Chapter 1
Nearest Neighbor Queries on Big Data

Georgios Chatzimilioudis and Andreas Konstantinidis and Demetrios Zeinalipour-Yazti

Abstract The proliferation of communication and positioning technologies in combination with the trend to share such information resulted in an explosion of geo-tagged temporal data. This opened an era of intelligent Big Data management for near real-time spatio-temporal applications as its sheer volume (i.e., millions of users world-wide) can not be processed in due time using conventional solutions. Followed by the emergence of location-aware social services, a new challenging *kNN* task arose, namely allowing mobile users to identify their *k* geographically closest neighboring nodes at all times. We analyze models and metrics that are the foundation of efficient solutions for near real-time processing Big Data in such spatio-temporal scenarios. In this book chapter, we study the problem of efficiently processing such a query in a cellular or WiFi network, both of which are ubiquitous. We present an algorithm, coined *Proximity*, which does not require any additional infrastructure or specialized hardware and its efficiency is mainly attributed to a smart *search space sharing* technique we present and analyze. Its implementation is based on a novel data structure, coined k^+-heap. *Proximity*, being parameter-free, performs efficiently in the face of high mobility and skewed distribution of users (e.g., the service works equally well in downtown, suburban, or rural areas).

Key words: k nearest neighbors; big data; distributed computing

1.1 Introduction

Big data refers to data sets whose size and structure strains the ability of commonly used relational DBMSs to capture, manage, and process the data within a toler-

Department of Computer Science, University of Cyprus
1 University Avenue, P.O. Box 20537, 1678 Nicosia, Cyprus
{gchatzim, akonstan, dzeina}@cs.ucy.ac.cy

able elapsed time [29]. The *volume-velocity-variety* of information in this kind of datasets give rise to the big data challenge, which is also known as the 3V challenge.

The *volume* of such datasets is in the order of few terabytes (TB) to petabytes (PB) that are often of high *information granularity*. Examples of such volumes are the U.S. Library of Congress that in April 2011 had more than 235 TB of data stored and the World of Warcraft online game using 1.3 PB of storage to maintain its game, the German Climate Computing Center (DKRZ) storing 60 PB of climate data. An example of high granularity is our Rayzit platform [44], where the level of detail goes as deep as a second in the temporal dimension and as a single location (longitude, latitude) in the space dimension per user.

The *velocity* of information in social media applications (such as photovoltaic, traffic and other monitoring apps) can grow exponentially as users join the community. Such growth can produce unprecedented volumes of data streams. For example, Ontario's Meter Data Management and Repository (MDM/R) [39] stores, processes and manages data from 4.6 million smart meters in Ontario, Canada and provides hourly billing quantity and extensive reports counting 110 million meter reads per day on an annual basis that exceeds the number of debit card transactions processed in Canada.

Furthermore, the *variety* of data can be anything from structured (relational or tabular) to semi-structured (XML or JSON) or even unstructured (Web text and log files) data and combination thereof. For example, Google's experimental robot cars [25], which have navigated thousands of miles of California roads, use an artificial-intelligence technique tackling big data challenges, parsing vast quantities of data and making decisions instantaneously.

Due to the high demand for big-data management, the literature witnessed an emergence of new techniques and tools for taming big data. For example, new data management related mechanisms are proposed [63, 36] such as Hadoop [26], i.e., a popular tool for analyzing data on racks of servers and NoSQL databases.

Furthermore, Computational Intelligence techniques [52, 47, 9] such as fuzzy logic, evolutionary computation, neurocomputing and other machine learning techniques provide us with complementary searching and reasoning means to bring forward solutions to the big data challenges. The computational intelligence techniques are traditionally used for *Knowledge Discovery in Databases (KDD)* since the evolution of soft computing, which according to Zadeh [61] is tolerant of imprecision, uncertainty, and partial truth. In [8], Brachman and Anand mentioned that KDD process requires several steps and different computational intelligence technologies [2] such as fuzzy models of the Takagi-Sugeno type for developing and understanding the application domain, the relevant prior knowledge, and identifying the goal of the KDD process. Neural networks [37], cluster analysis [1], decision trees [12], evolutionary computing [13] and neuro-fuzzy systems for data reduction and projection. Data mining related algorithm(s) for searching for patterns in the data include neuro-fuzzy methods [40], Genetic-Algorithms (GA) [30, 45], hybrid combination of GA and neural learning [3] and fuzzy clustering in combination with GA [48, 4].

As we enter the age of big data, many different evolutionary computation and machine learning techniques have been modified, combined, extended and inves-

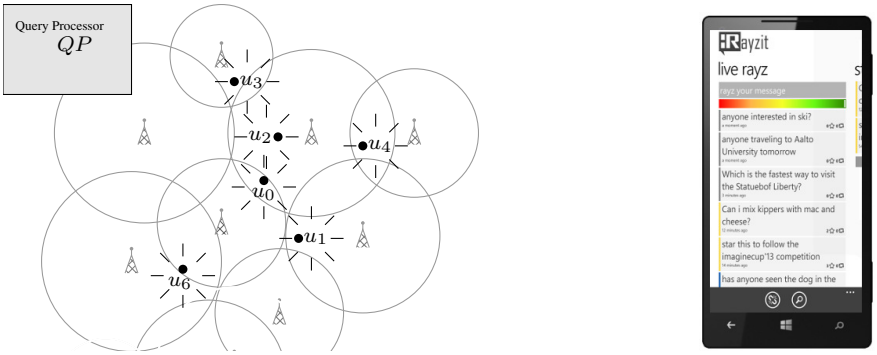
tigated for their ability to extract insights in an actionable manner. In [34], Stanford and Google researchers developed a deep learning based approach to build an online face detector by training a model on a large dataset of Youtube images (a model with 1 billion connections and a dataset with 10 million 200x200 pixel images downloaded from the Internet). The network is trained using model parallelism and asynchronous SGD on a cluster with 1,000 machines (16,000 cores) for three days. Furthermore, Hall. et al. [27] introduced a Decision tree approach for analyzing big data, which was extended in [42] by combining the Decision Trees with GAs for further improving their performance. Moreover in [35], Lu and Fahn proposed a hierarchical artificial neural network for recognizing high similar large data sets.

