Crowdsourcing Emergency Data in Non-Operational Cellular Networks

Georgios Chatzimilioudis[a], Constantinos Costa[a], Demetrios Zeinalipour-Yazti[a,*], Wang-Chien Lee[b]

[a]*Department of Computer Science, University of Cyprus, P.O. Box 20537, 1678 Nicosia, Cyprus*
[b]*Department of Computer Science & Engineering, Pennsylvania State University, University Park, PA 16802, USA*

Abstract

In overloaded or partially broken (i.e., non-operational) cellular networks, it is imperative to enable communication within the crowd to allow the management of emergency and crisis situations. To this end, a variety of emerging short-range communication technologies available on smartphones, such as, Wi-Fi Direct, 3G/LTE direct or Bluetooth/BLE, are able to enable users nowadays to shape point-to-point communication among them. These technologies, however, do not support the formation of overlay networks that can be used to gather and transmit emergency response state (e.g., transfer the location of trapped people to nearby people or the emergency response guard.) In this paper, we develop techniques that generate the *k-Nearest-Neighbor (kNN)* overlay graph of an arbitrary crowd that interconnects over some short-range communication technology. Enabling a kNN overlay graph allows the crowd to connect to its geographically closest peers, those that can physically interact with the user and respond to an emergency crowdsourcing task, such as seeing/sensing similar things as the user (e.g., collect videos and photos). It further allows for intelligent synthesis and mining of heterogeneous data based on the computed kNN graph of the crowd to extract valuable real-time information. We particularly present two efficient algorithms, namely *Akin+* and *Prox+*, which are optimized to work on a resource-limited mobile device. We use Rayzit, a real-world crowd messaging framework we develop, as an example that operates on a kNN graph to motivate and evaluate our work. We use mobility traces collected from three sources for evaluation. The results show that *Akin+* and *Prox+* significantly outperform existing algorithms in efficiency, even under a skewed distribution of users.

1. Introduction

In the age of smart urban and mobile environments, the mobile crowd generates and consumes massive amounts of heterogeneous data [27, 34]. Such streaming data offer the potential of enhanced science and services, such as emergency and crisis management services, among others. The availability of such services is specifically important in scenarios where a *cellular network becomes non-operational*.

A cellular network is deemed *non-operational* when there is no (sufficient) network connectivity. This might happen due to damage caused by a disaster (e.g., major flooding), or due to overloading caused by an unexpectedly large crowd trying to access telecommunication services simultaneously, e.g., consider the connection problems mobile users have faced during public celebrations of New Year's Eve.

Each cellular tower has a limited capacity of users it can service simultaneously. Specifically, each cellular tower has a limitation on its communication bandwidth to the carrier (backhaul bandwidth), a limitation on the aggregate bandwidth capacity offered by the spectrum and protocol used for wireless communication, and a limitation on the capacity of the network gears [24].

*Corresponding Author: Tel: +357-22-892755; Fax: +357-22-892701
Email addresses: gchatzim@cs.ucy.ac.cy (Georgios Chatzimilioudis), costa.c@cs.ucy.ac.cy (Constantinos Costa), dzeina@cs.ucy.ac.cy (Demetrios Zeinalipour-Yazti), wlee@cse.psu.edu (Wang-Chien Lee)

Figure 1: (left) Large crowd protesting in Syria (Reuters 2014), (right) Woman using her mobile while waiting for help in China floods (Reuters 2012).

The following are some real-world scenarios making a network non-operational. These are cases where emergency and crisis management services are needed the most.

Ad-Hoc Event Services. Large ad-hoc events can be cultural festivals (e.g., Woodstock, Old Car enthusiast gatherings), sporting events, conventions and fairs, ad-hoc demonstrations (e.g., Occupy Wall Street 2011) and ad-hoc protests (e.g., Egypt 2013, Syria 2014, Romania 2014, Hong Kong 2014) as seen in Figure 1 (left). *In such scenarios, being able to monitor and provide communication within the crowd can aid organizing authorities to better manage the crowd*[1] *and prevent lethal crowd disasters*[2,3]. Additional services can also be applied, like new entertainment services[4] and crowd-games[5].

