OR4NN: Preprocess for Scalable K Nearest Neighbor Join on Metric Data in a Distributed Computing Environment

January 2026

Version history

Version	Date	Description	Authors
1.0	12/01/2026	Initial version	Sheng Zhou

For further information on this report please contact:

Telephone: +44(238055782)
Mobile: +44(0)
Email: Sheng.Zhou@os.uk

Contents

Abstract 5

1 Introduction 6

2 Object Range for Nearest Neighbours (OR4NN) 8

2.1 Some Properties of KNN 9

2.2 Object Range for a Point Query 9

2.3 Object Range for Multi-Point Query 11

2.4 Object Range as a Facility for Scalable KNN 11

3 Object Range in 2D Euclidean Space 13

3.1 Extended Object Range for Multi-Point Objects 13

3.2 Extended Object Range for Line Segments 15

3.3 Extended Object Range for triangles and polygons 16

3.4 Query Space Partition and Object Range for Partitions 17

4 Object Range for KNN on Networks 19

4.1 The Network KNN Problem 19

4.2 Object Range for a Network Node 20

4.3 Network Partition and Boundary Nodes 21

4.4 Object Range for a Sub-Network 22

4.5 Spatial Partition for Spatial Networks 24

4.6 Object Range for Spatial Network KNN 25

4.7 Spatial Network KNN Join 26

4.8 Directed Network Links 27

4.9 Dynamic Breadth-First Search for KNN 28

5 Experimental Implementations 29

6 Summary and Discussion 30

Acknowledgments 32

Appendix A 33

References 36

Abstract

This research introduces OR4NN, a conceptual framework with a pre-processing strategy to facilitate efficient large scale KNN join. Query space is divided into query partitions. For each query partition, a candidate set of K objects from the object set to be queried are allocated. An "object range" is computed based on the extent of the partition and the candidate object set in a way which guarantees for any query inside the partition, its KNN will be found from objects intersecting the object range of the partition. A conceptual framework of OR4NN for metric data is described using set theory. Practical methods to construct object range for data in 2D Euclidean Space on Cartesian Distance as well as network data are presented with proof of correctness. An experimental implementation for geospatial data which generated favorable outputs in real world applications is also introduced.

1 Introduction

KNN (K-Nearest Neighbors) query is the problem to find the K objects from an object set (e.g. all schools in a country) that are closest to a given query (e.g. a home). Closeness is expressed as a dissimilarity function, which, in case of metric space, is a distance metric (e.g., Euclidean distance). KNN join query refers to for each query in a query set finding its K nearest neighboring objects from an object set [1].

In a centralised database system, indexes such as R-tree are often utilised to perform single KNN query [2]. For large scale KNN-Join in distributed parallel computing environments where the object set is too large to be handled by a single computing unit and has to be partitioned, the K nearest neighbors of a query may sit in multiple partitions. In a worst-case scenario, all partitions have to be examined in order to return the exact KNN results for a query. Several approaches have been proposed to address this issue. For example of geospatial dataset, exact results are sacrificed for better performance using space-filing order for locality representation [3]; a global partition index is utilised to safely eliminate unnecessary examination on partitions beyond a mathematically proven distance bound [4].

In this research we introduce OR4NN, a conceptual framework with a pre-processing strategy that tackles the problem of parallel KNN join on metric datasets from a different perspective. Our method will first divide the query space into several subset regions (to avoid confusion, we do not use "partition" now). Given a pre-defined K, for each region we compute an "object range" which is also as a subset region of the query space. Subsequently, the *k*NN for any queries (for a *k* up to K) in a region will be found from the set of objects intersecting the object range for this region, independent of the remainder of the entire object set. Obviously, this property will meet the requirements of the shared-nothing architecture perfectly.

Another obvious benefit of this approach is that object range information (either the ranges or the objects collected for each region) may be re-used for subsequent multiple KNN joins on the same object set.

This method is originated from a previous work on large KNN join and continuous moving object KNN queries in a spatial DBMS environment [5].

2 Object Range for Nearest Neighbors (OR4NN)

Given a metric space (X, d) where X is a non-empty set and $d: X \times X \rightarrow R$ is a metric (a.k.a. distance function) defined on X, we define the following notations:

- Dist(q, o) ($q, o \in X$): the original metric of distance between two points *q* and *o* in X
- Object $O = \{o_i \mid i = 1, 2, \ldots\} \subset X$: an object is a point set on X where o_i denotes a point in X. The number of points in a subset may be finite or infinite. A point object is a set that contains only one point. A multi-point object may be a finite collection of discrete points, or a set of infinite points (e.g., linestrings or polygons in geospatial dataset), or a mixture of both.
- Query $Q = \{q_i \mid i = 1, 2, \ldots\} \subset X$: a query is a point set on X where q_i denotes a point in X. Similarly, we have the notations of point query and multi-point query.
- Object Set $O^P = \{O_i \mid i = 1, n\}$: a set of objects to be queried against
- Query Set $Q^P = \{Q_i \mid i = 1, m\}$: a set of queries

Subsequently, we define the following extended metrics:

- MinDist(q, O) ($q \in X, O \subset X$) = $\inf\{\mathrm{Dist}(q,o):o\in O\}$: minimum distance between a point and a set, which is the distance between a point and the nearest point in the set.
- MinDist(Q, O) ($Q, O \subset X$) = $\inf\{\mathrm{Dist}(q,o):q\in Q,o\in O\} = \inf\{\mathrm{MinDist}(q,O):q\in Q\}$
- MinDist(Q, O^P) = $\inf\{\mathrm{MinDist}(Q,O):O\in O^P\}$
- MinDist(Q^P, O^P) = $\inf\{\mathrm{MinDist}(Q,O^P):Q\in Q^P\}$
- MaxDist(q, O) ($q \in X, O \subset X$) = $\sup\{\mathrm{Dist}(q,o):o\in O\}$: maximum distance between a point and a set, which is the distance between a point and the furthest point in the set.
- MaxDist(Q,O) = $\sup\{\mathrm{Dist}(q,o):q\in Q,o\in O\} = \sup\{\mathrm{MaxDist}(q,O):q\in Q\}$

- $\text{MaxDist}(Q, O^P) = \sup\{\text{MaxDist}(Q,O):O \in O^P\}$
- $\text{MaxDist}(Q^P, O^P) = \sup\{\text{MaxDist}(Q,O^P):Q \in Q^P\}$
- $\text{MaxMinDist}(Q, O) = \sup\{\text{MinDist}(q,O):q \in Q\}$: the largest minimum distance from any point in Q to the nearest point in O (a.k.a. semi-Harsdorff distance, which is directional).
- $\text{MaxMinDist}(Q, O^P) = \sup\{\text{MaxMinDist}(Q,O):O \in O^P\}$
- $\text{MaxMinDist}(q, O^P) = \sup\{\text{MinDist}(q,O):O \in O^P\}$: the largest minimum distance from a point q to all points in all O in O^P.

In subsequent discussions we will also use the terms object space for the extents of the object set in X and query space for the extent in X where a query may be defined.

2.1 Some Properties of KNN

Given an object set O^P and two query points *p* and *q*, and KNN(p, O^P) denoting the set of *K* nearest neighbors of p in O^P (or, used interchangingly, the space occupied by these objects in X). we have the following statements which are easily proven by contradiction:

- $\text{KNN}(\{p, q\}, O^P) \subseteq \text{KNN}(p, O^P) \cup \text{KNN}(q, O^P)$ (1)
- $\text{KNN}(Q=\{p_i | i = 1, n\}, O^P) \subseteq \cup(\text{KNN}(p_i, O^P) | \, i = 1, n)$ (2)

That is, the KNN of a multi-point query is in the union of the KNN of its constituent points.