Evolutionary Computation (EC) techniques are mainly used for optimization tasks, such as finding the parameters/models that optimize some pre-specified evaluation criteria given some observed data and can be utilized for dealing with both single and Multi-objective Optimization Problems (MOPs) [20]. The former aims at finding a single solution that maximizes/minimizes an objective function in a single run and the latter aims at optimizing two or more often conflicting objectives simultaneously, in a single run. MOPs, however, cannot be time-sensitive and they are often tackled offline, since Multi-Objective EC techniques [20] (such as NSGA-II [21], MOEA/D [64], etc.,) combined with machine learning techniques [31] (e.g., Genetic Programming [32]) are slow and require many computational resources to obtain high quality solutions.

On the other hand, the wealth of new data accelerates advances in computing - a virtuous circle of big data. Machine-learning algorithms, for example, learn on data, and the more data, the more the machines learn. Consider Siri [50], the talking, question-answering application in iPhones, which Apple introduced in April 2010. Its origins go back to a Pentagon research project that was then turned to a Silicon Valley start-up. Apple bought Siri in 2010, and kept feeding it more data. Now, with people supplying millions of questions, Siri is becoming an increasingly adept personal assistant, offering reminders, weather reports, restaurant suggestions and answers to an expanding universe of questions.

The *k Nearest Neighbor (kNN)* search [46] is one of the simplest non-parametric machine learning approaches mainly used for classification [18] and regression [6]. kNN aims at finding k objects that are the most similar to another object. Extensions of kNN include the Condensed nearest neighbor (CNN, the Hart algorithm) algorithm that reduces the data set for kNN classification [28] and the fuzzy-kNN [49] that deals with uneven and dense training datasets. kNN finds applicabilities in several domains such as computational geometry [16], [23], [10], image processing [55], [33], spatial databases [62], [14], and recently in social networks [9].

The proliferation of smartphone devices and the emergence of location-aware social services emits a new challenging *kNN* task, namely allowing smartphones to identify their *k* geographically closest neighboring nodes at all times. We term this task *Continuous All-kNN (CAkNN)* queries. Applications of this neighborhood "sensing" capability could enhance public emergency services like E9-1-1 [22] and NG9-1-1 [41], and facilitate the uptake of location-based social networks (e.g., Rayzit [44], Waze [56]).



(a) (b)

Fig. 1.1 (a) A snapshot of a cellular network instance, where the 2-nearest neighbors for u_0 are $\{u_1, u_2\}$. Similarly for the other users: $u_1 \rightarrow \{u_0, u_2\}$, $u_2 \rightarrow \{u_3, u_0\}$, $u_3 \rightarrow \{u_2, u_0\}$, $u_4 \rightarrow \{u_2, u_3\}$, $u_6 \rightarrow \{u_0, u_1\}$. (b) Rayzit [44], an example application of a proximity-based micro-blogging chat.

Consider a set of smartphone users moving in the plane of a geographic region. Let such an area be covered by a set of *Network Connectivity Points* (*NCP*) (e.g., cellular towers of cellular networks, WiFi access points of wireless 802.11 networks etc.) Each *NCP* inherently creates the notion of a *cell*. Without loss of generality, let the cell be represented by a circular area[1] with an arbitrary radius. A mobile user u is serviced at any given time point by one *NCP*, but is also aware of the other *NCP*s in the vicinity whose communication range reach u (e.g., cell-ids of different providers in an area, or MAC addresses of WiFi hot-spots in an area.)

To illustrate our abstraction, consider the example network shown in Figure 1.1, where we provide a Rayzit micro-blogging chat channel between each user u and its $k = 2$ nearest neighbors. In the given scenario, each user concurrently requires a different answer-set to a globally executed query, as shown in the caption of Figure 1.1. Notice that the answer-set for each user u is not limited within its own *NCP* and that each *NCP* has its own communication range. Additionally, there might be areas with dense user population and others with sparse user population. Consequently, finding the k-nearest neighbors of some arbitrary user u could naively involve from a simple lookup in the *NCP* of u to a complex iterative deepening into neighboring *NCP*s, as we will show in Figure 3(b).

The remaining of this book chapter is as follows: Section 1.2 defines our system model and the problem. Section 1.3 provides the related work necessary for understanding the foundations of this work. Section 1.4 presents the *Proximity* framework and a breakdown of our data structures and algorithms. Section 1.5 finally summarizes and concludes the knowledge acquired from existing research and discusses our future plans.

[1] Using other geometric shapes (e.g., hexagons, Voronoi polygons, grid-rectangles, etc.) for space partitioning is outside the scope of this paper.

Table 1.1 Notation used throughout this work

Notation	Description
NCP	network connectivity point
c, C	single *NCP*, set of all *NCP*s
$radius_c$	range of *NCP* c
λ	the maximum number of users an *NCP* can serve
u, U	a single user, set of all users in the network
n	number of users in the network ($\|U\|$)
U_c	set of users of *NCP* c
r, R	a single user report, all user reports for a single timestep
$loc(u)$	location of user u
$ncp(u)$	the *NCP* that a user is registered to
$ncp_{vic}(u)$	list of *NCP*s whose range cover user u
S_c	the search space of *NCP* c
d_c	distance of k^{th} nearest user to the border of *NCP* c
kNN(u)	the set of k-nearest neighbors of user u
kth_c	the k^{th} nearest outside user to the boundary of cell c

1.2 System Model and Problem Formulation

This section formalizes our system model and defines the problem. The main symbols are summarized in Table 1.1.

Let U denote a set of smartphone users moving in the plane of a geographic region. Let such an area be covered by a set of *Network Connectivity Points* (*NCP*) (e.g., cellular towers found in cellular networks, WiFi access points found in wireless networks etc.) Each *NCP* inherently creates the notion of a *cell*, defined as c_i. Without loss of generality, let the cell be represented by a circular area with radius $radius_c$. The number of users λ serviced by an *NCP* is a network parameter (cell capacity). A mobile user u is serviced at any given time point by one *NCP*, but is also aware of the other *NCP*s that are in its vicinity and whose communication range cover it (e.g., cell-ids of different providers in an area, or MAC addresses of WiFi hot-spots in an area, etc.)

Assume that there is some centralized (or cloud-like) service, denoted as QP (Query Processor), which is accessible by all users in user set U. Allow each user u to report its positional information to QP regularly. These updates have the form $r_u = \{u, loc(u), ncp(u), ncp_{vic}(u)\}$, where $loc(u)$ is the location of user u[2], $ncp(u)$ is the *NCP* user u is registered to and $ncp_{vic}(u)$ is a list of *NCP*s in the vicinity of u.

The problem we consider in this work is how to efficiently compute the k-nearest neighbors of all smartphones that are connected to the network, at all times. We consider a *timestep* that defines rounds where we need to recompute the *kNN*s of

[2] The location of a user can be determined either by fine-grain means (e.g., AGPS) or by coarse-grain means (e.g., fingerprint-based geo-location [24]).