[1]WorkingWithCrowds, Online: http://www.workingwithcrowds.com/
[2]Love Parade disaster, Online: http://goo.gl/2FpIbm
[3]Hillsborough disaster, Online: http://goo.gl/xvLRlc
[4]Opphos, Online: https://www.sics.se/projects/opphos
[5]CrowdControlGames, Online: http://crowdcontrolgames.com/

Crowdsourced Emergency Response. During a disaster, mobile users can be both victims and rescuers involved not only in receiving but also providing help from/to their neighboring peers (see Figure 1 (right)). *In this scenario, providing communication means within the crowd and streaming information to the first response team is vitally important because it can aid in organizing the crowd, allocating peer-to-peer aid and experts, and distributing tools and medicine optimally*. This would enable better management of the disaster monitoring and response cycle, where citizens can get involved in decision making, data acquisition, and advanced planning[6]. In a real world example, it was the citizen's joint efforts to map the 2012 floods in China that materialized faster and more accurately than that government-sanctioned map[7].

When the cellular network is non-operational, users can not rely on online services for their whereabouts, well-being and communication. Meanwhile, the authorities may not be able to receive all the information needed for intelligent synthesis in order to enable advanced services. In other words, the crowd is not able to generate possibly valuable information (e.g., sensors, tweets) and the authorities are not able to collect this information. Consider the example of a disaster response, the authorities need to operate in four phases: i) decision making; ii) implementation in the field; iii) evaluation of the results; and iv) making decisions. The right situational awareness is the key for decision making in such cases. The crisis management operator needs to have the right tools to communicate with the affected citizens. The operator has to evaluate and monitor the situation in order to learn and optimize operations in real time. It would be an omission that could lead to the loss of human lives, if technology could not support the crowd to generate data (e.g., reporting locations, victims using communication and social services to spread their situation) and the crisis management operator to collect and process this data during these phases.

It is imperative, therefore, to be able to create some overlay network that would connect cellular users by exploiting any available device-to-device short-range communication technology (e.g., Wi-Fi Direct, 3G/LTE direct or Bluetooth/BLE). It is equally important to ensure that this overlay network is *operational*, therefore hot-spots (e.g., too many devices trying to connect to each other), bottlenecks and disconnected components need to be avoided. For this reason we opt for using a kNN overlay graph to configure a communication network, since it guarantees an upper limit of connections per user and according to the crowd size n we can easily set the parameter k to guarantee that the graph is fully connected. Particularly, it leads to connected topologies if the degree $k > log_2 n$ [4]. In the Emergency Response scenario for example, the location of the users in the crowd can be first collected using a breadth-first-search broadcast over Wi-Fi Direct [32, 19] and then the computation of kNN overlay network can take place with the propositions in this work, determining which mobile nodes will connect to each other.

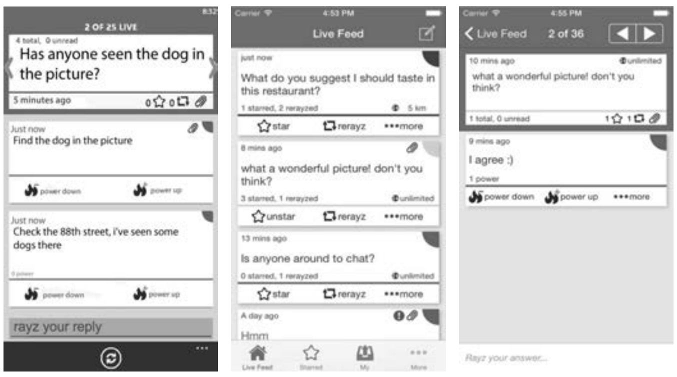

Has anyone seen the dog in the picture?
Find the dog in the picture
Check the 88th street, i've seen some dogs there
What do you suggest I should taste in this restaurant?
what a wonderful picture! don't you think?
Is anyone around to chat?
Hmm
what a wonderful picture! don't you think?
I agree :)

Figure 2: Our Rayzit [11] crowd messenger enabling users to interact with their k geographic Nearest Neighbors.

Specifically for mobile environments, valuable information can be mined based on the relations within k geographically nearest neighbors (kNN) [37, 15, 36, 23, 25]. Similar to a graph formed by social relationships, a kNN graph is formed by connecting each user to its geographically nearest neighbors. Such a graph can be mined and queried to produce valuable information, e.g., link prediction, shortest hop-distance, large connected communities, socialization suggestions, most central people, most influential people, etc [1].

