** $H_2 \neq \varnothing$ **then**
12: pop vertex $s$ from $H_2$; push $s$ into $H_1$; and set $s.flag$ = true
13: **if** $\alpha \times d_G(q, s) \leq r$ **then**
14: **if** $\alpha \times d_G(q, s) + (1 - \alpha) \times ud_M(q, s) \leq r$ **then** // Validated by Lemma 1
15: insert $v$ into $RQ(q, r)$
16: **else if** $\alpha \times d_G(q, s) + (1 - \alpha) \times ld_M(q, s) \leq r$ **then** // Pruned by Lemma 1
17: compute $d_M(q, s)$
18: **if** $\alpha \times d_G(q, s) + (1 - \alpha) \times d_M(q, s) \leq r$ **then**
19: insert $v$ into $RQ(q, r)$
20: **else** // Earlier termination by Lemma 7
21: **return** $RQ(q, r)$
22: **return** $RQ(q, r)$

answer vertex is made. Moreover, Lemma 1 can be employed to prune or validate vertices.

The best-first method includes *Best-first Range query Algorithm* (BeRA) and *Best-first kNN query Algorithm* (BeNA).

Algorithm 4 presents the pseudo-code of BeRA. It takes as inputs a query vertex $q$, a search radius $r$, a parameter $\alpha$, and a quasi-metric graph $MG(V, M_v, E, w)$, and outputs the result set $RQ(q, r)$. To begin with, for each vertex $v$ in $V$, BeRA initializes variables $d_G(q, v)$ and $v.flag$ that denotes whether exact $d_G(q, v)$ has been computed, and pushes $q$ into a queue $H_1$ (lines 1-4). Thereafter, a while-loop is performed (lines 5-21). In each iteration, BeRA first pops the top vertex $v$ from $H_1$, and for every $v$'s adjacent vertex $u$ whose exact $d_G(q, u)$ has not been calculated, BeRA updates its current $d_G(q, u)$ and pushes $u$ with the updated $d_G(q, u)$ into a priority queue $H_2$, in which vertices are sorted in ascending order of their current shortest path distances (line 6-10). Next, BeRA pops the top vertex $s$ with the minimum $d_G(q, s)$ from $H_2$, pushes $s$ into $H_1$ for further traversal, and sets $s.flag$ to true because the exact $d_G(q, s)$ is computed (line 11-12). In the

**Algorithm 5** Breadth-First Range Query Algorithm (BrRA)

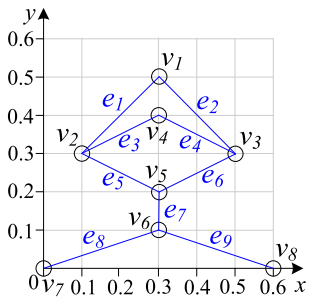
**Input:** a query vertex $q$, a search radius $r$, a quasi-metric graph $MG(V, M_v, E, w)$, a parameter $\alpha$
**Output:** the result set $RQ(q, r)$ of a range query

1: **for** each vertex $v \in V$ **do**
2: $v.visit$ = false; $d_G(q, v) = \infty$
3: $q.visit$ = true; $d_G(q, q) = 0$
4: push $q$ into a queue $H$
5: **while** $H \neq \varnothing$ **do**
6: pop the top vertex $v$ from $H$
7: **for** each adjacent vertex $u$ of $v$ **do**
8: **if** $u.visit$ = false **then**
9: $d_G(q, u) = d_G(q, v) + 1$; $u.visit = true$
10: push $u$ into the queue $H$
11: **if** $\alpha \times d_G(q, u) \leq r$ **then**
12: **if** $\alpha \times d_G(q, u) + (1 - \alpha) \times ud_M(q, u) \leq r$ **then** // Validated by Lemma 1
13: insert $v$ into $RQ(q, r)$
14: **else if** $\alpha \times d_G(q, u) + (1 - \alpha) \times ld_M(q, u) \leq r$ **then** // Pruned by Lemma 1
15: compute $d_M(q, u)$
16: **if** $\alpha \times d_G(q, u) + (1 - \alpha) \times d_M(q, u) \leq r$ **then**
17: insert $v$ into $RQ(q, r)$
18: **else**// Earlier termination by Lemma 7
19: **return** $RQ(q, r)$
20: **return** $RQ(q, r)$

sequel, if $\alpha \times d_G(q, s) \leq r$, BeRA proceeds to verify vertex $s$. If $\alpha \times d_G(q, s) + (1 - \alpha) \times ud_M(q, s) \leq r$, $s$ is added to the result set $RQ(q, r)$ by Lemma 1 (lines 14-15). Otherwise, if $\alpha \times d_G(q, s) + (1 - \alpha) \times ld_M(q, s) \leq r$, BeRA computes $d_M(q, s)$ and inserts $s$ into the result set $RQ(q, r)$ if $d(q, s) \leq r$ by Lemma 1 (lines 16-19). Once $\alpha \times d_G(q, s) > r$, BeRA stops, and returns the result set $RQ(q, r)$ due to the earlier termination condition presented by Lemma 7 in Section V-C (lines 20-21). Finally, after the whole iteration terminates, the algorithm returns the final result set $RQ(q, r)$ (line 22).

The differences between BeNA and BeRA are that, (i) BeNA uses a current $k$-th NN distance $curND_k$ instead of $r$ as the search radius for pruning, and updates its corresponding value using unpruned vertices; and (ii) BeNA cannot validate the vertices as the current $k$-th NN distance $curND_k$ is decreasing during the search, i.e., lines 14-15 of Algorithm 4 do not work for BeNA.

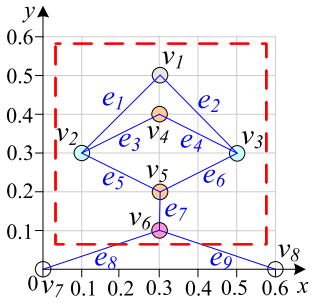
*Example 4: Fig. 5 illustrates an example of graph traversing method on the quasi-metric graph $MG(V, M_v, E, w)$ depicted in Fig. 5(a), where $V = \{v_1, v_2, \cdots, v_8\}$, $E = \{e_1, e_2, \cdots, e_9\}$, and $w(e_i)$ is equal to 1 for any edge $e_i$ ($1 \leq i \leq 9$). Suppose a pivot set $P$ is $\{v_7, v_8\}$, and metric distance function is $L_1$-norm. The distances $d_M(v_i, p_i)$ from all vertices $v_i$ in the quasi-metric graph to the pivots $p_i$ ($\in P$) can be precomputed and stored, as shown in Fig. 5(b). Given a range query with $q = v_1$ and $r = 1$, and set $\alpha$ as* 0.5.