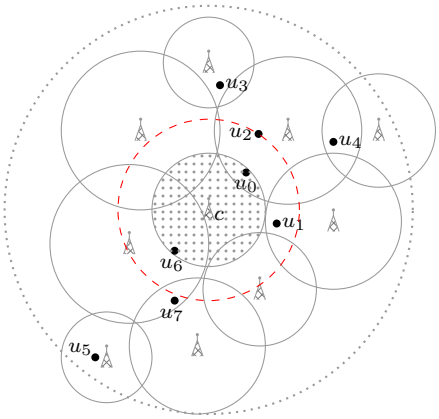


Fig. 1.2 The *search space* of cell c is the big circle with the dotted outline. Any user inside this circle is a *kNN* candidate for any user inside c.

the users. Depending on the application, this can take place either at a preset time interval or whenever we have a number of new user location updates arriving at the server. Formally, we aim to solve a problem we coin the *CAkNN* problem.

Definition 1 (*CAkNN* problem). Given a set U of n points in space and their location reports $r_{i,t} \in R$ at *timestep* $t \in T$, then for each object $u_i \in U$ and *timestep* $t \in T$, the *CAkNN* problem is to find the k objects $U_{sol} \subseteq U - u_i$ such that for all other objects $u_o \in U - U_{sol} - u_i$, $dist(u_k, u_i) \leq dist(u_o, u_i)$ holds.

In order to better illustrate our definition, consider Figure 1.2, where we plot a *timestep* snapshot of 7 users $u_0 - u_6$ moving in an arbitrary geographic region. The result for this *timestep* to a $k = 2$ query would be $kNN(u_0) = \{u_1, u_2\}$, $kNN(u_1) = \{u_0, u_2\}$, $kNN(u_2) = \{u_3, u_0\}$, $kNN(u_3) = \{u_2, u_0\}$, $kNN(u_4) = \{u_2, u_0\}$, $kNN(u_6) = \{u_7, u_1\}$.

Obviously, the solution for a user u will not always reside inside the same NCP cell c, but might reside in neighboring cells or even further (e.g., if neighboring cells do not have any users). Computing a separate search space for every user is very expensive. On the other hand, *search space sharing* is achieved when the same search space is used by multiple users and it guarantees the correct *kNN* solution for all of them. If we apply this reasoning for all users U_c in c, then the common search space S_c for U_c would be defined as the union of the individual search spaces of every user in U_c. We efficiently build S_c with the assistance of complementary data structures we devise in this work and explain next. In Figure 1.2, the search space constructed by our framework for users u_0 and u_6 is the largest dotted circle.

1.3 Related Work

In this section we provide an extensive coverage of the k nearest neighbor related work tackling static data, continuous data and distributed data.

1.3.1 kNN Queries on Static Data

For applications where data is represented by a linear array, constant time algorithms have been proposed to solve the *All Nearest Neighbor* (*ANN*) and *All k-Nearest Neighbor* (*AkNN*) problems. There has been extensive work in the field of image processing and computational geometry (e.g., [55, 33]).

In Euclidean space (and general metric spaces), there has been also extensive work on solving the *ANN* and *AkNN* problems. For large datasets residing on disk (external memory), works like Zhang *et al.* [62] and Chen *et al.* [14] exploit possible indices on the datasets and propose algorithms for R-tree based nearest neighbor search.

For small ANN and AkNN problems in Euclidean space, where data fits inside main memory, early work in the domain of computational geometry has proposed solutions. Clarkson et al. [16] was the first to solve the ANN problem followed by Gabow et al. [23], Vaidya [54] and Callahan [10]. Given a set of points, [16, 23] use a special quad-tree and [54] use a hierarchy of boxes to divide the data and compute the ANN. The worst case running time, for both building the needed data structures and searching in these techniques, is $O(nlogn)$, where n is the number of points in the system. For the AkNN problem works [16] propose an algorithm with $O(kn + nlogn)$ and [54] an algorithm with $O(knlogn)$ time complexity.

For multi-dimensional disk-resident data kNN Joins have been optimized in [9, 57, 60, 58, 36]. Works [9]-[58] propose centralized solutions. [62, 14, 9, 57, 60, 58, 36, 63]. Zhang et al. [62] index the disk-resident data using an R-tree and develop an efficient depth-first traversal algorithm and a hash based algorithm. In [14] the authors propose the minimum bounding rectangle enhanced quadtree as well as a new distance measure for pruning as many distance comparisons as possible. In [9], a specialized index together with an optimal page loading strategy were proposed to reduce both CPU and I/O cost for disk-resident data. Xia et al. [57] optimizes kNN Joins for high dimensional data by hashing points into blocks and sorting the blocks for a nested loop join. [60, 58] similarly propose efficient indexing techniques for avoiding scanning the whole dataset repeatedly and for pruning as many distance computations as possible.

1.3.2 Continuous kNN Queries

When it comes to streaming updates of object attributes, main memory processing is usually mandatory for spatio-temporal applications, where objects are highly mobile.

Yu et al. [59] followed by Mouratidis *et al.* [19] optimize Continuous kNN queries. Objects are indexed by a grid in main memory given a system-defined parameter value for the grid size. For each query they both use a form of iteratively enlarging a range search to find the kNN. For small object speeds and/or low object agility, both propose a *stateful* technique to incrementally compute the result of a query of the current timestep using the result of the previous timestep. They define an influence region for the query inside the grid and depending on what happens in this region, the new result is computed using the previous result, minimizing the search space. Whenever the query object moves or the agility and speed of the objects is high, both fall back to their slower *stateless* version where at each timestep the result of the query is computed from scratch.

Chatzimilioudis *et al.* [11] is the only work that optimizes Continuous All-kNN queries, termed Continuous-AkNN, in a centralized environment. The core intuition behind this work is geographically partitioning the object space based on the transmission radius of the network connectivity points (e.g., WiFi router, cellular base-station, etc.) and determining a candidate set for each network connectivity point. Then, each object v only scans the candidate set of its connectivity point and determines its kNN.

1.3.3 All-kNN Queries in Distributed Systems

The high information granularity of the location updates is very restrictive for disk-based storage and indexing. Therefore, distributed and parallel techniques for memory-resident data is desirable and demands optimization in respect to the CPU time.

Callahan's [10] main contribution is a parallel algorithm that solves the AkNN problem in $O(logn)$ using $O(n)$ processors with *shared-memory*. Given a set of points, Callahan use a special quad-tree to divide the data. His algorithm has $O(kn+nlogn)$ time complexity.