2.2 Object Range for a Point Query

Given integer $1 \leq K \leq n$, a point query $Q_q=\{q\}$, and a set of objects $O^K = \{O_i \mid i = 1, K\} \subseteq O^P$ as the **candidate object set**, we may define a set $\text{OR}(Q_q, O^K)$ on X as the **object range** for Q_q at $k \leq K$ w.r.t. O^K:

- $\text{OR}(Q_q=\{q\},O^K)=\{p|\text{Dist}(p,q) \leq \text{MaxMinDist}(q,O^K)\}$ (3)

Subsequently, we have:

- $\text{KNN}(Q_q=\{q\}, O^P) \subseteq \text{OR}(Q_q, O^K)$ (4)

Obviously, OR(Q_q, O^K) is a region in X possessing the shape of a disc (or ball in some literatures) in general terms centred at q with radius MaxMinDist(q, O^K). The KNN of Q_q w.r.t. O^P may be found in the region specified by OR(Q_q, O^K). Note that OR(Q_q, O^K) is in fact the minimum object range of the query point. Any set on X that contains OR(Q_q, O^K) is also an (larger) object range of the query point.

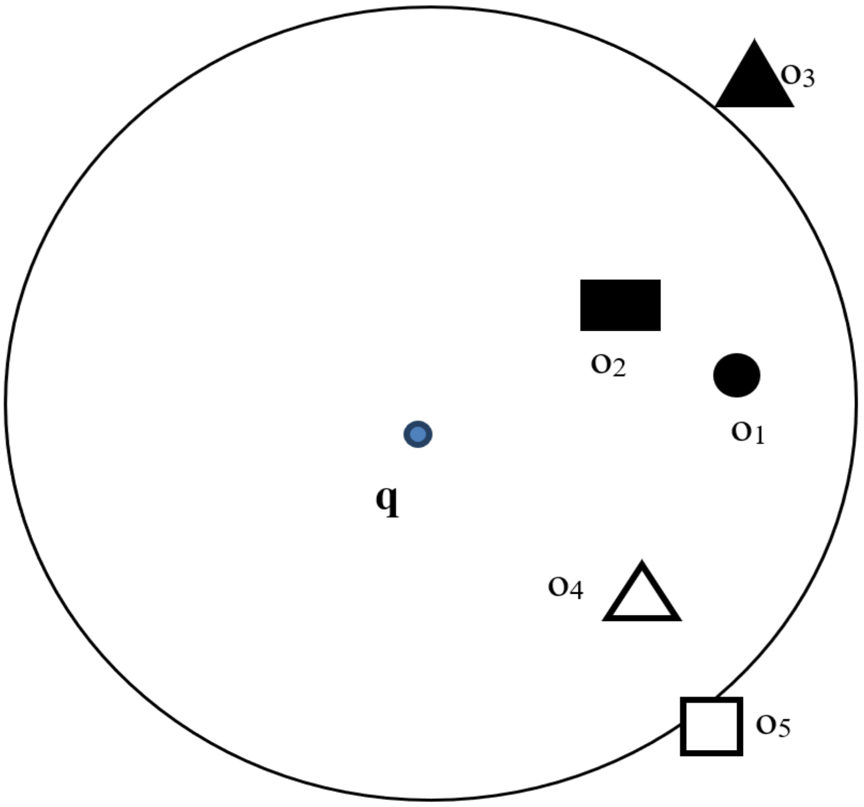

Figure 1: Object range for a point query q w.r.t. O^K={o_1,o_2,o_3}

In Figure 1, the object range OR({q}, O^K) for point query q w.r.t. to candidate object set O^K = {o_1, o_2, o_3} (K=3) is illustrated by the cycle centred at q. There might be other objects (o_4, o_5) from the object set that also intersect with the object range. Nevertheless O^K guarantees that KNN({q}, O^P) will be found inside the object range (in this case, KNN({q}, O^P) = {o_1, o_2, o_4}).

2.3 Object Range for Multi-Point Query

Given a multi-point query $Q_m = \{q_i \mid i = 1, 2, \ldots\}$, we may define an object range for each point in Q_m as $OR(\{q_i\}, O^K)$. From (2) and (4) we may derive:

- $KNN(Q_m, O^P) \subseteq \cup\,(OR(\{q_i\}, O^K)|i = 1, 2, \ldots)$ (5)

That is, the KNN of the multi-point query Q_m w.r.t. O^P is in the union of the object range of all its constituent points w.r.t. O^K. We define the object range of Q_m w.r.t. O^K as:

- $OR(Q_m, O^K) = \cup(OR(\{q_i\}, O^K)|i = 1, 2, \ldots)$ (6)

For a subset $Q_n \subseteq Q_m$, we may define $OR(Q_n, O^K)$ in the same way and it satisfies:

- $OR(Q_n, O^K) \subseteq OR(Q_m, O^K)$ (7)

Subsequently, we may conclude from (5) that:

- $KNN(Q_n, O^P) \subseteq QR(Q_n, O^K) \subseteq OR(Q_m, O^K)$ (8)

2.4 Object Range as a Facility for Scalable KNN

If we view Q_m as a region in X and $Q_n \subseteq Q_m$ a query on Q_m, the KNN of Q_n may be found in $OR(Q_m, O^K)$. This observation leads to a generic conceptual framework for handling large-scale KNN join:

1) Partition the query space into a set of partitions $R^S=\{R_i| \ i = 1, n\}$

2) Given a pre-defined K, for each partition R_i, select and assign a candidate object set $O^K{}_i \subseteq O^P$

3) Compute $OR(\{r_j\}|r_j \in R_i, O^K{}_i)$ for all points $r_i \in R_i$ and union the resulting object ranges to create $OR(R_i, O^K{}_i)$.

4) At query time, for each partition R_i, extract $Q^{R}{}_i =\{Q| \ Q \in Q^P, Q \cap R_i \neq \emptyset\}$ from query set Q^P as queries intersecting the query partition R_i, and extract $O^{R}{}_i =\{O| \ O \in O^P, O \cap OR(R_i, O^K{}_i) \neq \emptyset\}$ from the object set O^P as objects intersecting the query partition's object range $OR(R_i, O^K{}_i)$.

5) For each pair of (Q^R_i, O^R_i), KNN($Q|Q\in Q^R_i$, O^P) may be found from O^R_i using any preferred local KNN algorithms. In other words, each pair of (Q^R_i, O^R_i) may be joined locally and independently, no multi-pass or cross-node communication is required.

6) Joining results from all pairs of (Q^R_i, O^R_i) are returned and post-processed to produce the final output.

For a query that intersect with more than one partition, the reverse of (1) and (2) indicates that the query may be processed in each intersecting partition independently, and all returned results for the same query will be merged, de-duplicated and re-ranked to produce the final KNN for the query.

Step 2) is a crucial step (and potentially a performance bottleneck) in the process. As will demonstrate in following sections, ideally the candidate object set for a query partition should be close to the “centre” of the partition. Consequently, some search/indexing mechanisms on the object set will be required in order to perform candidate object selection efficiently.

The computed object ranges (or the collection of objects retrieved using the range) may also be stored and re-used for multiple subsequent joins between other query sets and the object set.

The object range computed at K will also support queries for any *k* = 1, K.

Step 3) is a conceptual method for object range computation. It is often not feasible (and impossible for continuous query space) to simply follow the definition (5) to construct object range of a query partition by computing the object range for all points in the partition. We will demonstrate how the object ranges may be computed practically for some types of data in sections 3 and 4.

3. Object Range in 2D Euclidean Space on Cartesian Distance

3.1 Extended Object Range for Multi-Point Objects

Before we introduce the method to construct object range for query regions in 2D Euclidean space, we will first make an extension to the definition of object range in (3). Instead of using MaxMinDist(q,O^K), we define the **extended object range** of Q_q={q} w.r.t. O^K as:

- $OR^E(Q_q, O^K) = \{p|Dist(p, q) \leq MaxDist(q, O^K)\}$ (9)

Here MaxDist(q,O^K) is used instead of MaxMinDist(q,O^K). Note that if the object set contains point objects only, OR and OR^E are identical.