(a) Quasi-metric graph

| $v_i$ | $d_M(v_7,v_i)$ | $d_M(v_8,v_i)$ |
|---|---|---|
| $v_1$ | 0.8 | 0.8 |
| $v_2$ | 0.4 | 0.8 |
| $v_3$ | 0.8 | 0.4 |
| $v_4$ | 0.7 | 0.7 |
| $v_5$ | 0.5 | 0.5 |
| $v_6$ | 0.4 | 0.4 |
| $v_7$ | 0 | 0.6 |
| $v_8$ | 0.6 | 0 |

(b) Mapped metric space




(c) Graph traversing

**FIGURE 5.** Example of graph traversing methods.

*Initially,* BeRA *pushes a query vertex* $v_1$ *with its corresponding shortest path distance into a queue* $H_1$*, resulting in* $H_1 = \{\langle v_1, 0\rangle\}$*. Then,* BeRA *performs a while-loop to traverse the quasi-metric graph from the vertex* $v_1$ *in best-first manner.*

*Loop 1: As* $H_1 \neq \varnothing$*,* BeRA *pops the top vertex* $v_1$ *from* $H_1$*, updates* $d_G(v_1, v_2) = 1$ *and* $d_G(v_1, v_3) = 1$ *as* $v_2$ *and* $v_3$ *are two adjacent vertices of* $v_1$*, and pushes* $\langle v_2, 1\rangle$ *and* $\langle v_3, 1\rangle$ *into a priority queue* $H_2$*, after which* $H_2 = \{\langle v_2, 1\rangle, \langle v_3, 1\rangle\}$*. Next, it pops* $\langle v_2, 1\rangle$ *from* $H_2$*, i.e., the exact shortest path distance between* $v_1$ *and* $v_2$ *is computed. Since* $\alpha \times d_G(v_1, v_2) = 0.5 < r$*,* $v_2$ *is pushed into* $H_1$*. In the sequel,* BeRA *computes* $d(v_1, v_2) = 0.7$ *because Lemma 1 cannot validate or prune* $v_2$*, and inserts* $v_2$ *into the result set* $RQ(v_1, 1)$*. After Loop 1, we can get that* $H_1 = \{\langle v_2, 1\rangle\}$*,* $H_2 = \{\langle v_3, 1\rangle\}$*, and* $RQ(v_1, 1) = \{v_2\}$*.*

*Loops 2–4: The processing is similar as Loop* 1*. After that, we can get that* $H_1 = \{\langle v_5, 2\rangle\}$*,* $H_2 = \varnothing$*, and* $RQ(v_1, 1) = \{v_2, v_3\}$*.*

*Loop 5: As* $H_1 \neq \varnothing$*,* BeRA *pops* $\langle v_5, 2\rangle$ *from* $H_1$*, and updates* $d_G(v_1, v_6) = 3$ *since* $v_6$ *is the adjacent vertex of* $v_5$*. Thereafter,* BeRA *pushes* $\langle v_6, 3\rangle$ *into* $H_2$ *and then pops* $\langle v_6, 3\rangle$ *from* $H_2$*. As* $\alpha \times d_G(v_1, v_6) = 1.5 > r$*,* BeRA *stops traversing the quasi-metric graph due to the earlier termination condition, and returns the final result set* $RQ(v_1, 1) = \{v_2, v_3\}$*.*

### B. BREADTH-FIRST METHOD

Best-first method can be applied to both weighted and unweighted quasi-metric graphs. Nonetheless, for the unweighted quasi-metric graph, a more efficient way for similarity search is breadth-first traversal from the query vertex. This is because, for the best-first method, only one vertex is verified in every iteration, whereas for the breadth-first method, the shortest path distances between the query vertex and all the traversed vertices are obtained due to the property of the unweighted graph, and thus, all the traversed vertices can be verified in each iteration, which boosts the search.

The breadth-first method contains *Breadth-first Range query Algorithm* (BrRA) and *Breadth-first kNN query Algorithm* (BrNA). Algorithm 5 depicts the pseudo-code of BrRA. Initially, for each vertex $v$ in $V$, it initializes $d_G(q, v)$ and $v.visit$ that denotes whether $v$ has been visited (lines 1-3), and pushes a query vertex $q$ into a queue $H$ (line 4). Thereafter, a while-loop is performed (lines 5-19). In every iteration, BrRA first pops the top vertex $v$ from $H$. Next, for every $v$'s adjacent vertex $u$ that has not been traversed, BrRA computes the exact $d_G(q, u)$, sets $u.visit$ as true, and pushes $u$ into the queue $H$ for further evaluation (line 7-10). Then, if $\alpha \times d_G(q, u) \leq r$, BrRA proceeds to verify $u$. If $\alpha \times d_G(q, u) + (1-\alpha) \times ud_M(q, u) \leq r$, $u$ is added to the result set $RQ(q, r)$ by Lemma 1 (lines 12-13). Otherwise, if $\alpha \times d_G(q, u) + (1-\alpha) \times ld_M(q, u) \leq r$, BrRA computes $d_M(q, u)$, and inserts $u$ into the result set $RQ(q, r)$ if $d(q, u) \leq r$ by Lemma 1 (lines 14-17). Once $\alpha \times d_G(q, u) > r$, BrRA stops, and returns the result set $RQ(q, r)$ according to Lemma 7 proposed in Section V-C (lines 18-19). Finally, BrRA returns the result set $RQ(q, r)$ (line 20).

The difference between BrNA and BrRA is similar as that between BeNA and BeRA and thus omitted.

*Example 5: Back to Example 4, we illustrate* BrRA *using a range query with* $q = v_1$ *and* $r = 1$*. Similar as Example 4,* BrRA *first pushes* $v_1$ *into* $H$*, after which* $H = \{\langle v_1, 0\rangle\}$*. Thereafter, it starts a while-loop to traverse the quasi-metric graph following the breadth-first fashion.*

*Loop 1:* BrRA *first pops the top vertex* $v_1$ *from* $H$*. Then, for two unvisited adjacent vertices* $v_2$ *and* $v_3$ *of* $v_1$*, the algorithm computes* $d_G(v_1, v_2) = 1$ *and* $d_G(v_1, v_3) = 1$*, and pushes* $v_2$ *and* $v_3$ *into* $H$*. Since* $d(v_1, v_2) = 0.7 < r$ *and* $d(v_1, v_3) = 0.7 < r$*,* $v_2$ *and* $v_3$ *are added to the result set* $RQ(v_1, 1)$*. After the loop, we can get that* $H = \{\langle v_2, 1\rangle, \langle v_3, 1\rangle\}$*,* $RQ(v_1, 1) = \{v_2, v_3\}$*.*

*Loops 2–4: The processing is similar as Loop* 1 *and hence skipped.*

*Loop 5:* BrRA *pops the top vertex* $v_5$ *from* $H$*, and then, it computes* $d_G(v_6, v_1) = 3$ *for the adjacent vertex* $v_6$*. As* $\alpha \times d_G(v_1, v_6) = 1.5 > r$*,* BrRA *stops traversing the quasi-metric graph, and returns the final result set* $RQ(v_1, 1) = \{v_2, v_3\}$*.*

### C. DISCUSSION

In this subsection, we first clarify the advantage of both best-first method and breadth-first method, compared with baseline methods/algorithms, and then, we analyze their correctness and time complexities.