In this paper, we develop centralized techniques that allow for the fast computation of the kNN graph in-situ on a mobile device for *non-operational* cellular network scenarios. With the kNN graph at hand, we can then create an overlay network that connects each victim with its k geographically closest users and use this to enable a variety of advanced services. The efficiency of the algorithm allows for frequent re-computations in order to adapt to the movement of the crowd. Our solution however is not designed as a continuous operator, where prior network state is utilized in subsequent network formations, mainly because of the transient network structure of the crowd. On the contrary, we opt for a stateless solution that is re-computed in-situ in an ongoing manner (e.g., every minute). We use Rayzit [11][8], a real-world crowd messaging framework we develop, as an example that operates on a kNN graph to motivate and evaluate our work (see Figure 2).

In our previous work, we have presented a centralized algorithm, called Proximity [8], which deals with All k Nearest Neighbors (AkNN) queries on a server that can collect the geographic coordinates of users on an ongoing basis. In this work, we re-focus the problem formulation by tackling the AkNN query-processing problem in emerging short-range communication overlay topologies. Particularly, we deal with AkNN computations directly on a smartphone of a crisis management operator that physically resides on the place of an emergency event where an overloaded or partially broken cellular network may exist. Our new contributions are summarized as follows:

- We adapt the previously proposed Proximity algorithm to operate in scenarios where the cellular network base

[6]Insight Project EU, Online: http://www.insight-ict.eu/

[7]Eric Blattberg "The Crowd Maps Beijing Floods" 2012, Online: http://goo.gl/0DHZ4v

[8]Rayzit – Rayz your message, Online: http://www.rayzit.com/

stations are not operational. Particularly, we apply the following improvements: i) we deploy a pre-processing step that partitions the space using an equi-width grid and implement the *Proximity* algorithm on top of this partitioning; ii) we propose an optimization that implements a tighter bound for the candidate set and achieving reduced CPU time. This new bound still guarantees correctness of results; iii) we propose a bulk processing method for the construction of the candidate set that trades slightly higher memory usage for smaller CPU times; and iv) we propose a new pruning heuristic for the final search phase. This heuristic is based on the fact that the candidates are already ordered. As a result, the candidate bound can be decreased as the nearest neighbors for a specific user are found while the candidate set is scanned.

- We provide an analytical study for the performance, scalability and correctness of our algorithm, showing that it can perform very well independent of the deployment scale and the distribution of input objects.
- We conduct an extensive experimental evaluation that validates our analytical results and shows the superiority of our propositions over competitive algorithms. Particularly, we use three different datasets to test implementations of proposed algorithms and find significant performance improvements.

The remainder of the paper is organized as follows. Section 2 provides the related work on AkNN query processing. Section 3 provides our problem definition, system model and desiderata. Section 4 presents the phases and algorithms of our AkNN framework, and analyzes correctness and complexity. Section 5 presents an extensive experimental evaluation and Section 6 concludes the paper.

2. Related Work

We provide a summary of existing state-of-the-art research works on neighborhood queries and categorize them according to the characteristics of these queries. Existing works can be classified in spatial data applications and spatio-temporal data applications.

2.1. kNN for Spatial Data

kNN search is a classical problem with many centralized algorithms that find applications in computational geometry [10, 6, 14] and image processing [35, 22]. An *All kNN (AkNN)* query, which can be viewed as a generalization of the basic kNN query, generates a kNN graph. For large datasets residing on disk (external memory), works like Zhang *et al.* [42], Chen *et al.* [9], and Sankaranarayanan *et al.* [30] exploit possible indices on the datasets and propose algorithms for R-tree based AkNN search. For smaller problems, where data fits inside main memory, early work in the domain of computational geometry has proposed solutions. Clarkson *et al.* [10] was the first to solve the *ANN* problem followed by Gabow *et al.* [14], Vaidya [33] and Callahan [6]. Given a set of points, a special quad-tree is used in [10, 14, 6] and a hierarchy of boxes to divide the data and compute the *ANN* is proposed in [33]. 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, the algorithms developed in [10, 6] propose an algorithm with *O(kn+nlogn)*, while the algorithm in [33] achieves *O(knlogn)*-time complexity.

2.2. kNN for Spatio-Temporal Data

In spatio-temporal data applications the datasets consist of objects and queries that move over time in some Euclidean space. Existing works in this category only tackle the problem of answering a k-nearest neighbor query for a single user over time (*CkNN* query).