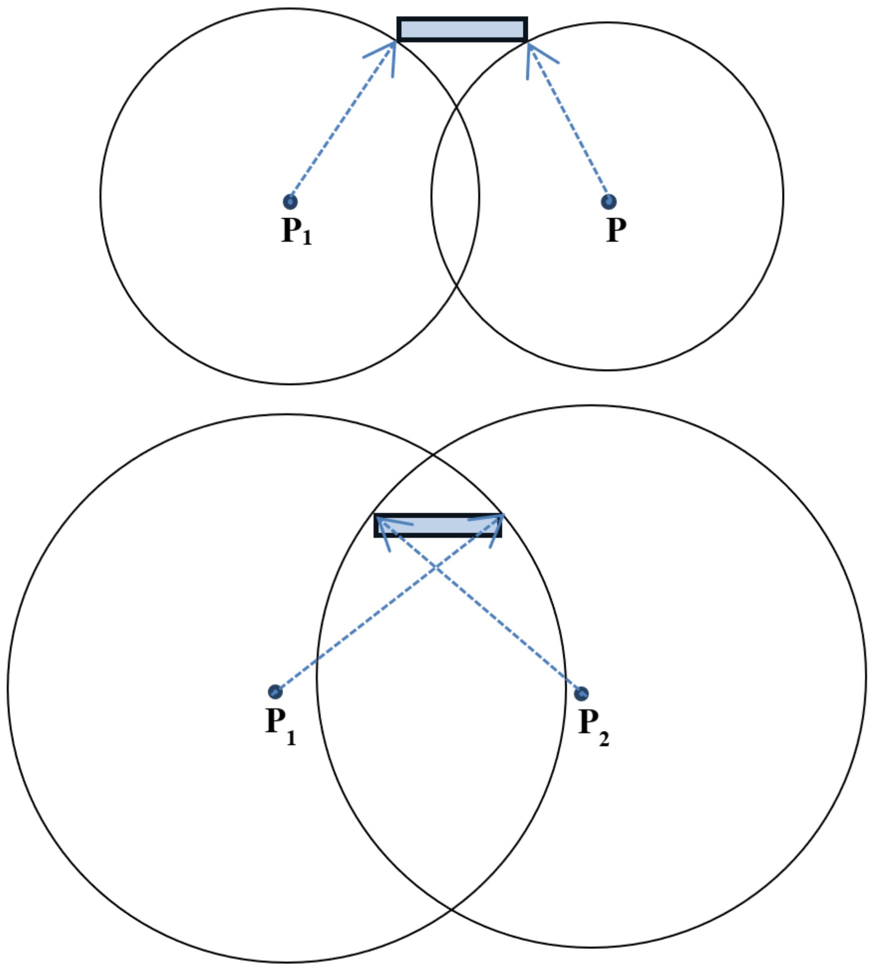

Figure 2: Original (top) and extended (bottom) object ranges for two query points q_1 and q_2

From this definition, we may derive a property of O^K. Given a query point set $Q_m = \{p_i \mid i = 1, m\}$, we have:

- $O^K \subseteq \cap(OR^E(\{p_i\}, O^K))$ (10)

That is, the candidate object set O^K is contained in the intersection of the extended object ranges of all query points. Obviously, for a point object set, the extended object range is identical to the original object range.

3.2 Extended Object Range for point query on Line Segments

Given two points p_s and p_e in metric space (X, d), a line segment in X with the two points as its endpoints may be defined as an infinite point set:

- $L(p_s, p_e)=\{p|Dist(p, p_s)+Dist(p, p_e) = Dist(p_s, p_e): p \in X\}$

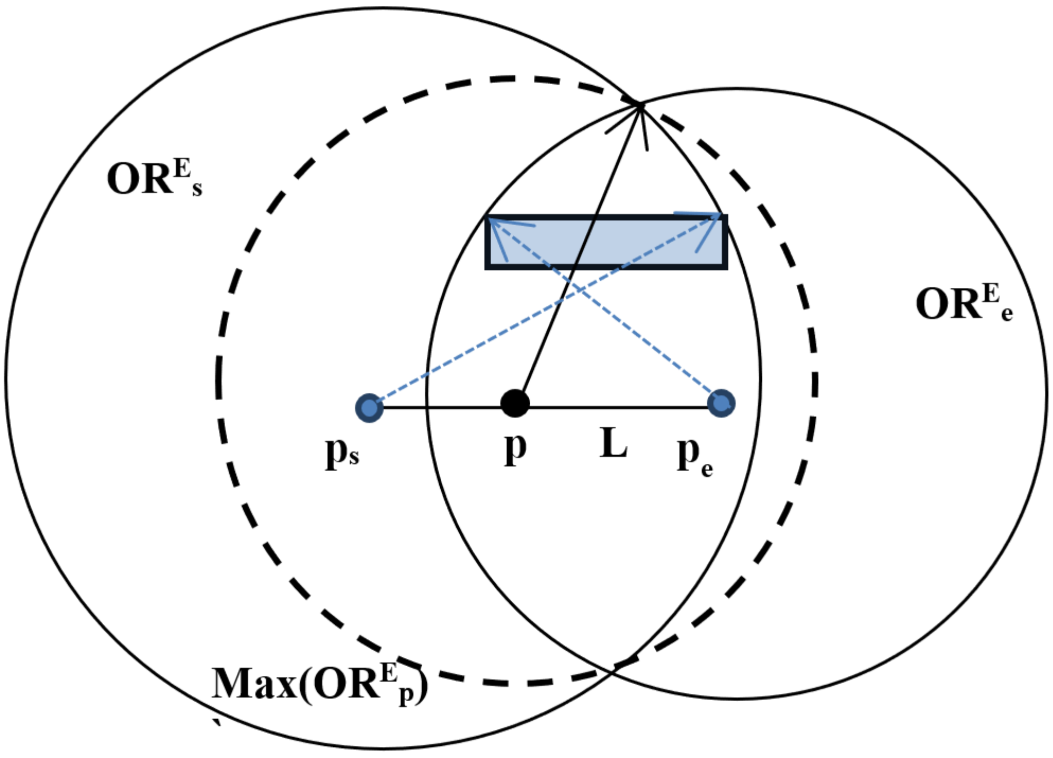

Figure 3: Extended object range for an interior point on a line segment

In particular, $\{p|p \neq p_s \wedge p \neq p_e\}$ are the interior of L. For an interior point p on L, it is easy to prove (see appendix for proof) that in case of 2D Euclidean space (Figure 3):

- $OR^E(p \in L, O^K) \subseteq U(OR^E_s, OR^E_e)$ (11)

And consequently,

- $KNN(p \in L, O^P) \subseteq U(OR^E_s, OR^E_e)$ (12)

That is, the KNN of an interior point on a line is in the union of the object ranges of the two end points of the line.

3.3 Extended Object Range for point query in triangles and polygons

Similarly, given three points p_1, p_2, and p_3 in 2D Euclidean space that are not collinear (i.e. Dist(p_1, p_2) + Dist(p_2, p_3) > Dist(p_1, p_3)), and a point p inside the triangle T(p_1, p_2, p_3), we have:

- $OR^E(p \in T, O^K) \subseteq U(OR^E_1, OR^E_2, OR^E_3)$ (13)
- $KNN(p \in T, O^P) \subseteq U(OR^E_1, OR^E_2, OR^E_3)$ (14)

Proof for (13) is in appendix. Here OR^E_1, OR^E_2 and OR^E_3 are the extended object ranges for vertices p_1, p_2 and p_3 respectively (Figure 4).