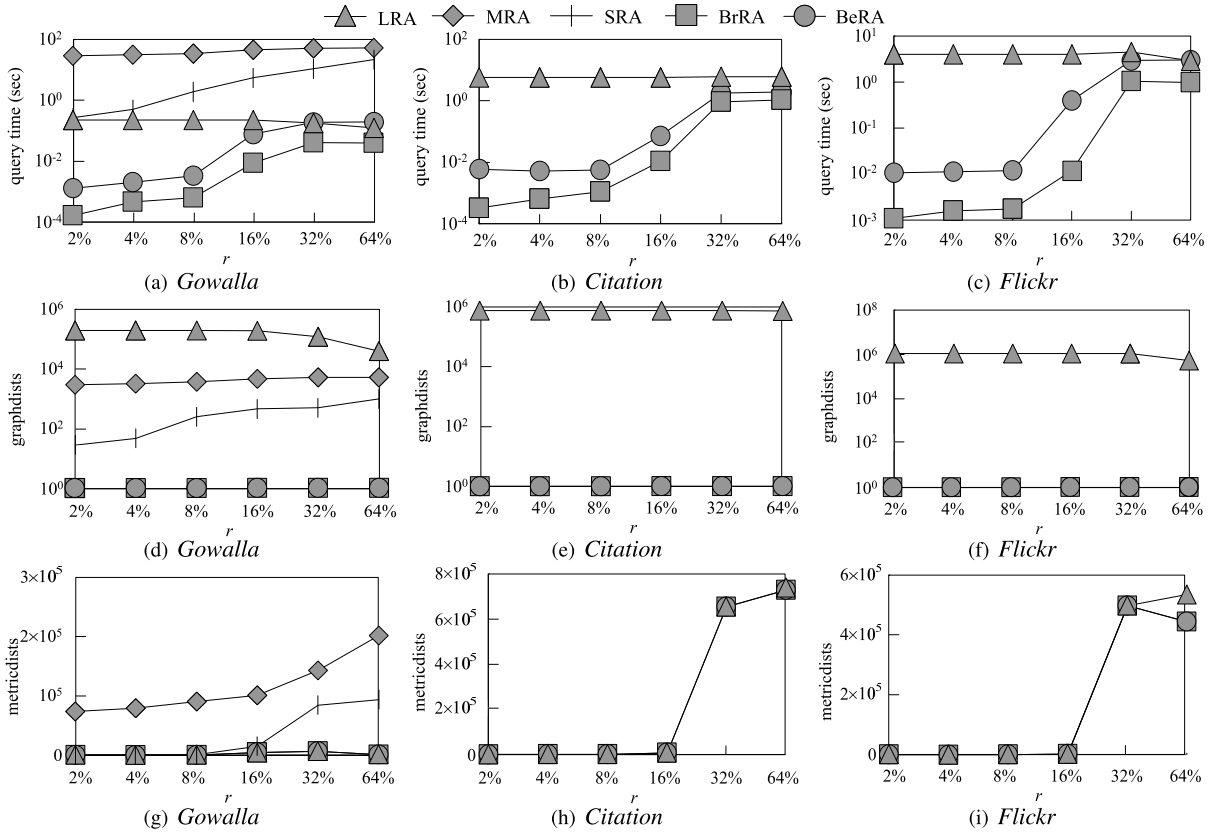



**FIGURE 6.** Range query performance vs. $r$.

Although baseline algorithms construct offline indexes that can be used to prune unqualified vertices, they still need to compute shortest path distances between all unpruned vertices and the query vertex during the search as mentioned in Section IV-D, which is costly due to traversing the quasi-metric graph multiple times. Using the best-first and breadth-first traversal paradigms, all the shortest path distances can be computed by traversing the quasi-metric graph only once. In addition, in most cases, both best-first method and breadth-first method traverse part of the graph due to the early termination condition. For instance, in Example 4 and Example 5, the search space is bounded by the red dashed rectangle in Fig. 5(c), resulting in better search performance as to be verified in Section VI-B.

To prove the correctness of best-first method and breadth-first method, we present two lemmas, as stated below.

*Lemma 6: Given a quasi-metric graph $MG(V, M_v, E, w)$ and a query vertex $q$, the best-first method can compute exact $d_G(q, u)$ when $u$ is popped from a queue $H_2$.*

*Proof:* Let $P_s = \{q, e_1, t_1, \cdots, t_{m-1}, e_m, u\}$ be the current shortest path when $u$ is popped from $H_2$, then, $t_1, \cdots, t_{m-1}$ must be the vertices that have been popped from $H_2$, and $u$ is the vertex with the minimal $d_G(q, u)$. By contradiction, assume that $P_s$ is not the exact shortest path. Thus, there exists an exact shortest path $P'_s$ containing vertices that have not been popped from $H_2$. Let $P'_s = \{q, e'_1, t'_1, \cdots, s, \cdots, t'_{m-1}, e_m, u\}$, $s$ be the first vertex that has not been popped from $H_2$, and $d'_G(q, u)$ be the corresponding exact shortest path distance. Hence, $d_G(q, s) \leq d'_G(q, u) < d_G(q, u)$, indicating that $s$ is the vertex with the minimal $d_G(q, s)$, which contradicts that $u$ is the vertex with the minimal $d_G(q, u)$. The proof completes. □

*Lemma 7: Given a quasi-metric graph $MG(V, M_v, E, w)$, a parameter $\alpha$, and a range query with a query vertex $q$ and a search radius $r$, best-first and breadth-first methods can terminate and return the exact result $RQ(q, r)$ if $\alpha \times d_G(q, u) > r$, in which $u$ is the visited vertex.*

*Proof:* Since vertices are visited in ascending order of their shortest path distances w.r.t. a query vertex $q$, we have $d_G(q, v) \geq d_G(q, u)$ for any vertex $v$ that has not been visited. Thus, if $\alpha \times d_G(q, u) > r$, then $\alpha \times d_G(q, v) > r$. Consequently, $d(q, v) \geq \alpha \times d_G(q, v) > r$, i.e., all the unvisited vertices cannot be in the final result set $RQ(q, r)$, and then, best-first and breadth-first methods can stop and return the final right result $RQ(q, r)$. The proof completes. □

Note that, Lemma 7 is also applicable for $k$NN search by replacing $r$ with $curND_k$. Obviously, Lemma 1, Lemma 6, and Lemma 7 guarantee the correctness of best-first and breadth-first methods.