For large-scale disk-resident datasets, Tao *et al.* [31], Benetis *et al.* [3], Iwerks *et al.* [18], Raptopoulou *et al.* [28], and Frentzos *et al.* [13] assume that the velocity of the moving objects is fixed and the future position of an object can be estimated. Huan *et al.* [17] assume that there is only some uncertainty in the velocity and direction of the moving objects. Accordingly, they propose algorithms to optimize the case were the future position estimation can also be uncertain. This set of works uses time parameterized R-trees to efficiently search for the nearest neighbors. Kollios *et al.* [20] propose a method to answer *NN* queries for moving objects in 1D space. Their method is based on the dual transformation where a line segment in the native space corresponds to a point in the transformed space, and vice-versa. Xiong *et al.* [38] focus on multiple *kNN* queries and propose an incremental search technique based on hashing objects into a regular grid, keeping CPU time in mind. The main objective of these works on disk-resident data is to minimize disk I/O operations, considering CPU time only as a secondary objective in the best case.

Main-memory processing is usually mandatory for spatio-temporal applications, where objects are highly mobile. The intensity of the location updates is very restrictive for disk-based storage and indexing, and demands optimization in respect to the CPU time. Yu *et al.* [39] (*YPK*) followed by Mouratidis *et al.* [26] (*CPM*) and Hu *et al.* [16] optimize *kNN* queries in a similar fashion as Xiong *et al.* do for disk-resident data. Data objects are indexed by a grid in main-memory (see Figure 3) 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*. Assuming a grid size of $\sqrt{n}$ cells, their stateless solution has a time complexity of $O(n^{1.5})$ for uniform distributions and $O(n^{2.5})$ for the worst-case distribution, where the search for most of the users needs to be deepened iteratively until it covers most of the space.

2.3. Mobile User Community Network

Similar to the motivating examples in the previous section, Konstantinides *et al.* [21] present a distributed search architecture for intelligent search of objects in a mobile social community. Their framework is founded on an in-situ data storage

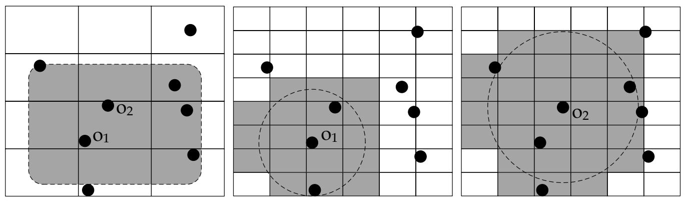


Figure 3: a) In *Proximity* the candidate set is pre-constructed for all users of the same cell (e.g., o_1 and o_2); whereas b) for existing state-of-the-art algorithms the candidate set needs to be iteratively discovered by expanding a ring search for each user separately into neighboring cells.

Table 1: Summary of Notation

Notation	Description
o, O, n	Object o, set of all o, $n = \|O\|$
$kNN(o, O)$	k nearest neighbors of o in O
$AkNN(O)$	k nearest neighbors of every o in O
$dist(o_a, o_b)$	L_p-norm distance between o_a and o_b
c, C, O_c	Cell c, set of all c, objects in c
S_c	Candidate Set of c

model, where captured objects remain local on smartphones. It searches over a sophisticated structure that is computed dynamically and optimizes several conflicting objectives in a single run. Then a decision-making subsystem is utilized to tune the retrieval preferences of the query user. Their framework yields high query recall rates with minimized CPU time on each mobile device.

3. System Model

This section formalizes our system model and design principles. Our main notation is summarized in Table 1.

The *k Nearest Neighbors (kNN)* of an object o from some dataset O, denoted as $kNN(o, O)$, are the k objects that have the most similar attributes to o [29]. Formally, given objects $o_a \neq o_b \neq o_c$ and $\forall o_b \in kNN(o_a, O)$ and $\forall o_c \in O - kNN(o_a, O)$, it always holds that $dist(o_a, o_b) \leq dist(o_a, o_c)$[9].

An *All kNN (AkNN)* query, viewed as a generalization of the basic kNN query, computes the $kNN(o, O)$ result for every $o \in O$ and has a quadratic worst-case bound. An AkNN search can alternatively be viewed as a *kNN Self-Join* operation: *Given a dataset O and an integer k, the kNN Self-Join of O combines each object $o_a \in O$ with its k nearest neighbors from O, i.e.,* $O \bowtie_{kNN} O = \{(o_a, o_b) | o_a, o_b \in O$ *and* $o_b \in kNN(o_a, O)\}$.