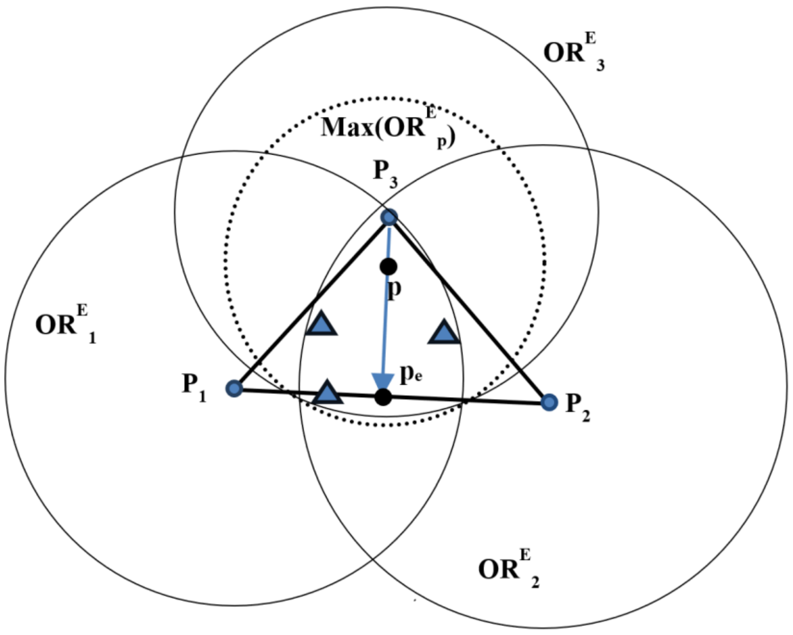

Figure 4: Extended object range for an interior point inside a triangle

Given a polygon PLG(p_1, p_2, ..., p_n) which may be triangulated into a collection of multiple triangles, we may conclude that:

- $OR^E(p \in PLG, O^K) \subseteq U(OR^E_i \mid i = 1, n)$ (15)
- $KNN(p \in PLG, O^P) \subseteq U(OR^E_i \mid i = 1, n)$ (16)

That is, the object range of a polygon in 2D Euclidean space is in the union of the object range of all its boundary vertices (Figure 5). The KNN query for any point sets in the polygon can be found from the set of objects intersecting the object range of the polygon.

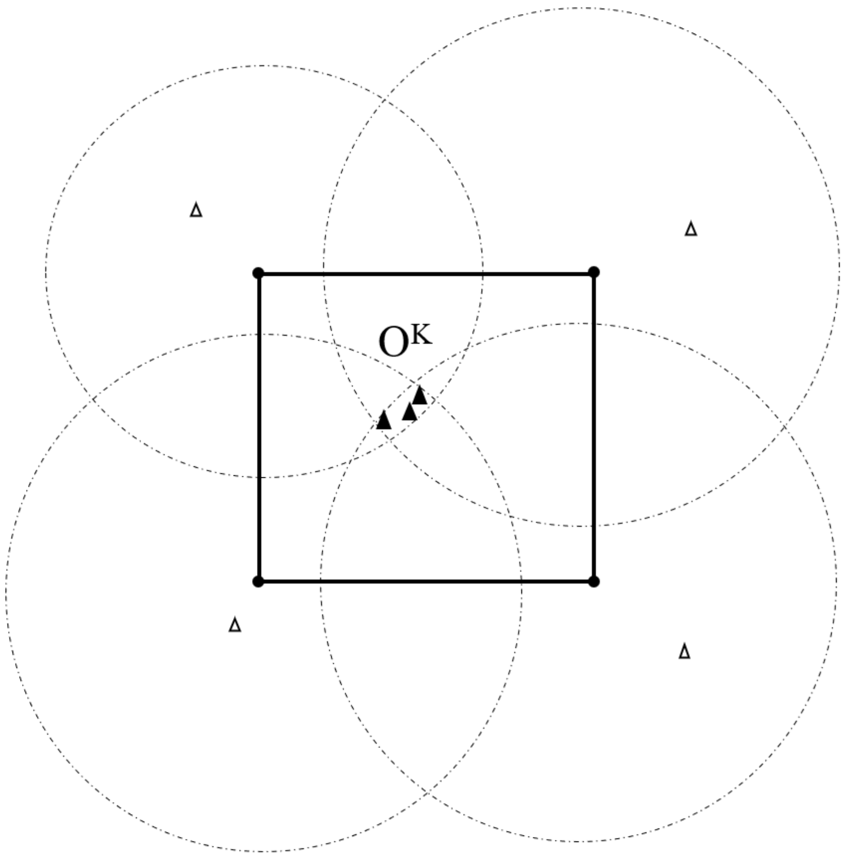

Figure 5: Extended object range for a square partition w.r.t. O^K (K=3). Hollow triangles are other objects inside the OR^E.

3.4 Query Space Partition and Object Range for Partitions

For a 2D Euclidean query space, there are many ways to create a partition (or more generically, a cover, if overlaps are allowed) of the space, for example, a regular grid or Voronoi diagram from sampled objects.

As already mentioned, the object range of a vertex on the boundary of a partition is a disc. For practical implementation, it is easier for the union operation by using the circumscribed polygon (e.g. bounding box or octagon) instead.

A smaller object range for a partition will result in a smaller retrieved object set and is more efficient for subsequent KNN queries on this partition. Given a partition, the size of its object range depends on the assigned candidate object set O^K. It is possible to find an optimal minimised solution but in principle it should be sufficient to select the K objects closest to the centre of the partition as the candidate object set.

Once the object ranges are computed for each partition, KNN join may subsequently be performed between the query set and object set following the steps described in 2.4. Both the object and query sets may contain arbitrary combination of points, lines and polygons. Any local KNN algorithms (e.g. [2]) may be used to find the KNN for queries in a query partition from the objects intersecting the object range of the partition.

The introduction of extended object range also enables a selection of distance metrics (MinDist, MaxDist or MaxMinDist) in KNN query, instead of MinDist only.

4. Object Range for KNN on Networks

In this section we apply the concept of object range to large scale KNN joins on a network.

A network of nodes and links can be represented as a (weighted) graph. Network, node and link are the equivalents of graph, vertex and edge in graph theory. Here we will mainly use the former notations which are more common in application domains but may also use the latter in an interchangeable manner.

A network has a natural metric of the shortest path distance between any two connected points on the network. However, in practice it is common that the path is directional, i.e., Dist(a, b) ≠ Dist(b, a), which violates one of the basic properties of metric space. For now we will assume Dist(a, b) = Dist(b, a) and will discuss the directional issue later.

4.1 The Network KNN Problem

The network KNN problem may be defined as: given a network G(N, L) with N as the node set and L as the link set, $N^O \subset N$ as the object node set and $Q^O \subset N$ as the query node set, for each $q \in Q^O$, find the K nearest neighbours in N^O by shortest path distance.

A network may not necessarily be fully connected. There may not be a path from one node to another node in the network. In such cases, the network may be divided into several components (a.k.a. connected components). Each component as a sub-network is fully connected but there is no connection between any two components. At present also we assume a component contains at least K object nodes, and all queries are represented by nodes on a network. Consequently, a node may be an object node, or a query node, or both. In practice queries could be points or intervals on links. Such cases, however, may be converted to the node case. Given a link l with end nodes n_s and n_e, for any location p on l, we have:

- $KNN(p, N^o) \subseteq U(KNN(n_s, N^O), KNN(n_e, N^O))$ (17)

That is, KNN for a location on a link may be derived from the KNNs for the two end nodes of the link. This also applies to intervals on a link.

4.2 Object Range for a Network Node

Given a query node n_q and a set of object node N^{O_K}, we may define the object range of the node accordingly. Below are a few notations we will use for the definition:

- G.Nodes: node set of network G
- G.Links: link set of network G
- $SD(n_1, n_2)$: the shortest path distance from a node n_1 to a node n_2
- SD(n, l): the smaller value of the shortest path distance from a node n to the two end nodes of a link l
- Node(L): all nodes connected by links in link set L
- Link(N): all links connected to nodes in node set N
- G.BFS_KNN(n, K): the sub-network generated by a Breadth-first style search (on weighted graph) from node n until K nearest nodes are found or the whole network is searched. It contains the shortest paths from the node to the K object nodes as well as other paths searched in the process.