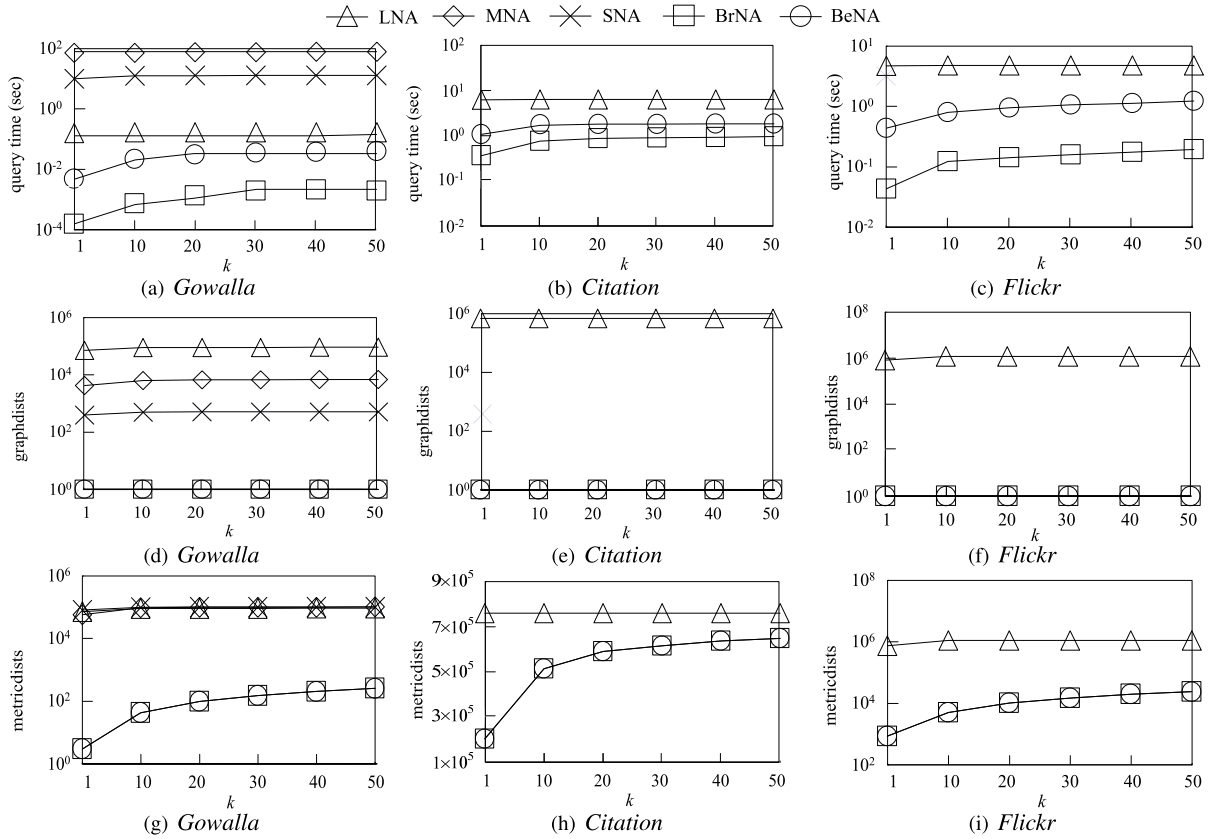



**FIGURE 7.** *k*NN query performance vs. *k*.

Next, we present the time complexities of best-first method and breadth-first method, respectively.

The best-first method maintains two queues. One is used for graph traversal, and the other is a priority queue that is utilized to compute the shortest path distance. The best-first method traverses the vertices in ascending order of their shortest path distances w.r.t. the query vertex, in order to take advantage of shortest path distance pruning. This is similar as Dijkstra algorithm. Thus, in the worst case (i.e., all the vertices need to be evaluated), the time complexity of best-first method is $O(|E| + |V| \times (log|V| + f(m)))$, where $f(m)$ is the cost for metric distance computation.

The breadth-first method only maintains one queue that is used for graph traversal, and the shortest path distance between a vertex $v$ and the query vertex is evaluated once the vertex $v$ is visited. Thus, in the worst case, the cost for computing shortest path distances between all the vertices and the query vertex is $O(|E|+|V|)$, and the total time complexity of breadth-first method is $O(|E| + |V| \times (1 + f(m)))$.

## VI. EXPERIMENTAL EVALUATION

In this section, we present a comprehensive experimental evaluation. In what follows, we first introduce experiment settings, and then, we evaluate the efficiency and effectiveness of our methods.

### A. EXPERIMENT SETTINGS

We employ three real datasets, viz., *Gowalla*,[1] *Flickr*,[2] and *Citation*.[3] *Gowalla* contains locations, in which $L_2$-norm is utilized to compute the metric distance. Two locations are connected if they are shared by the same user. *Flickr* includes images, where every image is associated with 282-dimensional features, and $L_2$-norm is used to compare image features. If two images are tagged by the same user, an edge is added between them. *Citation* provides a comprehensive list of research papers, in which every paper is associated with a set of keywords, and Jaccard distance is employed to measure the corresponding metric similarity. Two papers are connected if one cites another. Table 3 summarizes the statistics of the real datasets used in our experiments.

We study the performance of the algorithms when varying the parameters shown in Table 4, where the bold denotes the defaults, and $d^+$ is the maximal distance between any two vertices. In every experiment, we change one parameter, and set the others to their default values. The main performance metrics include query time, the number of shortest path distance computations (*graphdists* for short), and the number of metric distance computations (*metricdists* for short).

[1]Gowalla is available at https://snap.stanford.edu/data/loc-gowalla.html.
[2]Flickr is available at https://www.flickr.com/.
[3]Citation is available at https://cn.aminer.org/citation.