Research Goal. *Given a set of objects O with their locations in a bounding area, a resource-poor query processor QP computes the AkNN result of O by minimizing* CPU time.

Let O be a set of n mobile users moving in a planar area, denoted by their current locations, which can either be obtained at a fine granularity (e.g., GPS and RadioMaps) or coarse granularity (e.g., Cell_ID and Wi-Fi_ID databases). Assume that there is a mobile device, denoted as QP (Query Processor), which collects the locations of all users in user set O. Our main desiderata is to minimize CPU time on QP, given that this determines the energy consumption of QP and also determines how frequently the kNN networks can be re-computed.

[9]In our discussion, *dist* can be any L_p-norm, such as Manhattan (L_1), Euclidean (L_2) or Chebyshev (L_∞).

3.1. Emergency Network Model

In a disaster scenario with a non-operational cellular network, the first task is to construct an ad-hoc communication network that would enable information routing within the crowd. This can be done using existing technologies (e.g., Wi-Fi Direct [24]) and algorithms [32, 19], assuming that each user can directly communicate with peers that are within a certain communication radius. With the initial ad-hoc network in place, each user can send out a request to construct a query routing tree, using techniques like [40, 41, 2, 7], in order to collect the location of the users in the crowd and compute the kNN graph. We assume that only one request completes, e.g., by letting the request with the oldest initiation timestamp dominate. Therefore, one user within each connected overlay network retrieves the needed locations and computes the kNN graph.

4. Centralized AkNN Query Processing

In this section, we introduce *Proximity*, a centralized AkNN query processing framework [8] we developed previously. We then describe the adaptations undertaken to make it functional for non-operational cellular network scenarios using a grid partitioning. Subsequently, we also propose two optimizations, namely *Prox* and *Akin*, which reduce the CPU time for the AkNN computation. We conclude with some further improvements upon *Prox* and *Akin*, coined *Prox+* and *Akin+*, which are founded on more powerful pruning heuristics than their basic counterparts.

4.1. Background on Proximity

The *Proximity* framework is designed in such a way that it is: i) *Memory-resident*, since the dynamic nature of mobile user makes disk resident processing prohibitive; ii) *Specifically designed* for *highly mobile* and *skewed distribution* environments performing equally well in congested and sparsely-deployed scenarios; and iii) *Infrastructure-ready*, since it does not require any additional infrastructure or specialized hardware.

The original *Proximity* algorithm exploits the natural geographic partitioning that is provided by the *Network Connectivity Points (NCP)* of a cellular network (e.g., cell towers). This partitioning allows for the derivation of kNN candidate sets, which provide the AkNN answer-set in linear time. Particularly, for every timestep *Proximity* works in two phases: In the first phase one *k*+-heap data structure is constructed per *NCP*, using the location reports of users (the *k*+-heap will be discussed

Algorithm 1 . Proximity with Grid Partitioning

```
Input: User locations $O$, set $C$ of all grid cells
Output: kNN answer-set for each user in $O$
 1: for all $c \in C$ do
 2:     initialize $S_c$                                  ▷ Initialize our k+-heap
 3: end for
 4: for all $o \in O$ do                                 ▷ Phase 1: build k+-heap
 5:     for all $c \in C$ do
 6:         insert$(o, S_c)$
 7:     end for
 8: end for
 9: for all $o \in O$ do                                 ▷ Phase 2: scan k+-heap
10:     $kNN_0 = \emptyset$                              ▷ Conventional k-max heap
11:     $c \leftarrow o.cell$
12:     for all $o' \in S_c$ do
13:         if $o'$ is a kNN of $o$ then
14:             update$(kNN_o, o')$
15:         end if
16:     end for
17: end for
```

Algorithm 2 . k+-heap: Insert(o_{new}, c)

```
Input: Object to be added $o_{new}$, Cell $c$ of k+-heap
Output: $S_c$ updated
 1: $kth_c \leftarrow head(K_c)$
 2: if $contained\_inside(o_{new}, c)$ then
 3:     $insert(o_{new}, O_c)$
 4: else if $dist(o_{new}, c) < dist(kth_c, c)$ then
 5:     $insert(o_{new}, K_c)$
 6:     if $K$ heap has more than $k$ elements then
 7:         $kth_c \leftarrow pophead(K_c)$
 8:         $insert(kth_c, B_c)$
 9:         $Update\_boundary(head(K_c))$
10:     end if
11: else if $dist(o_{new}, c) < dist(kth_c, c) + diag_c$ then
12:     $insert(o_{new}, B_c)$
13: else
14:     discard $o_{new}$
15: end if
```