We define the object range of a query node n_q w.r.t. the candidate object node set N^{O_K} as:

- $OR_{Net}(n_q, N^{O\_K}) = G(N_{OR}, L_{OR})$ where $L_{OR} = \{l \mid SD(n_q, l) \leq \sup(SD(n_q, n \in N^{O\_K})\}$ and $N_{OR} = Node(L_{OR})$ (18)

That is, the graph of all links within the maximum shortest path distance from n_q to nodes in N^{O_K}, and all nodes on these links. Here we assume n_q and N^{O_K} are in the same component so they are connected.

A big difference between a network and the Euclidean space is that a network is discrete. In general cases of a network with multiple connected components, it is not guaranteed that a node and a randomly provided object node is connected. On the other hand, if they are connected, the number of paths between a query node and an object node are finite and nodes on the path have inter-dependency. For these reasons, the object range of a network node may be defined in an alternative way:

- $OR_{Net}(n_q, K, N^O) = G.BFS\_KNN(n_q, K)$ (19)

That is, the candidate object set is not provided but discovered by a breadth-first style search process. In fact, if we already know the KNN object nodes for the query node and then apply definition (18), we will get the same sub-network.

4.3 Network Partition and Boundary Nodes

A network G(N, L) may be partitioned into several sub-networks $\{G_i(N_i, L_i)|_I = 1, n\}$. Figure 6-A illustrates a network, 6-B describes the conventional approach of partitioning the network into two sub-networks by way of removing some links from the network. In particular, the nodes are boundary nodes if they are nodes of the removed links originally connecting the two sub-networks. Other nodes in the sub-network are internal nodes. Links connecting boundary nodes in two sub-networks are regarded as boundary links. These boundary links are not included in any sub-networks partitioned by the conventional method (Figure 6-B).

It is obvious that the boundary nodes "shadow" the internal nodes of the sub-network:

- A path from n_{int}, an internal node of a sub-network to a node n_{ext} in another sub-network will pass at least one boundary node n_{bnd}. If the path is the shortest path between n_{int} and n_{ext}, it overlaps the shortest path between n_{bnd} and n_{ext}. (20)

In fact, a path to another internal node may also pass boundary nodes of the sub-network.

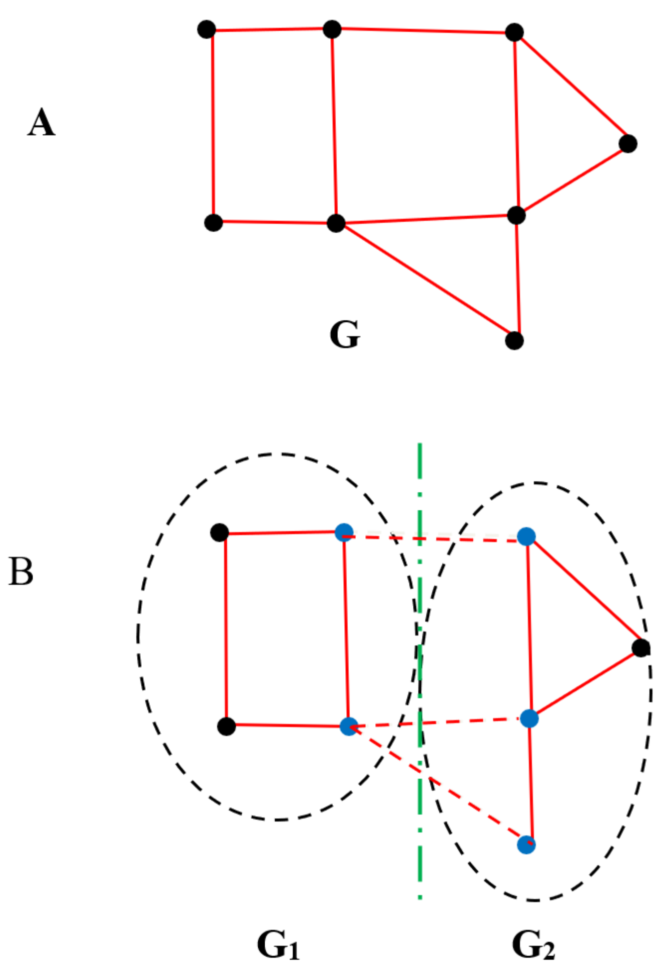

Figure 6: Network partition and boundary nodes

4.4 Object Range for a Sub-Network

Given a sub-network $G_i(N_i, L_i)$ partitioned from a network G(N,L) with N_i^N as its boundary node set, and N^O as the object node set of the whole network G(N,L), we define the object range for a sub-network as:

- $OR_{net}(G_i, K, N^O) = U(G_i, \{OR_{net}(n_j \in N_i^N, K, N^O):j = 1, m\}) = U(G_i, \{G.BFS\_KNN(n_j \in N_i^N, K):j = 1, m\})$ (21)

That is, the object range of a sub-network is a network which is the union of the sub-network and the object ranges of all its boundary nodes. Here the union operation merges the node sets and link sets of two or more sub-networks respectively to form a new sub-network.

Subsequently, for object set $N^O \subset N$, we have:

- $KNN(n \in N_i, N^O) \subseteq OR_{net}(G_i, K, N^O).Nodes$ (22)

That is, for any query node in the sub-network, its network KNN may be found in the object range of the sub-network as defined in (21).

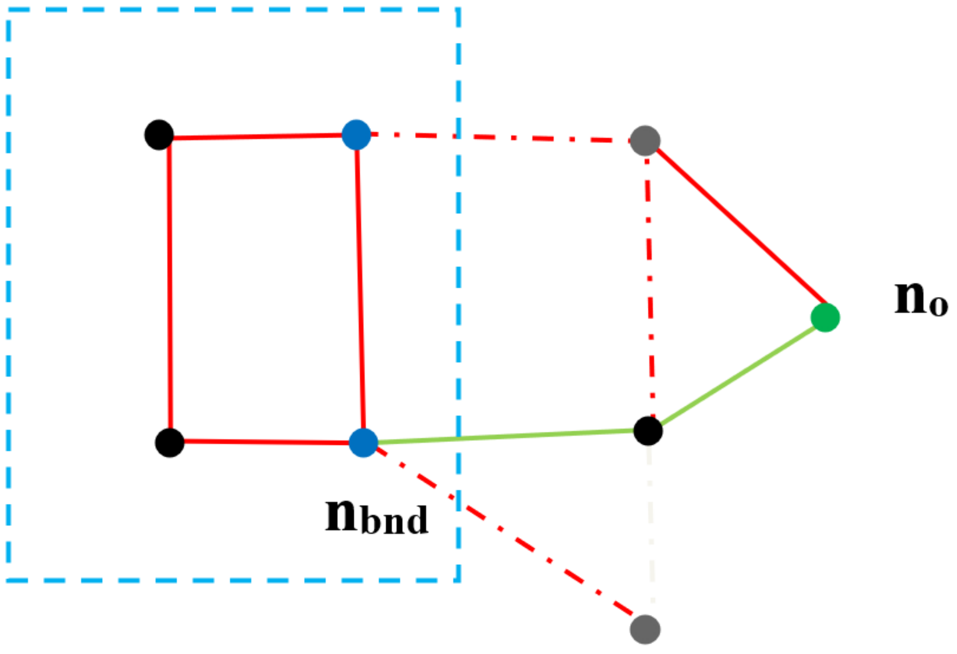

Figure 7: Simplification of object range for boundary nodes

Based on (20), we may further simplify the object range for a sub-network. If a KNN object node (n_o) for a boundary node (n_{bnd}) is outside the sub-network (inside the dashed rectangle), we only need to include the nodes and links on the shortest path (Figure 7, links in green from n_{bnd} to n_o) to the object node in the object range for the sub-network and omits other nodes and links (dashed links in red) discovered during the BFS style search.

Here we have used multiple candidate object sets (indeed, one set per boundary node) instead of a single set in previous discussion to create a more compact object range for a sub-network. This is possible mainly due to the discrete and finite nature of networks.