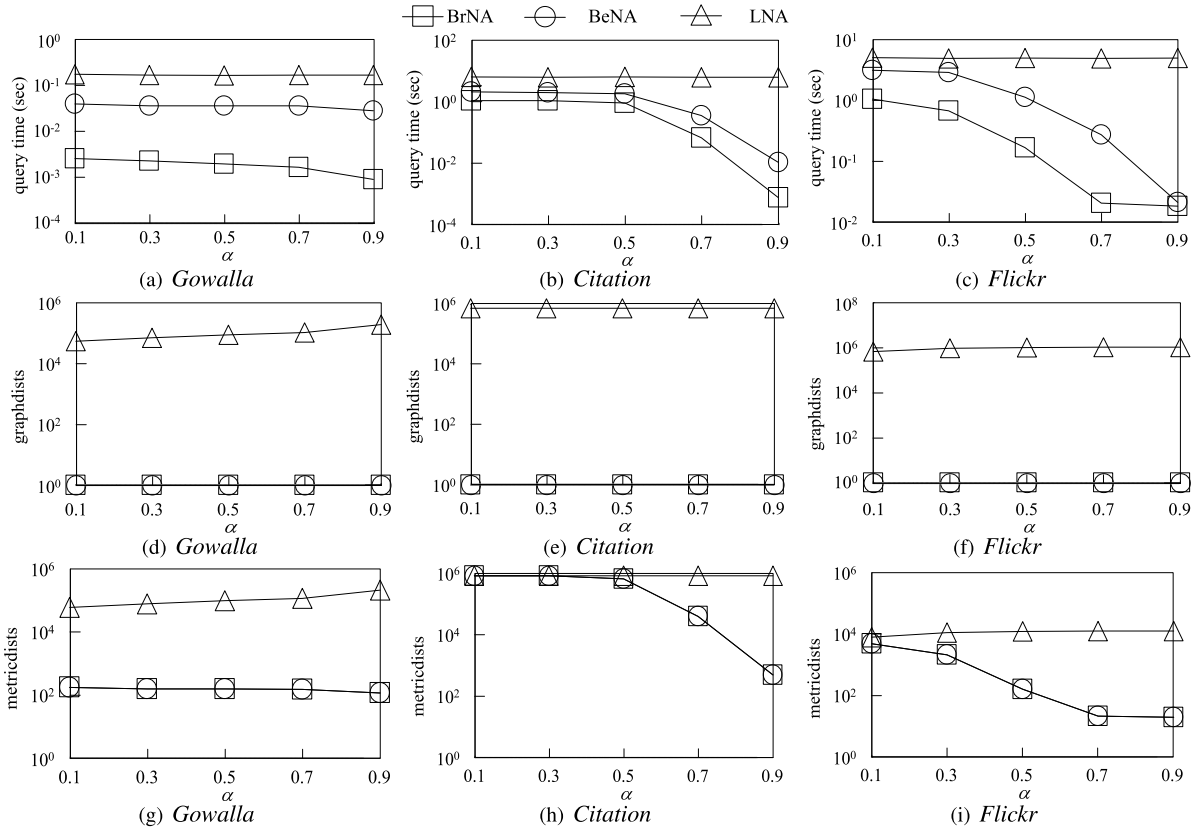



**FIGURE 8.** 30NN ($k = 30$) query performance vs. $\alpha$.

**TABLE 3.** Statistics of the datasets used.

| Graph | $\|V\|$ | $\|E\|$ | Dim. | $d_M$ |
|---|---|---|---|---|
| *Gowalla* | 196,591 | 950,327 | 2 | $L_2$-norm |
| *Flickr* | 1,146,841 | 4,263,536 | 282 | $L_2$-norm |
| *Citation* | 758,751 | 4,073,787 | $2 \sim 41$ | *Jaccard distance* |

**TABLE 4.** Parameter settings.

| Parameter | Range |
|---|---|
| search radius $r$ (% of $d^+$) | 2, 4, 8, 16, **32**, 64 |
| the number $k$ of objects requested | 10, 20, **30**, 40, 50 |
| the number of pivots $\|P\|$ | 1, **3**, 5, 7, 9 |
| the parameter $\alpha$ | 0.1, 0.3, **0.5**, 0.7, 0.9 |

Each measurement we report is the average of 50 queries. All the algorithms are implemented in C++, and all the experiments run on an Intel Xeon E312xx(Sandy Bridge) 2.10GHz virtual machine with 60GB RAM.

### B. EFFICIENCY OF OUR METHODS

#### 1) RANGE QUERY PERFORMANCE

The first set of experiments evaluates the performance of BeRA and BrRA for supporting range queries on quasi-metric graphs, compared with three baseline algorithms (i.e., LRA, MRA, SRA). Fig. 6 depicts the query results under various $r$ values. Note that, values of MRA and SRA are missed on *Citation* and *Flickr*, because indexes cannot be built due to the large cardinality of the graphs. The first observation is that, BeRA and BrRA outperform baseline algorithms in term of query time, the number of metric distance computations, and the number of shortest path distance computations. The reason is that, BeRA and BrRA traverse the quasi-metric graph only once (i.e., *graphdists* $=$ 1) while baseline algorithms traverse the graph multiple times for computing shortest path distances of unpruned vertices. The second observation is that, among three baseline algorithms, the query time of MRA and SRA is more than LRA. This is because, even though the number of shortest path distance computations of MRA and SRA is fewer than that of LRA, the shortest path distance computations for MRA and SRA are more costly than LRA as discussed in Section IV-D. In addition, the number of metric distance computations of LRA is fewer. The third observation is that, BrRA performs better than BeRA. This is because, BrRA verifies vertices as long as they are visited, resulting in obtaining the final result earlier. As expected, the query time of BeRA and BrRA first ascends and then tends to be stable as $r$ increases, This is because, more vertices cannot be pruned or verified with the growth of $r$, incurring more distance computations. Note that, on *Citation* and *Flickr*, the query time of BeRA and BrRA increases dramatically when $r$ is