Recently, solutions utilizing large-scale distributed data processing have been proposed by Lu et al. [36] optimized for exact kNN Joins and Zhang et al. [63] optimized for approximate kNN Joins.

Zhang et al. [63] give a MapReduce technique that optimizes AkNN queries. It splits A into equally-sized $\sqrt{m}$ disjoint subsets, i.e., $A = \bigcup_{1 \leq l \leq \sqrt{m}} A_l$, creating m distinct combinations of 2 subsets $\{A_l, A_h\}$. In a first parallel phase, each server is assigned one such pair of subsets and scans A_h to find the kNNs for each object in A_l. All intermediate answers are then collected and distributed in a second parallel phase to compute the top-k neighbors for each object in A.

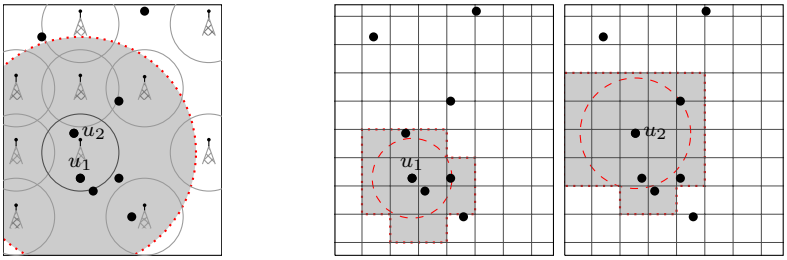


Fig. 1.3 (a) In *Proximity* the search space is pre-constructed for all users of the same cell (e.g., u_1 and u_2); whereas (b) for existing state-of-the-art algorithms the search space needs to be iteratively discovered by expanding a ring search for each user separately into neighboring cells.

Lu et al [36] propose a 2-job Map-Reduce solution for optimizing AkNN queries on static datasets. In a centralized pre-processing step a set of optimal pivot points are carefully selected. In the first Map-Reduce job, the mappers split set A into m disjoint subsets A_i based on the Voronoi cells generated by the selected pivots, and record the maximum and minimum distance between each pivot and its subset. Each area A_i and corresponding set O_i is mapped to a server s_i. There is no reduce step in their first Map-Reduce job. In the second Map-Reduce job, for each A_i a candidate set $CS_i \subseteq A$ is computed that guarantees to include all kNNs for the subset A_i. Each A_i is mapped to its candidate set CS_i. The reducer s_i then computes the kNNs for each point $o \in O_i$ using CS_i.

1.3.4 Shortcomings Of Existing Work

Techniques that optimize disk I/O are unattractive for solving *CAkNN* queries, since the CPU latency is the actual bottleneck as shown by Chen *et al.* [14]. Moreover, tree-based techniques proposed for *ANN* queries require super-linear time for their structure build-up phase (as [16, 23, 54, 10]) and need to be updated or re-built in every timestep, which is inefficient.

No previous work tackles the problem of continuous all k-nearest neighbor (*CAkNN*) queries specifically. In smartphone network applications the users are highly mobile with hard-to-predict mobility patterns and their location distribution is far from uniform [43]. This makes *stateful* techniques inefficient as shown in [59, 19], since keeping previous answers (states) of the query becomes more of a burden than a help for faster query evaluation. Furthermore, in proximity applications considered in this paper, smartphone users are moving and are both the objects of interest and the focal points of queries.

Our framework, *Proximity*, is main-memory based and *stateless*, i.e., no previous data/calculation of the previous evaluation round is used in the current round. A *stateless CAkNN* solution would solve an *AkNN* problem at each timestep. In [11],

we compare *Proximity* analytically to the early work of computational geometry [16, 23, 54, 10] and show that the running time complexity of our framework is better (i.e., $O(n(k+\lambda))$ as opposed to $O(knlogn)$). We compare *Proximity* experimentally against an adaptation of state-of-the-art *CkNN* solution [59, 19]. Due to the agility of the realistic mobile datasets used, these works can only make use of their *stateless* algorithm, which solves a *kNN* query in every timestep. Thus, such adaptations can only optimize a *kNN* query for each timestep separately and for each user separately, building a new search space for each user. We show that our specialized *Proximity* framework performs better, mainly due the batch processing capability of the *AkNN* queries. The most significant difference is that the *Proximity* framework groups users of the same cell together and uses the same search space for each group (*search space sharing*).

1.4 The *Proximity* Framework

We start out outlining of the *Proximity* framework and the intuition behind its operation. We then describe in detail how the search space is built-up using our k^+-heap data structure and its associated insertion and update algorithms.

1.4.1 Outline Of Operation

The *Proximity* framework is designed in such a way that it is: i) *Stateless*, in order to cope with transient user populations and high mobility patterns, which complicate the retrieval of the continuous *kNN* answer-set. In particular, we solve the *CAkNN* problem for every timestep separately without using any previous computation or data; ii) *Parameter-free*, in order to be invariant to parameters that are network-specific (such as cell size, capacity, etc.) and specific to the user-distribution; iii) *Memory-resident*, since the dynamic nature of mobile user makes disk resident processing prohibitive; iv) *Specially designed* for *highly mobile* and *skewed distribution* environments performing equally well in downtown, suburban, or rural areas; iv) *Fast and scalable*, in order to allow massive deployment; and v) *Infrastructure-ready* since it does not require any additional infrastructure or specialized hardware.

For every timestep *Proximity* works in two phases (Algorithm 1): In the first phase one k^+-heap data structure is constructed per *NCP*, using the location reports of the users (lines 1-8). In the second phase, the k-nearest neighbors for each user are determined by scanning the respective k^+-heap and the results are reported back to the users (lines 9-19).

At each timestep the server QP initializes our k^+-heap for every *NCP* in the network. The k^+-heap integrates three individual sub-structures that we will explain next. The user location reports are gathered and inserted into the k^+-heap of every *NCP*. After all location reports have been received and inserted, each *NCP* has its

Algorithm 1 . Proximity Outline

Input: User Reports R (single timestep), set C of all *NCP*s
Output: *kNN* answer-set for each user in U

```
1: for all c ∈ C do
2:     initialize k+_c                          ▷ Initialize our k+-heap
3: end for
4: for all r ∈ R do                             ▷ Phase 1: build k+-heap
5:     for all c ∈ C do
6:         insert(r, k+_c)
7:     end for
8: end for
9: U ← users(R)
10: for all u ∈ U do                            ▷ Phase 2: scan k+-heap
11:     kNN_u = ∅                               ▷ Conventional k-max heap
12:     c ← r_u.ncp
13:     for all v ∈ k+_c do
14:         if v is a kNN of u then
15:             update(kNN_u, v)
16:         end if
17:     end for
18:     report kNN to node u
19: end for
```

search space stored inside its associated k^+-heap. After the build phase, each user scans the k^+-heap of its *NCP* to find its k-nearest neighbors.