Recalling the discussion on step 2) in section 2.4, the use of BFS style search to discover KNN nodes for boundary nodes implies that we require an efficient access method beyond a sub-network during the pre-process stage.

The object range of a partitioned sub-network is a discrete and finite structure with a limited number of nodes. Consequently, we may pre-compute and store the KNN for all nodes in the partition. At query time, queries on nodes may be answered immediately; queries associated with locations on a link may also be answered efficiently using (17) (i.e., via the KNNs of the two nodes of the link).

As already stated, the discussion in this section assumes that all links are un-directional (hence metric space properties are met). Also, all objects and queries are on existing nodes in the network, therefore we may partition the network to create sub-networks without boundary links (figure 6-B). For many applications this is not the most suitable model. We will show how this model may be extended to cover some of the scenarios.

4.5 Spatial Partition for Spatial Networks

A spatial network (e.g. transportation network) is a graph whose nodes and links are point sets with locations defined in coordinate systems in Euclidean space. Consequently, these links and nodes may be selected by spatial queries.

The spatial network KNN problem is defined as:

- Given a network G(N, L) with N as the node set and L as the link set, $N^O \subset N$ as the object set and $Q = \{q_i \in l_j \mid q_i \in Q, l_j \in L\}$ as the query set in the form of a set of locations on L, for each $q \in Q$, find K nearest neighbours in N^O by shortest path distance. (23)

Obviously, by definition, q is not necessarily on a node. The reason we use locations on links rather than nodes in network as queries is to better support spatial partition of networks as well as queries with arbitrary spatial locations that are not on existing nodes in the network.

An example of such scenarios is road network and address points. Address points are not on the network. However, we may assign a location on the network for each address (e.g., by proximity or projection). In the discuss below, we assume the query location assignment on links is performed in a separate process. Indeed, this assignment process itself may be a nearest neighbour query to which the methods we introduced in section 3 may be applied.

We may also use locations on links for objects. However, this will greatly increase the complexity of design and implementation comparing to simply adding objects as nodes into the network.

With locations of nodes and links in a spatial network, it may be partitioned by any spatial partition scheme (e.g., a regular grid). Given a partition geometry, all nodes and links intersecting the partition will be collected; nodes of the collected links, if outside the partition, are also collected (Figure 8).

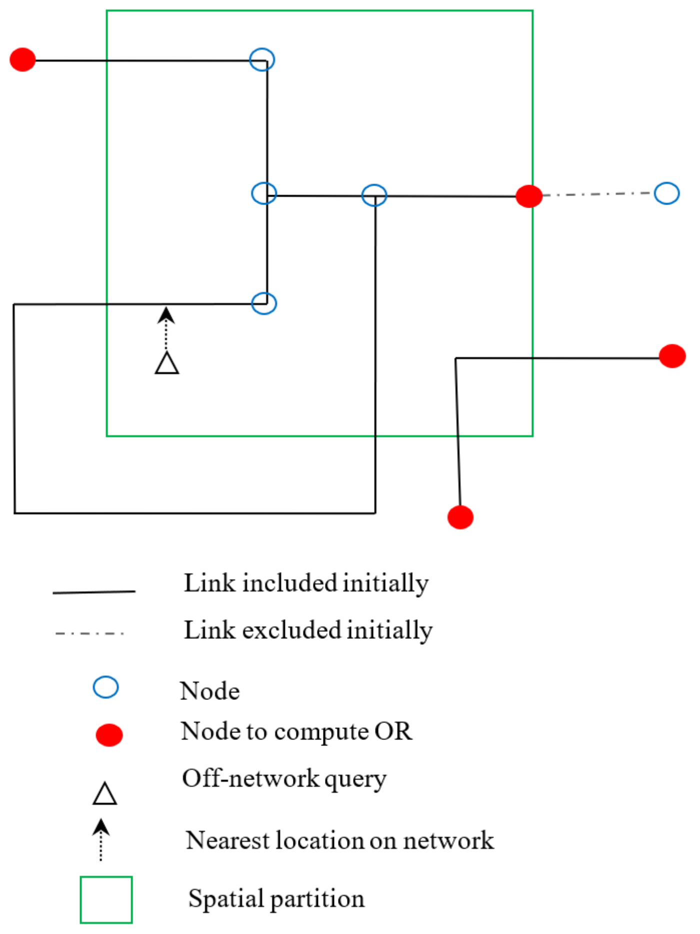

Figure 8: Partition of a spatial network

Note the special case where both nodes of a link are outside a partition but the interior of the link intersects with the partition. This case is due to the spatial intersection between partition and links. It does not appear in generic network partition where a link is processed as a single entity.

4.6 Object Range for Spatial Network KNN

After the spatial network is partitioned with nodes and links associated to each partition, we may now compute the object range for each partition.

Different from the generic network cases we described previously, the nodes to compute object ranges in a spatial network context are: nodes on partition boundary, the node **outside** the partition of a boundary link where the other node is inside the partition, both nodes of the link that intersects the partition but both nodes are outside (the solid dots in figures 8 illustrate the nodes selected for object range computation).

Below is the outline of the process:

1) Step 1: Apply the chosen spatial partition scheme to partition the spatial network and allocate intersecting nodes and links for each partition (remark: nodes and links may appear in the sub-network of more than one partitions)

2) Step 2: For each partition, find all intersecting nodes and links to create the initial sub-network.

    a. Step 2-1: Identify the node for object range computation by one of the two methods described above and compute object range for each of them via a breadth-first style network search process.

    b. Step 2-2: Merge computed node object range with initial sub-network to form the final object range for the partition (in the form of a network).

3) Step 3: Store the computed object range for partitions or pre-compute the KNN for all nodes inside the partition.

Remark 1: there are two options for storing object range information. We may store all the links and nodes in the object range sub-network along with the partition. Alternatively, due to the spatial nature of the network, we may compute the convex hull (or more compact, the concave hull) of all nodes and links in the object range sub-network and store the hull geometry with the partition. In query time, the hull geometry may be used to retrieve the nodes and links of the object range via spatial query.

Remark II: A more extreme approach is to pre-compute KNN for all nodes contained by the partition (e.g., we can use Floyd–Warshall algorithm for finding shortest path distance between all pairs of nodes in the object range sub-network). If we only need the identities of the KNN object nodes, the required additional storage is minimal and we don’t need to store the object range at all.

4.7 Spatial Network KNN Join

To process queries in a partition, we follow the steps below:

1) Step 1: Load the stored object range links and nodes for a partition; or use the hull geometry of the object range to retrieve all links and nodes via spatial query

2) Step 2: Build a graph using the links and nodes from step 1, and use preferred network shortest path algorithm to compute KNN for each query location.

If the KNN for all nodes are pre-computed, we will only need to retrieve all links intersecting a partition and all nodes of these links. We may then apply (17) to get KNN for any query location that is associated to a link intersecting the partition (as mentioned previously, we assume association of off-network query locations to links is performed in a separate process; otherwise, online association will hugely increase the complexity of the process since a location inside a partition may be associated to an unknown link outside the partition).

4.8 Directed Network Links

So far we have restricted our discussion on un-directional network links in order to satisfy the property of a metric space. In fact, this restriction may be relaxed for many applications that demands shortest path distance on one direction only (i.e., from a query location to an object node).