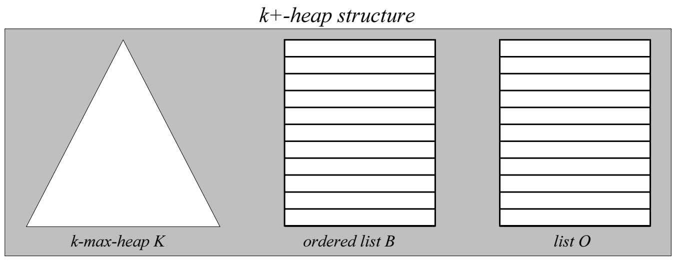


Figure 4: A visualization of a k+-heap (denoted as S_c) for a specific cell c, comprises of three structures: O_c, K_c and B_c.

more thoroughly in 4.2). 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.

4.2. Proximity with Grid Partitioning

In this section we adapt the *Proximity* algorithm such that it is effective even under a non-operational cellular network, where there are no NCPs to naturally partition the space. In Algorithm 1 we outline the proposed solution that operates over a grid partitioning of the space into *equi-width* cells C. In Algorithm 2 we discuss in more detail the specifics of the insertion procedure of the basic underlying k+-heap data structure.

Algorithm 1 (Outline): Each cell $c \in C$ contains a disjoint subset $O_c \subset O$ of objects. Algorithm 1 computes a correct candidate set S_c for each cell $c \in C$ by constructing a k+-heap data structure for each cell (lines 1-8). In the second phase, the kNN for each user $o \in O$ are determined by scanning the respective k+-heap and computing *kNN*(o, S_c) (lines 9-17).

The k+-heap consists of three separate data structures (see Figure 4): i) an unordered list of internal objects lying inside the boundary of c (coined O_c); ii) a conventional k-max-heap (coined K_c) capturing the closest external candidates; and iii) an ordered list of external boundary objects (coined B_c), which are necessary besides K_c for the correct computation of any AkNN query.

The K_c and B_c structures are defined according to definitions 1 and 2, respectively. In Definition 3, we conclude with a formal definition of the k+-heap (coined *Candidate Set*, S_c).

Definition 1 (K_c, k-max-heap of cell c)**.** *Given a set of external users $O - O_c$, which is ordered with ascending distance $dist(o, c)$ to the border of cell c, set K_c consists of the top-k elements of this set.*

Definition 2 (B_c, Boundary Set of cell c)**.** *Given a cell c and its k^{th} closest external user to the border of c, set B_c consists of all users $o \in O - (O_c \cup K_c)$ with distance $dist(o, c) < dist(kth_c, c) + diag_c$ from the border of c. In other words B_c consists of all users $o \in O$ with distance $dist(kth_c, c) < dist(o, c) < dist(kth_c, c) + diag_c$.*

Definition 3 (S_c, Candidate Set of cell c)**.** *Given a cell c and its O_c, K_c and B_c sets, the candidate set S_c of c consists of users $o \in O_c \cup K_c \cup B_c$.*

Performance Analysis of Prox: Assuming a grid size of $\sqrt{n}$ cells and uniform data distribution of n objects, Algorithm 1 runs in $O(n^{1.5}\ log\sqrt{n})$ time. Particularly, the loop at line 1 runs in $O(\sqrt{n})$. The loop at line 4 runs in $O(n^{1.5} log\sqrt{n})$, given that the insertion procedure of the heap runs in $O(log\sqrt{n})$. Finally, the last loop in line 9 runs in $O(n^{1.5})$, given that each k+-heap has $O(\sqrt{n})$ users under a uniform distribution assumption. In the worst-case distribution, where all users fall in a single cell this algorithm runs in $O(n^2)$ time.