1.4.2 Constructing The Search Space

Here we describe the intuition behind our search space sharing concept. Every user covered by an *NCP* uses the same search space to identify its *kNN* answer-set.

In order to construct a correct search space for each *NCP*, we need to be able to identify nodes that might be part of the *kNN* answer-set for any arbitrary user of a given *NCP*. For instance, consider two users u_0 and u_6, in Figure 1.2, which are positioned on the perimeter of their *NCP* c. Also, consider user u_2 being outside c and close to u_0. In such a scenario, the search space for c must obviously include u_2, as it is a better *kNN* candidate to u_0 than u_6. However, even if we were aware of the k closest users to c (besides the users in c), would not allow us to correctly determine the *kNN* for any arbitrary user in c. To understand this, consider again Figure 1.2 with a *2NN* query. u_1 and u_2 are the two closest outside nodes to the border of c. Yet, we can visually determine that u_7 is a more appropriate *2NN* candidate for u_6 than all aforementioned nodes, i.e., u_0, u_1, u_2.

To overcome this limitation, we define a prune-off threshold, denoted as kth_c, which determines the size of the search space of c. kth_c is the k^{th} closest outside user to the border of c, which determines the width d_c of the *search expansion* (striped ring as seen in Figure 1.4). Inside this ring there are k users by definition.

These k users form the K-set. In our running example $kth_c = u_2$. This guarantees that the search space will have at least k users. All users at distance less that $2 * radius_c + d_c$ from c's border, are also part of the search space. This guarantees that each user inside c will find its actual kNN inside the search space.

The size of each *NCP* search space depends on the communication area of the *NCP* and the k^{th} closest outside user to the border of its communication area. The users inside c comprise set U_c and the users that are at distance greater than d_c and less than $2*radius_c+d_c$ from the cell's border comprise set B_c (grey ring in Figure 1.4). Set K, set B, and the users U_c inside c form the search space S_c of c.

Definition 2 (K-set). Given a set of users $u \in U - U_c$ outside *NCP* cell c that is ordered with ascending distance $dist(u, c)$ to the border of c, set K_c consists of the first k elements of this set (striped ring in Figure 1.4).

Definition 3 (k^{th} outside neighbor of the *NCP* cell). Given K_c (ordered as in Definition 2), the k^{th} user is called the k^{th} nearest neighbor of c and denoted kth_c.

Definition 4 (B, Boundary Set). Given an *NCP* denoted as c and its k^{th} outside neighbor kth_c, set B_c consists of all users $u \in U - (U_c \cup K)$ with distance $dist(u, c) < dist(kth_c, c) + 2 * radius_c$ from the border of c. In other words B_c consists of all users $u \in U$ with distance $dist(kth_c, c) < dist(u, c) < dist(kth_c, c) + 2 * radius_c$.

Definition 5 (S, Search Space set). Given an *NCP* c and its K_c set, the search space S_c of c consists of all users $u \in U_c \cup K_c \cup B_c$ (big circle with dotted outline in Figure 1.4).

In our Figure 1.4 example, at the end of the build phase, the k^+-heap of c includes users $\{u_6, u_0, u_1, u_2, u_3, u_4, u_5\}$. This is the common search space S_c for all users $U_c = \{u_0, u_6\}$ of c, which guarantees to include their exact k-nearest neighbors.

1.4.3 Specialized Heap: The k^+-heap

Computing the search space for each cell inefficiently might be prohibitive for the application scenarios we envision as detailed in the introduction. In this section, we show in detail how the search space for an *NCP* is constructed using our k^+-heap data structure. Recall that as user reports arrive at the server QP they are inserted into each k^+-heap. A user report either stays inside a k^+-heap or eventually gets evicted using a policy that we will describe later. After all user reports have been

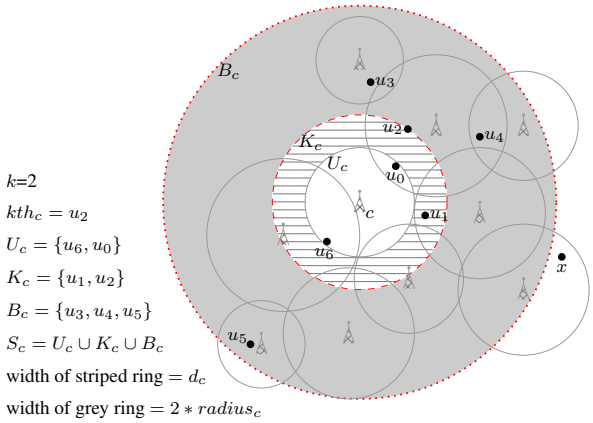


Fig. 1.4 An example of the common search space for the users inside cell c (white circle) for $k = 2$. The search space S_c of c is $\{u_0, u_1, u_2, u_3, u_4, u_5, u_6\}$ and is represented by the big circle with the dotted outline. Set S_c includes all users inside c (set U_c), the striped ring (set K_c) and the grey ring (set B_c). Any node outside S_c (e.g., user x) is guaranteed NOT to be a *kNN* of any user inside cell c. The 2-nearest neighbors for the nodes in c are $kNN(u_0) = \{u_1, u_2\}$ and $kNN(u_6) = \{u_0, u_1\}$.

probed through the k^+-heap of every *NCP*, each k^+-heap contains the actual search space of its *NCP*. Consequently, the build phase takes a total of $n * |C|$ insertions.

The k^+-heap consists of three separate data structures: a heap for the set K_c and two lists for *Boundary* set B_c and the set U_c. The heap used for set K_c is a conventional k-max-heap. It stores only the k users outside c with the minimum distance $dist(u, c)$ from the border of c. Thus, the heap K has always kth_c at its head. The *boundary* list is a list ordered by $dist(u, c)$, which stores set B. Its elements are defined by kth_c (see Definition 4). Similarly, we use a list to store the users $U_c \subseteq U$ of c. Notice that some *NCP* cells will be overlapping, so there are areas where users are inside multiple cells. Such users are inserted into all lists U_j of $c_j \in C$ that cover them. The k^+-heap has $O(1)$ lookup time for the k^{th} nearest neighbor of c. It has worst case $O(log(k * |B|))$ insertion time and contains $|S_c| = k + |B_c| + |U_c|$ elements.