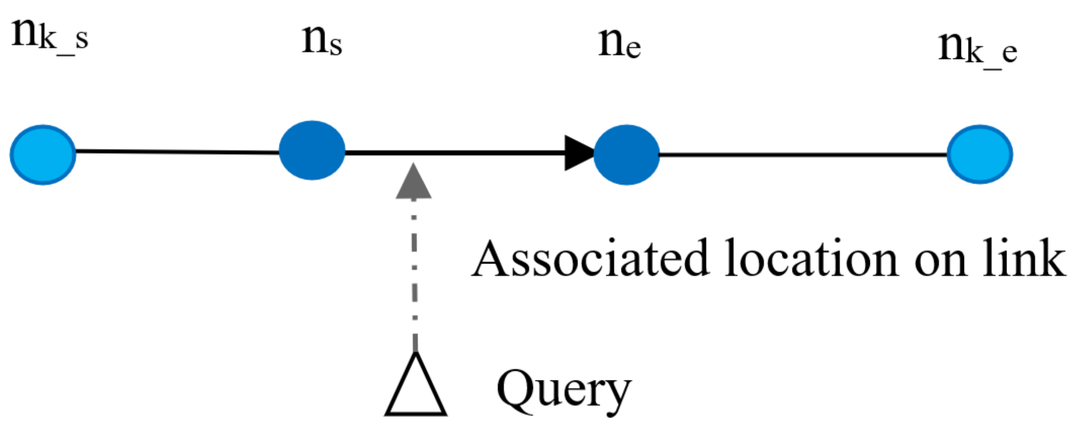

Figure 9: KNN query location on a directed link

In Figure 9, the link $l(n_s, n_e)$ is a directed link. For a query associated to the link, its KNN is the same as $KNN(n_e) = n_{k\_e}$ rather than being selected from the union of $KNN(n_s)$ and $KNN(n_e)$ since there is no path from the query to n_s without passing n_e.

An interesting question in a directed network context is that the distances from the query to the object and from the object to the query may be different. Depending on the specified directionality, the breadth-first style search will need to use either in-direction (w.r.t. query source) or in-opposite-direction weight/distance.

4.9 Dynamic Breadth-First Search for KNN

In this we discuss the potential implementation of the breadth-first search algorithm we mentioned previous. This algorithm is required to discover the KNN for boundary nodes of a partition. The difference between this algorithm and a conventional graph search algorithm is that we do not know a part of the graph and may require access to other part of the entire graph dynamically.

We suggest that a modified version of the well-known Dijkstra (which is a breadth-first search algorithm) will meet our requirements.

Given a sub-network, we assume we are able to know whether a node has links not in current sub-network but are connected to it and we are able to efficiently retrieve these links (and nodes on these links not currently in the sub-network).

For a priority-queue based implementation of Dijkstra algorithm, two modifications to the original algorithm will be made:

1) When a node is popped from the priority-queue, it will be checked and if it has connecting links not currently in the sub-network, the links will be retrieved and add to the sub-network, and nodes from the links not currently in the sub-network will be added dynamically to the priority-queue.

2) The algorithm will terminate if:

    a. K object nodes are discovered and the next popped node has a greater distance to the source; or

    b. The entire network has been searched .

At query time after the sub-network within the object range of the partition is retrieved, the same method may be applied to find KNN object nodes for a query, except that the dynamic retrieval of links and nodes not in the network is no longer required.

5. Experimental Implementations

An experimental implementation for metric data in 2D Euclidean Space on Cartesian Distance described in section 3 has been completed on Spark-based Databricks platform [7]. Source code and relevant Databricks notebooks and other resources may be found at:

https://github.com/OrdnanceSurvey/OR4NN

This implementation uses a regular grid for query space partition. Java Topology Suite (JTS) [8] is used to provide a local KNN search algorithm which is based on STR-Tree [9] and runs on a per quey basis. For this reason, we feel it is difficult to generate a fair direct performance comparison to other algorithms/packages. Nevertheless, this implementation has been used in more than 20 tasks of large scale KNN joins at Ordnance Survey (the British national mapping agency) where commercial packages either failed to generate exact results or unable to meet the performance requirements (on one occasion, our implementation took about 6 minutes to complete the task on top of about 4 minute re-useable pre-process time; in comparison, the commercial package we previously used took about 1.5 hours for the same task).

6. Summary and Discussion

In this research we present OR4NN as a conceptual framework to provide a strategy to tackle the problem of large scale KNN join in a shared-nothing distributed parallel computing environment such as Spark. Our initial motivation is to enable us to perform practical KNN join tasks on large scale 2D geospatial datasets before we realise our approach may be generalised to other types of data as well. Our experimental implementation and application to multiple real-world tasks demonstrates the feasibility of this approach. Nevertheless, there are several issues to be addressed/discussed.

Some of the issues are design/implementation decision related. As stated previously, object range generated for a given K will support KNN queries for $k \leq K$. If $k < K$, the object set retrieved by the object range will contain redundancy. In the era of "storage is cheap", it may be more query-time efficient if for each $k \leq K$ we generate a dedicated object range. If we decide to pre-select all objects intersecting the object range and store them along with the partition rather than retrieve them at query time, we may use a labelling mechanism to support redundancy-free object set at query time. For example, given a $K = 5$, we compute the object range for $k = 1$, retrieve objects intersecting this object range and label them with label 1; next we compute the object range for $k=2$ and label the new objects (i.e. excluding those retrieved at $k=1$) with label 2, and so on. At query time, for a $k_q = 2$, we will retrieve the objects with a label $\leq k_q$ (i.e. 1 and 2).

Another important issue worth detailed discussion is the allocation of the K candidate objects to a partition. The ideal scenario is that for each partition, there are K or slightly more objects intersecting the partition. If that is not the case, an indexing/search mechanism is required to discover more objects from adjacent partitions. As stated in section 2.4, this could potentially become a performance bottleneck for pre-process stage of an OR4NN implementation.

Data distribution is an important factor in candidate object allocation. For unevenly distributed dataset, regular partitioning scheme such as regular grids may not be the best option. Partitions such as Voronoi diagram generated from sampled objects may offer better performance.

An example of extreme data distribution is a task that we undertook, with all residential houses in the Great Britain as the query set and all “sea polygons” surrounding the Great Britain as the object set (sea polygons are the result of dividing the sea zone surrounding the GB coast into more manageable small polygons). Obviously, under a regular grid partition scheme, there are many partitions (in sea) containing no query; there is no object intersecting most of the land partitions, and for most land partitions, object allocation will require outward spiral search that covers great distance and many other (empty) partitions. This is the only task (among more than 20 others) that our regular grid based partition scheme struggles and we had to look for ad hoc solutions to get the work done. We are still uncertain what the best solution is for this kind of scenario and we believe it deserves more discussion and exploration.

It is an open question whether our method will perform well in high dimensions. It is well known that indexing and partitioning at high dimensions is difficult and inefficient. Our bold guess is that since a pre-process stage is normally less sensitive to performance, the workload offset by pre-process will result in performance gain at query time.

Another interesting aspect of potential OR4NN adoption/application is to offer a buffering mechanism to support continuous KNN queries of moving objects, similar to the description in [5] in a DBMS environment. When an object moves cross a partition, its real-time KNN queries may be answered by objects intersecting the object range of the partition. Consequently, frequent data loading/refreshing may be avoided.

ACKNOWLEDGMENTS

The authors would like to thank Dr. Chris Jochem and Dr. Kate New of Data Science and Analytics team at Ordnance Survey for applying and evaluating the experimental geospatial data implementation of OR4NN on real world datasets.

A patent application based on this research has been filed (UK Patent Office No. 2411135.3).

Appendix A

Claim (11) for object range of interior points on a line segment is a key step in the development of the method to construct object range for geometric data in 2D Euclidean space. Here we provide a detailed proof of the claim.

Without loss of generality, we use a line segment with endpoints (0,0) and (1,0) (Figure 10). Assuming for a give candidate object set, the object range for point (0,0) is a disc D_1 centered at (0,0) with radius r_1, and for point (1,0) a disc D_2 centered at (1, 0) with radius r_2. One of the intersection points of the boundaries of the two discs is at (a, b).