Algorithm 2 (Insertion): We now discuss in more detail the specifics of the insertion procedure of the k+-heap. When inserting a new element o_{new} into the k+-heap of c, we distinguish among four cases (see Algorithm 2): i) o_{new} is covered by c and belongs to set O_c (line 2), ii) o_{new} belongs to set K_c (line 4), iii) o_{new} belongs to set B_c (line 11), or iv) o_{new} does not belong to the candidate set $S_c = O_c \cup K_c \cup B_c$ of cell c (line 13). In case (i) the element is inserted into the O_c list. In case (ii) we need to insert o_{new} into heap K (line 5) and move the current head kth_c from K to the boundary list B (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 2. In case (iii) we insert o_{new} into the ordered boundary list B (line

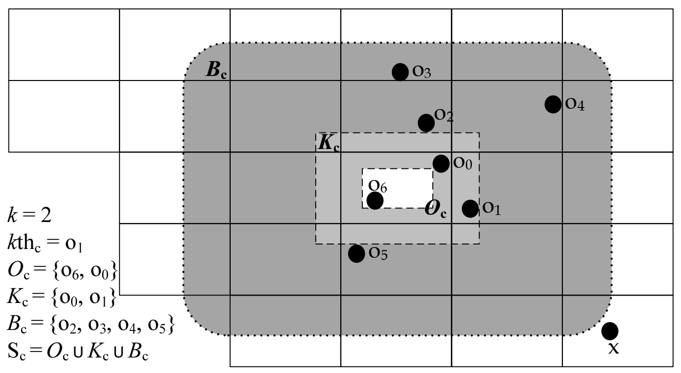


Figure 5: (Running Example) The construction of S_c with k=2. The candidate set S_c of c is $\{o_0, o_1, o_2, o_3, o_4, o_5, o_6\}$ and is represented by the area within the dotted line with the rounded corners. Set S_c includes all users O_c inside c (solid line cell), users inside K_c the lighter square ring and the users B_c inside the darker ring. 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*$(o_0) = \{o_1, o_2\}$ and *kNN*$(o_6) = \{o_5, o_0\}$.

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.

4.3. Prox: An Optimized Candidate Set Bound

Our initial *Proximity* algorithm has a suboptimal bound determining the candidate set per cell. Here we propose a tighter bound, coined *Prox*, which is founded on the observation that the K'_c set inside a *k*+-heap could have been constructed from objects that reside both inside a cell and outside a cell, as opposed to the original *Proximity* algorithm, where the K_c set emerged from objects residing outside a given cell only. Particularly, we use the following definition to define the optimized candidate set bound:

Definition 4 (K'_c, smaller K_c)**.** *Given a set of any user in O, which is ordered with ascending distance $dist(o, c)$ to the border of cell c, set K'_c consists of the top-k elements of this set.*

Note that the difference of K_c to K'_c is that the former definition includes only users *outside* cell c. This does not allow for a tight bound and a minimum cut-off threshold θ_c. Particularly, by expanding the scope of K'_c, immediately makes B'_c smaller or equal in size to B_c. This effectively allows Prox to process a smaller search space, as the S'_c set now comprises of $B'_c \cup O_c \cup K'_c$, as opposed to $B_c \cup O_c \cup K_c$. Another way to present the difference is to mention that with Prox, it holds that $dist(kth'_c, c) \leq dist(kth_c, c)$.

Running Example of Prox: We shall now present a running example of Prox in Figure 5. Consider that the following object locations arrive at QP: $O = \{o_0, o_1, o_2, o_3, o_4, o_5, o_6, o_x\}$. Every object is again inserted into every *k*+-heap on the QP (see Algorithm 1, lines 1-5). The order in which the objects are inserted into a *k*+-heap does not affect the correctness of the candidate set. For our example, assume that the objects are inserted in the order seen in the first column of Table 2. For

Table 2: Build-up phase of S_c in Prox as object locations are inserted

Object	**Set K'_c**	**Set B'_c**	**Set O_c**
o_4	$\{o_4\}$	$\{\}$	$\{\}$
o_x	$\{o_x, o_4\}$	$\{\}$	$\{\}$
o_2	$\{o_4, o_2\}$	$\{o_x\}$	$\{\}$
o_3	$\{o_3, o_2\}$	$\{o_4, o_x\}$	$\{\}$
o_1	$\{o_2, o_1\}$	$\{o_3, o_4\}$	$\{\}$
o_5	$\{o_2, o_1\}$	$\{o_3, o_4, o_5\}$	$\{\}$
o_6	$\{o_2, o_1\}$	$\{o_3, o_4, o_5\}$	$\{o_6\}$
o_0	$\{o_0, o_1\}$	$\{o_2, o_3, o_4, o_5\}$	$\{o_6, o_0\}$

every insertion we can see the contents of S'_c in the same Table (i.e., last three columns). For simplicity, we only follow the operation on the S'_c of cell c (that similarly applies to all cells).