1.4.4 Insertion Into The k$^+$-Heap (Algorithm 2 and 3)

When inserting a new element u_{new} into the k^+-heap of c, we distinguish among four cases (see Algorithm 2): i) u_{new} is covered by c and belongs to set U_c (line 2), ii) u_{new} belongs to set K_c (line 4), iii) u_{new} belongs to set B_c (line 11), or iv) u_{new} does not belong to the search space $S_c = U_c \cup K_c \cup B_c$ of *NCP* c (line 13). In case (i) the element is inserted into the U_c list. In case (ii) we need to insert u_{new} into heap K (line 5) and move the current head kth_c from K to the boundary list B

Algorithm 2 . k^+-heap: Insert(u_{new})

Input: u_{new}, c of u_{new}
Output: ${k^+}_c$
1: $kth_c \leftarrow head(K_c)$
2: **if** $dist(u_{new}, c) < radius_c$ **then**
3: $\quad insert(u_{new}, U_c)$
4: **else if** $dist(u_{new}, c) < dist(kth_c, c)$ **then**
5: $\quad insert(u_{new}, K_c)$
6: $\quad$ **if** K heap has more than k elements **then**
7: $\quad\quad kth_c \leftarrow pophead(K_c)$
8: $\quad\quad insert(kth_c, B_c)$
9: $\quad\quad Update\_boundary(head(K_c))$
10: $\quad$ **end if**
11: **else if** $dist(u_{new}, c) < dist(kth_c, c) + 2 * radius_c$ **then**
12: $\quad insert(u_{new}, B_c)$
13: **else**
14: $\quad$ discard u_{new}
15: **end if**

Algorithm 3 . k^+-heap: Update_boundary(kth_c)

Input: kth_c (the k^{th} outside neighbor of *NCP* c)
Output: B_c updated
1: $d \leftarrow dist(kth_c, c) + 2 * radius_c$
2: $i \leftarrow$ element with the max. distance smaller than d using binary search
3: $remove(B_c, i + 1, end)$

(lines 7-8). This yields a new head kth'_c in K (line 9). Every time the kth_c changes, the boundary list B needs to be updated, since it might need to evict some elements according to Definition 4. In case (iii) we insert u_{new} into the ordered boundary list B (line 12). Note that the sets K_c and B_c are formed as elements are inserted into the k^+-heap. The first k elements inserted in the empty k^+-heap define the K_c set. In case (iv) the element is discarded.

1.4.5 Running Example

Using Figure 1.4 as our network example in timestep t we will next present the *Proximity* framework step-by-step.

Server QP initiates a k^+-heap for every *NCP* in C. The k^+-heap consists of heap K, ordered list B, and list U. The reports that arrive at QP are $R = r_0, r_1, r_2, r_3, r_4, r_5, r_6, r_x$. Every report is inserted into every k^+-heap on the QP (see Algorithm 1, lines 1-5). The order in which the reports are inserted into a k^+-heap does not affect the correctness of the search space.

For our example, assume that the reports are inserted in the order seen in the first column of Table 1.2. For every insertion we can see the contents of ${k^+}_c$ in the same

Table 1.2 Build-up phase of the k^+-heap of *NCP* c as user location reports are inserted.

Arriving Reports	**Structure K_c**	**Structure B_c**	**Structure U_c**	**line in Algorithm 2**
r_4	$\{r_4\}$	$\{\}$	$\{\}$	1,4,5
r_x	$\{r_x, r_4\}$	$\{\}$	$\{\}$	1,4,5
r_2	$\{r_4, r_2\}$	$\{r_x\}$	$\{\}$	1,4-11
r_3	$\{r_3, r_2\}$	$\{r_4, r_x\}$	$\{\}$	1,4-11
r_1	$\{r_2, r_1\}$	$\{r_3, r_4\}$	$\{\}$	1,4-11
r_5	$\{r_2, r_1\}$	$\{r_3, r_4, r_5\}$	$\{\}$	1,12,13
r_6	$\{r_2, r_1\}$	$\{r_3, r_4, r_5\}$	$\{r_6\}$	1-3
r_0	$\{r_2, r_1\}$	$\{r_3, r_4, r_5\}$	$\{r_6, r_0\}$	1-3

Table. For simplicity we will only follow the operation for the k^+-heap of *NCP* c. When report r_4 is inserted into ${k^+}_c$ it ends up inside heap K_c, since user u_4 is not inside *NCP* cell c (condition line 2) and heap K_c is empty. Next, report r_x is inserted into ${k^+}_c$ and it also ends up inside heap K_c since this is not full yet. When r_2 is inserted, it ends up inside heap K_c (line 5) and it becomes the new head of the heap kth_c. The old head of the heap was r_x and is popped out of K and is inserted into the B_c list (lines 7-8). The update on the B_c list is triggered (line 9) which, in this case, does not affect the list. Similarly, when r_3 is inserted the same operations (lines 5-10) take place as with the insertion of r_2. Next, r_1 is inserted with the same effect, only this time the B_c list is altered during its update (line 9). r_2 is the new head of heap K_c and according to Definition 5 defines a new search space radius $d = dist(u_2, c) + 2 * radius_c$ (line 1 of Algorithm 3). The report r_x inside list B_c has $dist(u_x, c) > d$, thus it belongs to the tail of the list that is discarded in line 3 of Algorithm 3. When r_5 is inserted it ends up directly inside list B_c (line 12), since it is outside c, further away than kth_c but closer than $dist(kth_c, c) + 2 * radius_c$ to the border of c. Reports r_0 and r_6 both end up directly inside list U_c (line 3), since they are covered by c, satisfying the condition in line 2.

After all reports are inserted into the k^+-heaps phase 1 of Algorithm 1 is completed and the search space is ready. For the second phase of Algorithm 1 the server scans a single k^+-heap for each user. The server can scan the k^+-heap of any *NCP* that covers a user u to get the k neighbors of u. In our Algorithm 1 the server scans the *NCP* that actually services the user $ncp(u)$ (lines 12). For users u_0 and u_6, the server Q scans ${k^+}_c = u_2, u_1, u_3, u_4, u_5, u_6$ and finds nearest neighbors $\{u_2, u_1\}$ and $\{u_0, u_1\}$ for user u_0 and u_6 respectively.