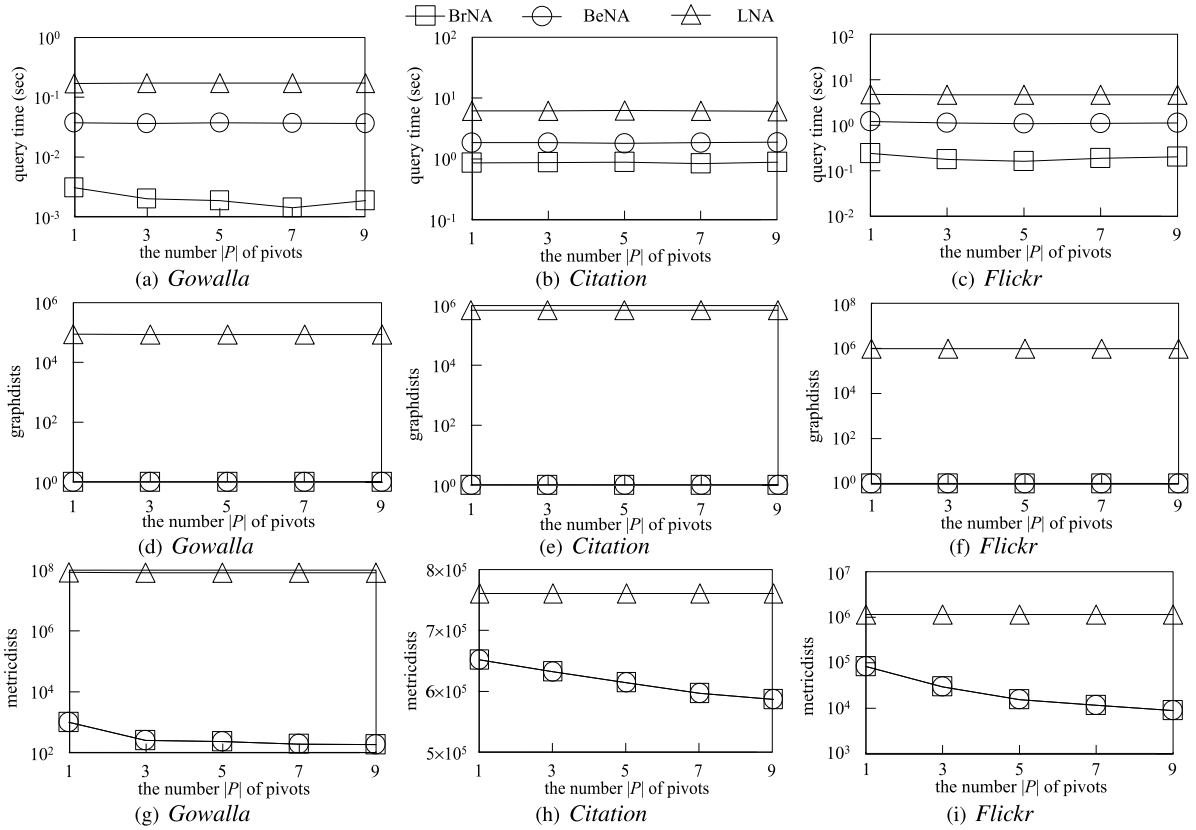



**FIGURE 9.** 30NN ($k = 30$) query performance vs. the number $|P|$ of pivots.

greater than 16%, because, when $r \geq 16\%$, large number of vertices need further verification. Accordingly, the number of metric distance computations sharply grows as shown in Fig. 6(h) and Fig. 6(i).

#### 2) $k$NN SEARCH PERFORMANCE

The second set of experiments verifies the performance of BeNA and BrNA for supporting $k$NN queries, compared with three baseline algorithms (i.e., LNA, MNA, SNA). Fig. 7 shows the results under different $k$ values. As analyzed above for range queries, BeNA and BrNA outperform baseline algorithms, and BrNA performs better than BeNA. For example, on *Flickr*, the query time of BrNA is 6 times fewer than that of BeNA, and is 25 times fewer than that of LNA, when $k$ is set as 50. In addition, the query costs of BeNA, BrNA, and baseline algorithms increase with the growth of $k$ due to larger search space.

#### 3) EFFECT OF $\alpha$

The third set of experiments studies the impact of parameter $\alpha$ on $k$NN ($k = 30$) queries. Note that, the results of range queries have similar performance and hence are omitted. Fig. 8 plots the results. It is observed that, the query costs of BeNA and BrNA drop dramatically while that of LNA increases as $\alpha$ ascends. The reason is that, for LNA, the larger $\alpha$ is, the weaker the pruning ability becomes, incurring more distance computations, as illustrated in Figs. 8(d) through 8(i). For BeNA and BrNA, as $\alpha$ grows, the weight of graph similarity $d_G(u, v)$ increases, and therefore, the pruning power of BeNA or BrNA becomes stronger, resulting in better query performance.

#### 4) EFFECT OF THE NUMBER OF PIVOTS $|P|$

The last set of experiments aims to explore the impact of the number $|P|$ of pivots on $k$NN ($k = 30$) queries. We vary the number $|P|$ of pivots from 1 to 9, and Fig. 9 depicts the results. The first observation is that, for LNA, the query costs are not sensitive to $|P|$. This is because, the pruning power of LNA is weak although the number of pivots grows. The second observation is that, for BeNA and BrNA, the query time first drops and then stays stable or increases as the number of pivots ascends. The reason is that, both the costs and the abilities for pruning or validating vertices grow with more pivots. The third observation is that, the number of metric distance computations drops when the number of pivots grows. This is because, using more pivots, the lower and upper bounds (defined in Definition 4) are tighter and thus the pruning power is stronger.

### C. EFFECTIVENESS OF OUR METHODS

To evaluate the effectiveness of our methods, we select randomly 100 query papers from *Citation* dataset, and for each

**TABLE 5. Quality of the result.**

| $\alpha$ \ $k$ | 5 | 10 | 20 | 30 | 40 | 50 |
|---|---|---|---|---|---|---|
| 0 | 0.21 | 0.20 | 0.21 | 0.22 | 0.24 | 0.26 |
| 0.25 | 0.48 | 0.49 | 0.49 | 0.51 | 0.53 | 0.55 |
| **0.5** | **0.61** | **0.63** | **0.65** | **0.67** | **0.70** | **0.72** |
| 0.75 | 0.35 | 0.36 | 0.39 | 0.40 | 0.40 | 0.40 |
| 1 | 0.30 | 0.30 | 0.29 | 0.30 | 0.32 | 0.34 |

query paper, we merge three 50NN query result sets in the cases when $\alpha = 0.25$, $\alpha = 0.5$, and $\alpha = 0.75$ (i.e., using combined similarity metric), (ii) $\alpha = 0$ (i.e., only considering metric data similarity), and (iii) $\alpha = 1$ (i.e., only considering graph similarity), resulting in an answer set $S_A$. Then, we assign every query paper with the merged answer set $S_A$ to a group of users $U = \{u_i \mid 1 \leq i \leq 9\}$ by using the crowdsourcing strategy, and every user $u_i$ selects 50NN answers (i.e., user-wanted answers) $S_{u_i}$ for each query paper. Based on these, the quality of the returned result set $S_r$ can be measured as $\frac{1}{|U|}\sum_{u_i \in U}\frac{|S_{u_i} \cap S_r|}{|S_r|}$. Table 5 reports the quality of the result set returned by our methods under $\alpha = 0, \alpha = 0.25$, $\alpha = 0.5$, $\alpha = 0.75$, and $\alpha = 1$, respectively. It is observed that, the quality of the result sets under $\alpha = 1$ and $\alpha = 0$ is much lower than that of the result sets under $\alpha = 0.25$, $\alpha = 0.5$, and $\alpha = 0.75$, meaning that the combined similarity metric achieves high result quality. This confirms that considering both graph similarity and metric data similarity in node similarity search is significant and effectiveness. In addition, our approach is much more flexible, and thus, users can obtain preferred results by tuning parameter $\alpha$. It is recommended that, $\alpha = 0.5$ is a good choice when considering both graph similarity and metric data similarity, because it achieves the highest quality of the result as shown in Table 5.