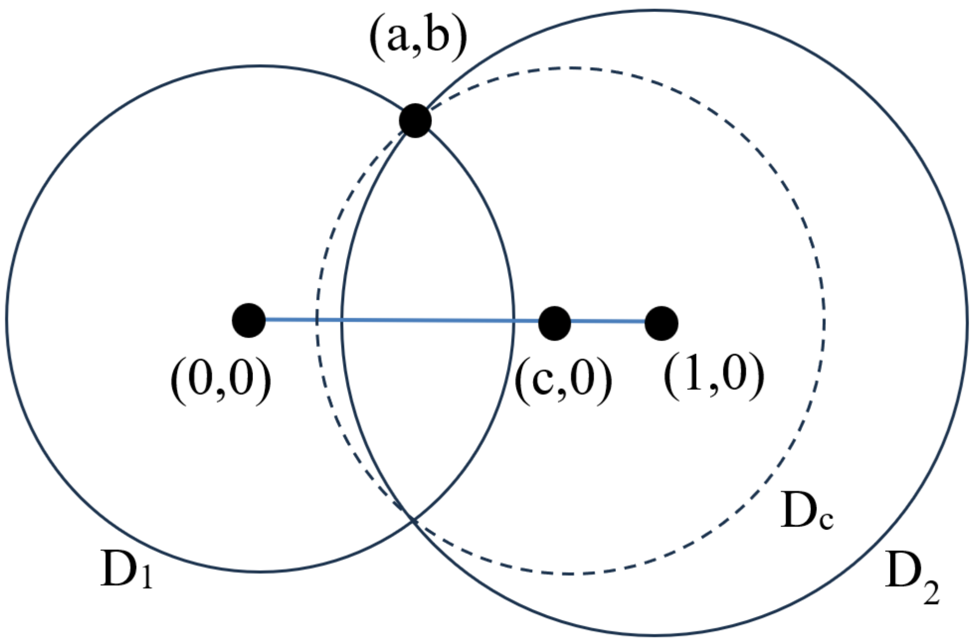

Figure 10: Object range of points on a line segment

Given an interior point (c, 0) on the line segment ($0 < c < 1$), we wish to prove that the disc D_c centered at (c, 0) with radius r_c = Dist((c, 0), (a, b)) satisfies:

- $D_c \subseteq \cup(D_1, D_2)$ and $\cap(D_1, D_2) \subseteq D_c$

D_1, D_2 and D_c are represented as:

$$x^2 + y^2 \leq r_1^2 = a^2 + b^2$$

$$(x-1)^2 + y^2 \leq r_2^2 = (a-1)^2 + b^2$$

$(x-c)^2 + y^2 \leq r_c^2 = (a-c)^2 + b^2$

We first prove $D_c \subseteq \cup(D_1, D_2)$. For $x \leq a$:

$(x-c)^2 + y^2 \leq (a-c)^2 + b^2 \quad \leftrightarrow$

$x^2 + y^2 \leq a^2+b^2-2ac+2cx \quad \leftrightarrow$

$x^2 + y^2 \leq a^2+b^2+2c(x-a) \leq a^2+b^2 = r_1^2$

That is, all points ($x\leq a$, y) in D_c are in D_0. For $x \geq a$:

$(x-c)^2 + y^2 \leq (a-c)^2 + b^2 \quad \leftrightarrow$

$x^2-2cx + c^2 + y^2 \leq a^2-2ac +c^2 + b^2 \quad \leftrightarrow$

$x^2-2x+1-2cx + c^2 + y^2 \leq a^2-2ac +c^2 + b^2 -2x +1 \quad \leftrightarrow$

$(x-1)^2 + y^2 \leq a^2-2a+1-2ac + b^2 +2cx- 2x + 2a \quad \leftrightarrow$

$(x-1)^2 + y^2 \leq (a-1)^2 + b^2 -2(1-c)(x-a) \leq (a-1)^2 + b^2 = r_2^2$

That is, all points ($x \geq a$, y) in D_c are in D_1. Therefore, $D_c \subseteq \cup(D_1, D_2)$.

Next we prove $\cap(D_1, D_2) \subseteq D_c$. For for $x \leq a$:

$(x-1)^2 + y^2 \leq r_2^2 = (a-1)^2 + b^2 \quad \leftrightarrow$

$x^2-2x+1+ y^2 \leq a^2-2a+1+ b^2 \quad \leftrightarrow$

$(x-c)^2-2x+1+ y^2 +2cx-c^2\leq a^2-2a+1+ b^2 \quad \leftrightarrow$

$(x-c)^2+ y^2 \leq a^2-2a + b^2+2x-2cx+c^2 \quad \leftrightarrow$

$(x-c)^2+ y^2 \leq (a-c)^2+ b^2+2(1-c)(x-a) \leq (a-c)^2+ b^2 = \quad r_c^2$

That is, all points ($x \leq a$, y) in D_2 are in D_c. For $x \geq a$:

$x^2 + y^2 \leq r_1^2 = a^2 + b^2 \leftrightarrow$

$(x-c)^2 + y^2 +2cx - c^2 \leq a^2 + b^2 \quad \leftrightarrow$

$(x-c)^2 + y^2 \leq a^2 + b^2-2cx+c^2 \quad \leftrightarrow$

$(x-c)^2 + y^2 \leq (a-c)^2 + b^2-2c(x-a) \leq (a-c)^2 + b^2 = r_c^2$

That is, all points ($x \geq a$, y) in D_1 are in D_c. Therefore, $\cap(D_1, D_2) \subseteq D_c$. This concludes the proof of claim (11).

Claim (13) may easily be derived from (11). In Figure 4, p is a point at the interior of the triangle. If we draw a line from vertex p_3 over p to cross edge p_1-p_2 where p_e is the intersection point, according to (11), $OR^E(p_e) \subseteq \cup(OR^E(p_1), OR^E(p_2))$. Since p is on the line segment p_3-p_e, according to (11) again we have $OR^E(p) \subseteq \cup(OR^E(p_3), OR^E(p_e)) \subseteq \cup(OR^E(p_1), OR^E(p_2), OR^E(p_3))$.

References

[1] Christian Bohm and Florian Krebs, 2004. The k-Nearest Neighbour Join: Turbo Charging the KDD Process. *Knowledge and Information Systems* 6 (6), 728–749. DOI: https://doi.org/10.1007/s10115-003-0122-9

[2] Nick Roussopoulos, Stephen Kelley and Frédéric Vincent, 1995. Nearest neighbor queries. *ACM SIGMOD Record* 24(2), 71-79. DOI: https://doi.org/10.1145/568271.2237

[3] Chi Zhang, Feifei Li and Jeffrey Jestes, 2012. Efficient parallel kNN joins for large data in MapReduce. In *Proceedings of the 15th International Conference on Extending Database Technology (EDBT'12)*. ACM Press, New York, NY, 38-49 DOI: https://doi.org/10.1145/2247596.2247602

[4] Dong Xie, Feifei Li, Bin Yao, Gefei Li, Liang Zhou and Minyi Guo, 2016. Simba: Efficient In-Memory Spatial Analytics. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD'16)*. ACM Press, New York, NY, 1075-1085. DOI: https://doi.org/10.1145/2882903.2915237

[5] Sheng Zhou and Jonathan Simmons, 2020. Data Pre-process Facilitating Efficient K-NN Queries in Spatial Database. *In Proceedings of Geographical Information Science Research UK (GISRUK) 2025*. https://easychair.org/publications/preprint/BKdm

[6] Francisco José García García, Antonio Corral, Luis Iribarne and Michael Vassilakopoulos, 2023. Efficient distributed algorithms for distance join queries in spark-based spatial analytics systems. *International Journal of General Systems*, 52(3), 206–250. DOI: https://doi.org/10.1080/03081079.2023.2173750

[7] Leila Etaati, 2019. Azure Databricks. In: *Machine Learning with Microsoft Technologies*. Apress, Berkeley, CA. DOI: https://doi.org/10.1007/978-1-4842-3658-1_10

[8] LocationTech. JTS Topology Suite. https://github.com/locationtech/jts. Accessed 2025-02-28.

[9] Scott Leutenegger, Jeffrey Edgington and Mario Lopez, 1997. STR: a simple and efficient algorithm for R-tree packing. In *Proceedings 13th International Conference on Data Engineering*. Birmingham, UK, pp. 497-506. DOI: https://doi.org/10.1109/ICDE.1997.582015.