1.5 Summary and Future Work

We have presented *Proximity*, an algorithm for continuously answering all k-nearest neighbor queries in a cellular network. It is based on a division of the search space based on the network connectivity points and exploiting search space sharing among

users of the same connectivity point. The *Proximity* framework has a better time complexity compared to solutions based on existing work. Our experiments verify the theoretical efficiency and shows that *Proximity* is very well suited for large scale scenarios.

Proximity is easily adaptable to provide some location privacy in terms of *spatial cloaking*. It is naturally extendible to the scenario where users report to the server using *spatial cloaking* [15]. The location of a user is represented by an area, instead of a simple point. We will further investigate privacy extensions for our algorithms aiming towards strong privacy in future work. Existing work towards this direction has been published by Papadopoulos et al [51], where they answer a single snapshot of a *kNN* query guaranteeing strong privacy.

Extensions and future plans for this work are also placed in parallelizing the *Proximity* algorithm, specializing it for cloud environments and adapt our search space sharing for user-defined k values, instead of a system-defined global k value. This allows for scalability in the face of the Big Data era. Lastly, kNN on Big Data can benefit further by combining it with Computer Intelligence techniques, such as neural networks.

References

1. S. Abe, R. Thawonmas, Y. Kobayashi, "Feature selection by analyzing regions approximated by ellipsoids," *IEEE Trans. on Systems, Man, and Cybernetics, Part. C*, vol. 28, no. 2, pp. 282-287, 1998.
2. J. Abonyi, B. Feil, A. Abraham, "Computational Intelligence in Data Mining," *Informatica*, vol. 29, pp. 3–12, 2005.
3. A. Abraham, "Evonf: A framework for optimization of fuzzy inference systems using neural network learning and evolutionary computation," In *IEEE International Symposium on Intelligent Control*, ISIC02, IEEE Press, ISBN 0780376218, pp. 327-332, Canada, 2002.
4. A. Abraham, "i-miner: A web usage mining framework using hierarchical intelligent systems," In *The IEEE International Conference on Fuzzy Systems* FUZZ-IEEE03, IEEE Press, ISBN 0780378113, pp. 1129–1134, USA, 2003.
5. F. N. Afrati, A. D. Sarma, S. Salihoglu, and J. D. Ullman, "Upper and lower bounds on the cost of a map-reduce computation," in *Proceedings of the 39th international conference on Very Large Data Bases*, ser. PVLDB'13. VLDB Endowment, 2013, pp. 277- 288.
6. N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," In *The American Statistician*, vol. 46, no. 3, pp. 175-185, 1992.
7. N. Armenatzoglou, S. Papadopoulos, and D. Papadias, "A general framework for geo-social query processing," *Proc. VLDB Endow.*, vol. 6, no. 10, August 2013.
8. R. Brachman, T. Anand, "The process of knowledge discovery in databases," *Advances in Knowledge Discovery and Data Mining*, AAAI/MIT Press, pp. 37-58, 1994.
9. C. Boehm and F. Krebs, "The k-nearest neighbour join: Turbo charging the kdd process," *Knowledge Information Systems*, vol. 6, no. 6, pp. 728–749, Nov. 2004.
10. P. B. Callahan, "Optimal parallel all-nearest-neighbors using the well-separated pair decomposition," in *Proceedings of the 1993 IEEE 34th Annual Foundations of Computer Science*. Washington, DC, USA: IEEE Computer Society, 1993, pp. 332–340.
11. G. Chatzimilioudis, D. Zeinalipour-Yazti, W.-C. Lee, and M. D. Dikaiakos, "Continuous all k-nearest neighbor querying in smartphone networks," in *13th International Conference on Mobile Data Management (MDM'12)*, 2012.

12. A. A. S. Chebrolu, J. Thomas, "Feature deduction and ensemble design of intrusion detection systems," *Computers and Security*, Elsevier, 2004.
13. Y. Chen, B. Yang, J. Dong, A. Abraham, "Time-series forecasting using flexible neural tree model," *Information Sciences*, vol. 174, no. 3–4, pp. 219–235, 2005.
14. Y. Chen and J. Patel, "Efficient evaluation of all-nearest-neighbor queries," in *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, Apr. 2007, pp. 1056 –1065.
15. R. Cheng, Y. Zhang, E. Bertino, and S. Prabhakar, "Preserving user location privacy in mobile data management infrastructures," in *Privacy Enhancing Technologies*, ser. Lecture Notes in Computer Science, G. Danezis and P. Golle, Eds., vol. 4258. Springer, 2006, pp. 393–412.
16. K. L. Clarkson, "Fast algorithms for the all nearest neighbors problem," *Foundations of Computer Science, Annual IEEE Symposium on*, vol. 83, pp. 226–232, 1983.
17. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms (3rd ed.)*. The MIT press, 2009.
18. T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions of Information Theory*, vol. 13, no. 1, pp. 21–27, Sep. 2006.
19. K. Mouratidis, D. Papadias, and M. Hadjieleftheriou, "Conceptual partitioning: an efficient method for continuous nearest neighbor monitoring," in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '05. New York, NY, USA: ACM, 2005, pp. 634–645.
20. K. Deb, "Multi-Objective Optimization Using Evolutionary Algorithms," *Wiley and Sons*, 2002.
21. K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA II," *IEEE TEC*, 2002.
22. (2014, Jan.) Federal Communications Commission - Enhanced 911 website. [Online]. Available: `http://www.fcc.gov/pshs/services/911-services/enhanced911/`
23. H. N. Gabow, J. L. Bentley, and R. E. Tarjan, "Scaling and related techniques for geometry problems," in *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, ser. STOC '84. New York, NY, USA: ACM, 1984, pp. 135–143.
24. (2014, Jan.) Geolocation API website. [Online]. Available: `http://code.google.com/apis/gears/api\_geolocation.html`
25. (2014, Jan.) Popular Science: Inside Google's Quest To Popularize Self-Driving Cars article. [Online]. Available: `http://www.popsci.com/cars/article/2013-09/google-self-driving-car`
26. (2014, Jan.) Hadoop website. [Online]. Available: `http://hadoop.apache.org/`
27. L. 0. Hall, N. Chawla , K. W. Bowyer, "Decision Tree Learning on Very Large Data Sets," In *IEEE International Conference on System, Man and Cybernetics (SMC)* , pp. 187–222, Oct 1998.
28. P. E. Hart, "The Condensed Nearest Neighbor Rule" In *IEEE Transactions on Information Theory*, no. 18, pp. 515-516, 1968.
29. J. Hoffer, V. Ramesh, H. Topi, Modern Database Management, 2013.
30. H. Ishibuchi, T. Nakashima, T. Murata, "Performance evaluation of fuzzy classifier systems for multidimensional pattern classification problems," *IEEE Trans. SMCB*, vol. 29, pp. 601-618, 1999.
31. A. Konstantinidis, H. Haralambous, A. Agapitos, H. Papadopoulos, "A GP-MOEA/D Approach for Modelling Total Electron Content over Cyprus," *CoRR*, 2011.
32. J.R. Koza, "Genetic Programming: On the Programming of Computers by Means of Natural Selection," *MIT Press*, 1992.
33. T. H. Lai and M.-J. Sheng, "Constructing euclidean minimum spanning trees and all nearest neighbors on reconfigurable meshes," *IEEE Transactions of Parallel Distributed Systems*, vol. 7, no. 8, pp. 806–817, Aug. 1996.
34. Q. Le, M. Ranzato, R. Monga, M. Devin, K. Chen, G. Corrado, J. Dean, A. Ng, "Building high-level features using large scale unsupervised learning," In *International Conference in Machine Learning*, 2012.