In summary, our proposed methods (i.e., best-first method and breadth-first method) are significantly faster than three baseline approaches in answering similarity search on quasi-metric graphs, and there is no need for them to build any complicated indexes.

## VII. CONCLUSION

In order to support similarity search on graphs (e.g., geo-social network, citation graph, social image graph, etc.) based on both metric data similarity and graph similarity, we introduce the new notion of quasi-metric graph and study similarity search (including range query and $k$NN search) on quasi-metric graphs. We propose three baseline methods and present two simple but efficient methods, i.e., best-first method and breadth-first method, which traverse the quasi-metric graph following the best-first paradigm and the breadth-first paradigm, respectively. In addition, several pruning and validating techniques are developed to avoid unnecessary evaluation. Extensive experiments using three real datasets verify the effectiveness and efficiency of our methods. Compared with three baseline methods, both best-first method and breadth-first method support more efficient similarity search without constructing complicated graph indexes or metric indexes. In the future, we plan to extend our algorithms to various distributed environments such as Pregel and Spark. Another possible direction for future work is to utilize other similarity metrics (e.g., SimRank, PPR, etc.) to measure the similarity between vertices in the graph.

## REFERENCES

[1] E. Chávez, G. Navarro, R. A. Baeza-Yates, and J. L. Marroquín, "Searching in metric spaces," *ACM Comput. Surv.*, vol. 33, no. 3, pp. 273–321, 2001.
[2] G. R. Hjaltason and H. Samet, "Index-driven similarity search in metric spaces," *ACM Trans. Database Syst.*, vol. 28, no. 4, pp. 517–580, 2003.
[3] H. Samet, *Foundations of Multidimensional and Metric Data Structures*. San Mateo, CA, USA: Morgan Kaufmann, 2006.
[4] S. Khemmarat and L. Gao, "Fast top-$k$ path-based relevance query on massive graphs," in *Proc. ICDE*, Mar./Apr. 2014, pp. 316–327.
[5] Y. Wu, R. Jin, and X. Zhang, "Efficient and exact local search for random walk based top-k proximity query in large graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 5, pp. 1160–1174, May 2016.
[6] A. Silva, W. Meira, Jr., and M. J. Zaki, "Mining attribute-structure correlated patterns in large attributed graphs," *Proc. VLDB Endowment*, vol. 5, no. 5, pp. 466–477, 2012.
[7] Z. Xu, Y. Ke, Y. Wang, H. Cheng, and J. Cheng, "A model-based approach to attributed graph clustering," in *Proc. SIGMOD*, 2012, pp. 505–516.
[8] Y. Zhou, H. Cheng, and J. X. Yu, "Graph clustering based on structural/attribute similarities," *Proc. VLDB Endowment*, vol. 2, no. 1, pp. 718–729, Aug. 2009.
[9] Y. Zhou, H. Cheng, and J. X. Yu, "Clustering large attributed graphs: An efficient incremental approach," in *Proc. ICDM*, Dec. 2010, pp. 689–698.
[10] I. Kalantari and G. McDonald, "A data structure and an algorithm for the nearest point problem," *IEEE Trans. Softw. Eng.*, vol. SE-9, no. 5, pp. 631–634, Sep. 1983.
[11] H. Noltemeier, K. Verbarg, and C. Zirkelbach, "Monotonous bisector* trees—A tool for efficient partitioning of complex scenes of geometric objects," in *Data Structures and Efficient Algorithms*. Berlin, Germany: Springer, 1992, pp. 186–203.
[12] S. Brin, "Near neighbor search in large metric spaces," in *Proc. VLDB*, 1995, pp. 574–584.
[13] J. K. Uhlmann, "Satisfying general proximity / similarity queries with metric trees," *Inf. Process. Lett.*, vol. 40, no. 4, pp. 175–179, 1991.
[14] G. Navarro, "Searching in metric spaces by spatial approximation," *VLDB J.*, vol. 11, no. 1, pp. 28–46, 2002.
[15] L. Aronovich and I. Spiegler, "CM-tree: A dynamic clustered index for similarity search in metric databases," *Data Knowl. Eng.*, vol. 63, no. 3, pp. 919–946, 2007.
[16] P. Ciaccia, M. Patella, and P. Zezula, "M-tree: An efficient access method for similarity search in metric spaces," in *Proc. VLDB*, 1997, pp. 426–435.
[17] C. Traina, Jr., A. Traina, B. Seeger, and C. Faloutsos, "Slim-trees: High performance metric trees minimizing overlap between nodes," in *Proc. EDBT*, 2000, pp. 51–65.
[18] V. Dohnal, C. Gennaro, P. Savino, and P. Zezula, "D-index: Distance searching index for metric data sets," *Multimedia Tools Appl.*, vol. 21, no. 1, pp. 9–33, 2003.
[19] V. Dohnal, C. Gennaro, and P. Zezula, "Similarity join in metric spaces using eD-index," in *Proc. DEXA*, 2003, pp. 484–493.