35. Y.-L. Lu and C.-S. Fahn, "Hierarchical Artificial Neural Networks for Recognizing High Similar Large Data Sets," In *International Conference on Machine Learning and Cybernetics*, vol. 7, pp. 1930–1935, 2007.
36. W. Lu, Y. Shen, S. Chen, and B. C. Ooi, "Efficient processing of k nearest neighbor joins using mapreduce," *Proc. VLDB Endow.*, vol. 5, no. 10, pp. 1016–1027, Jun. 2012.
37. J. Mao, K. Jain, "Artificial neural networks for feature extraction and multivariate data projection," *IEEE Trans. on Neural Networks*, vol. 6, no. 2, pp. 296-317, 1995.
38. J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *OSDI 2004*, 2004, pp. 137–150.
39. (2014, Jan.) Smart Metering Entity website. [Online]. Availble: `http://www.smi-ieso.ca/mdmr`
40. D. Nauck, R. Kruse, "Obtaining interpretable fuzzy classification rules from medical data," *Artificial Intelligence in Medicine*, vol. 16, pp. 149-169, 1999.
41. (2014, Jan.) Department of Transportation: Intelligent Transportation Systems New Generation 911 website. [Online]. Available: `http://www.its.dot.gov/NG911/`
42. D. V. Patil, R. S. Bichkar, "A Hybrid Evolutionary Approach To Construct Optimal Decision Trees With Large Data Sets," In *IEEE International Conference on Industrial Technology*, pp. 429–433, 2006.
43. T. Rappaport, *Wireless Communications: Principles and Practice*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001.
44. (2014, Jan.) Rayzit website. [Online]. Available: `http://www.rayzit.com`
45. C. Reeves, "Genetic algorithms," *Handbook of Metaheuristics*, Kluwer, pp. 65-82, 2003.
46. N. Roussopoulos, S. Kelley, and F. Vincent, "Nearest neighbor queries," in *Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '95. New York, NY, USA: ACM, 1995, pp. 71–79.
47. U. Seiffert, F.-M. Schleif, D. Zhlke, "Recent trends in computational intelligence in life sciences," In *ESANN*, 2011.
48. M. Setnes, R. Babuska, H. Verbruggen, "Rule-based modeling: precision and transparency," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* vol. 28, pp. 165-169, 1998.
49. W. Shang, H. Huang, H. Zhu, Y. Lin, Z. Wang, Y. Qu, "An Improved kNN Algorithm - Fuzzy kNN," In *Computational Intelligence and Security*, Lecture Notes in Computer Science, vol. 3801, pp. 741–746, 2005.
50. (2014, Jan.) Siri website. [Online]. Available: `Siri:http://www.apple.com/ios/siri/`
51. S. Papadopoulos, S. Bakiras, and D. Papadias, "Nearest neighbor search with strong location privacy," *PVLDB*, vol. 3, no. 1, pp. 619–629, 2010.
52. S. Thomas, and Y. Jin, "Reconstructing biological gene regulatory networks: where optimization meets big data," *Evolutionary Intelligence*, pp. 1-19, 2013.
53. (2014, Jan.) Universal Mobile Telephone System World website. [Online]. Available: `http://www.umtsworld.com/technology/capacity.htm`
54. P. M. Vaidya, "An o(n log n) algorithm for the all-nearest-neighbors problem," *Discrete Comput. Geom.*, vol. 4, pp. 101–115, January 1989.
55. Y.-R. Wang, S.-J. Horng, and C.-H. Wu, "Efficient algorithms for the all nearest neighbor and closest pair problems on the linear array with a reconfigurable pipelined bus system," *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, pp. 193206, March 2005.
56. (2014, Jan.) Waze website. [Online]. Available: `Waze,http://www.waze.com/`
57. C. Xia, H. Lu, B. C. Ooi, and J. Hu, "Gorder: an efficient method for knn join processing," in *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, ser. VLDB '04. VLDB Endowment, 2004, pp. 756–767.
58. B. Yao, F. Li, and P. Kumar, "K nearest neighbor queries and knn-joins in large relational databases (almost) for free," in *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, 2010, pp. 4–15.

59. X. Yu, K. Q. Pu, and N. Koudas, "Monitoring k-nearest neighbor queries over moving objects," in *Proceedings of the 21st International Conference on Data Engineering*, ser. ICDE '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 631–642.
60. C. Yu, B. Cui, S. Wang, and J. Su, "Efficient index-based knn join processing for high-dimensional data," *Information Software Technologies*, vol. 49, no. 4, pp. 332–344, Apr. 2007.
61. L. Zadeh, "Roles of soft computing and fuzzy logic in the conception, design and deployment of information/intelligent systems," In *Computational Intelligence: Soft Computing and Fuzzy-Neuro Integration with Applications*, Sringer Verlag, Germany, pp. 1–9, 1998.
62. J. Zhang, N. Mamoulis, D. Papadias, and Y. Tao, "All-nearest-neighbors queries in spatial databases," *Scientific and Statistical Database Management, International Conference on*, vol. 0, p. 297, 2004.
63. C. Zhang, F. Li, and J. Jestes, "Efficient parallel knn joins for large data in mapreduce," in *Proceedings of the 15th International Conference on Extending Database Technology*, ser. EDBT '12. New York, NY, USA: ACM, 2012, pp. 38–49.
64. Q. Zhang, H. Li, "MOEA/D: A Multi-objective Evolutionary Algorithm Based on Decomposition," IEEE Transactions on Evolutionary Computation, 2007.