[20] E. Chávez and G. Navarro, "A compact space decomposition for effective metric indexing," *Pattern Recognit. Lett.*, vol. 26, no. 9, pp. 1363–1376, 2005.
[21] G. Navarro and N. Reyes, "Dynamic list of clusters in secondary memory," in *Proc. SISAP*, 2014, pp. 94–105.
[22] R. Paredes and N. Reyes, "Solving similarity joins and range queries in metric spaces with the list of twin clusters," *J. Discrete Algorithms*, vol. 7, no. 1, pp. 18–35, 2009.
[23] J. Almeida, R. da Silva Torres, and N. J. Leite, "BP-tree: An efficient index for similarity search in high-dimensional metric spaces," in *Proc. CIKM*, 2010, pp. 1365–1368.
[24] W. A. Burkhard and R. M. Keller, "Some approaches to best-match file searching," *Commun. ACM*, vol. 16, no. 4, pp. 230–236, 1973.
[25] L. Micó, J. Oncina, and R. C. Carrasco, "A fast branch & bound nearest neighbour classifier in metric spaces," *Pattern Recognit. Lett.*, vol. 17, no. 7, pp. 731–739, 1996.
[26] E. V. Ruiz, "An algorithm for finding nearest neighbours in (approximately) constant average time," *Pattern Recognit. Lett.*, vol. 4, no. 3, pp. 145–157, 1986.
[27] G. Ruiz, F. Santoyo, E. Chávez, K. Figueroa, and E. S. Tellez, "Extreme pivots for faster metric indexes," in *Proc. SISAP*, 2013, pp. 115–126.
[28] R. A. Baeza-Yates, W. Cunto, U. Manber, and S. Wu, "Proximity matching using fixed-queries trees," in *Proc. 5th Annu. Symp. Combinat. Pattern Matching (CPM)*, 1994, pp. 198–212.
[29] T. Bozkaya and Z. M. Özsoyoglu, "Distance-based indexing for high-dimensional metric spaces," in *Proc. SIGMOD*, 1997, pp. 357–368.
[30] P. N. Yianilos, "Data structures and algorithms for nearest neighbor search in general metric spaces," in *Proc. SODA*, 1993, pp. 311–321.
[31] C. Traina, Jr., R. F. Filho, A. J. Traina, M. R. Vieira, and C. Faloutsos, "The omni-family of all-purpose access methods: A simple and effective way to make similarity search more efficient," *Proc. VLDB J.*, vol. 16, no. 4, pp. 483–505, 2007.
[32] T. Skopal, J. Pokorný, and V. Snásel, "PM-tree: Pivoting metric tree for similarity search in multimedia databases," in *Proc. ADBIS*, 2004, pp. 27–37.
[33] D. Novak, M. Batko, and P. Zezula, "Metric index: An efficient and scalable solution for precise and approximate similarity search," *Inf. Syst.*, vol. 36, no. 4, pp. 721–733, 2011.
[34] L. Chen, Y. Gao, X. Li, C. S. Jensen, and G. Chen, "Efficient metric indexing for similarity search," in *Proc. ICDE*, Apr. 2015, pp. 591–602.
[35] P. Zezula, G. Amato, V. Dohnal, and M. Batko, *Similarity Search: The Metric Space Approach*. Berlin, Germany: Springer, 2006.
[36] T. Akiba, Y. Iwata, and Y. Yoshida, "Fast exact shortest-path distance queries on large networks by pruned landmark labeling," in *Proc. SIGMOD*, 2013, pp. 349–360.
[37] S. Ma, K. Feng, J. Li, H. Wang, G. Cong, and J. Huai, "Proxies for shortest path and distance queries," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 7, pp. 1835–1850, Jul. 2016.
[38] M. Potamias, F. Bonchi, C. Castillo, and A. Gionis, "Fast shortest path distance estimation in large networks," in *Proc. CIKM*, 2009, pp. 867–876.
[39] Z. Li, Y. Fang, Q. Liu, J. Cheng, R. Cheng, and J. C. S. Lui, "Walking in the cloud: Parallel SimRank at scale," *Proc. PVLDB*, vol. 9, no. 1, pp. 24–35, 2015.
[40] B. Tian and X. Xiao, "SLING: A near-optimal index structure for SimRank," in *Proc. SIGMOD*, 2016, pp. 1859–1874.
[41] Y. Fujiwara, M. Nakatsuji, H. Shiokawa, T. Mishima, and M. Onizuka, "Efficient ad-hoc search for personalized pagerank," in *Proc. SIGMOD*, 2013, pp. 445–456.
[42] S. Wang, Y. Tang, X. Xiao, Y. Yang, and Z. Li, "HubPPR: Effective indexing for approximate personalized pagerank," *Proc. VLDB Endowment*, vol. 10, no. 3, pp. 205–216, 2016.
[43] J. Neville, M. Adler, and D. Jensen, "Clustering relational data using attribute and link information," in *Proc. IJCAI*, 2003, pp. 689–698.
[44] H. Gao, J. Tang, and H. Liu, "gSCorr: Modeling geo-social correlations for new check-ins on location-based social networks," in *Proc. CIKM*, 2012, pp. 1582–1586.
[45] J. Shi, N. Mamoulis, D. Wu, and D. W. Cheung, "Density-based place clustering in geo-social networks," in *Proc. SIGMOD*, 2014, pp. 99–110.
[46] W. Lu, J. Janssen, E. Milios, N. Japkowicz, and Y. Zhang, "Node similarity in the citation graph," *Knowl. Inf. Syst.*, vol. 11, no. 1, pp. 105–129, 2007.
[47] X. Zhou, L. Chen, Y. Zhang, L. Cao, G. Huang, and C. Wang, "Online video recommendation in sharing community," in *Proc. SIGMOD*, 2015, pp. 1645–1656.

**TIANMING ZHANG** received the B.S. and M.S. degrees in computer science from Northeastern University, China, in 2012 and 2014, respectively. She is currently pursuing the Ph.D. degree with the College of Computer Science, Zhejiang University. Her research interest includes graph databases especially for parallel (massive) graph processing.

**YUNJUN GAO** received the Ph.D. degree in computer science from Zhejiang University, China, in 2008, where he is currently a Professor with the College of Computer Science. His research interests include spatial and spatio-temporal databases, metric and incomplete/uncertain data management, graph databases, spatio-textual data processing, and database usability. He is a member of the ACM.

**LU CHEN** received the B.S. degree in software engineering from Southeast University, China, in 2011, and the Ph.D. degree in computer science from Zhejiang University, China, in 2016. She is currently an Assistant Professor with the College of Computer Science and Engineering, Aalborg University, Aalborg. Her research interests include indexing and querying metric spaces, heterogeneous graph analysis, and trajectory data processing.

**GUANLIN CHEN** is currently a Professor with the School of Computer and Computing Science, Zhejiang University City College, China. He is a member of the Cloud Computing Expert Committee of China Communications Society and the Safety Technology Professional Committee of Zhejiang Computer Information System Security Association. He is an expert of Hangzhou Industry and Information Technology. His research interests include computer networks, information and network security, and urban management information.

**SHILIANG PU** received the Ph.D. degree from the National Academy of Sciences, French. He is currently the Senior Vice President and also the Dean of the Research Institute of Hikvision, Hikvision Digital Technology Company, and is responsible for the company's technical research in the field of artificial intelligence and big data. He was a recipient of many honors, such as the 19th Qiushi Excellence Outstanding Youth Award in Zhejiang Province, Zhejiang, and the Hubei Science and Technology Progress Award.