When object o_4 is inserted into S'_c it is added to the heap K'_c, which records the closest objects around the cell border of c (both internal and external objects to c). The same logic applies to the next object o_x. At this point, however, the K'_c heap gets full (assume it is a 2-max-heap). The third insertion of o_2 into K'_c evicts o_x from K'_c (given that in Figure 5, o_2 and o_4 are the closer to the boundary c than o_x). It however does not discard o_x, given that it might still be a good candidate for some hypothetical other object o_y in c (e.g., one that resides on the opposite site of the cell).

Before transferring it blindly to B_c, o_x is checked against the new threshold: $\theta'_c = dist(o_4, c) + diag_c$. Given that o_x is below θ'_c, it is inserted into B_c. Alternatively, o_x would have been discarded. A similar procedure is followed for the next three insertions, i.e., o_3, o_1 and o_5. It is important to mention that on each insertion, θ'_c might become smaller. Every time this happens, objects inside B'_c have to be re-evaluated and discarded accordingly (e.g., o_x is discarded when o_1 is inserted). As a concluding remark, notice that any "inside" object (e.g., o_6 and o_0), is automatically added to O_c but these objects are also considered for K'_c (i.e., in our example only o_0 qualifies to be part of K'_c as it is close to the border c).

Phase 1 of Algorithm 1 is completed and the candidate sets are ready after all objects are inserted into the S_c sets. In phase 2 the server scans a single S'_c for each user o_x, according to the cell o_x is mapped to. For users o_0 and o_6, the server QP scans $S_c = o_2, o_1, o_3, o_4, o_5, o_6$ and finds nearest neighbors $\{o_2, o_1\}$ and $\{o_5, o_0\}$, respectively.

4.4. Akin: Bulk Candidate Set Construction without a k+-heap

In this section, we introduce an alternative technique, called *Akin*, to reduce the CPU time for the candidate set construction. Our proposition is founded on a bulk construction of the search space without a *k*+-heap. Particularly, we adopt the linear-time heap construction algorithm proposed by Robert Floyd in [12]. The given algorithm, makes the necessity of breaking our initial search space into three sub-structures unnecessary, given that a single heap is constructed for the complete search space of each cell in linear time.

Algorithm 3 . *Akin*(O, S_c) Algorithm

```
Input: Candidate set $S_c$ and set of objects $O$
Output: $S_c$ updated
1: construct Min Heap $H_c$ from $O$ based on dist$(o, c)$
2: $kth_c$ = extract top k objects from $H_c$
3: $\theta_c = diag_c + dist(kth_c, c)$
4: for all $o \in O$ do
5:     if dist$(o, c) < \theta_c$ then
6:         $S_c = S_c \cup o$
7:     end if
8: end for
9: $S_c = S_c \cup O_c$
```

In Algorithm 3, we present our Akin algorithm for the search space construction of cell $c \in C$. Particularly, we scan once each object $o \in O$ to build a k-min heap H_c based on the minimum distance *dist*(o, c) between o and the cell-border (line 1). The k objects are then scanned from H_c to determine threshold θ_c of c. We subsequently scan objects $o \in O$ once again to determine the set S_c of objects that satisfy the threshold (lines 4-8). At the end of the execution, we carry out the union of O_c and S_c to derive the updated search space.

Performance Analysis of Akin: Assuming a grid size of $\sqrt{n}$ cells and uniform data distribution of n objects, Algorithm 3 runs in $O(n^{1.5})$ time. Particularly, the heap construction in line 1 takes $O(n)$ using the Floyd algorithm. The subsequent loop in line 4 also takes $O(n)$, consequently the above complexity is $O(n)$ for each cell. For all $\sqrt{n}$ cells the AkNN computation cost is $O(n^{1.5})$. In the worst-case distribution, where all users fall in a single cell this algorithm runs in $O(n^2)$ time.

4.5. Internal Pruning of Candidate Set: Prox+ and Akin+

In this section we introduce an internal pruning strategy of the candidate set S_c, which is applicable to both the Prox and the Akin algorithms introduced earlier. We denote the respective algorithms with the extension '+', if they use this further optimization, i.e., Prox+ and Akin+. The particular strategy can be applied once the S_c search space has been constructed, at which point the algorithm is ready to return the AkNN results for each user.

Particularly, the internal pruning heuristic is founded on the observation that the internal structures of the S_c structure (i.e., O_c, K_c and B_c), can be accessed in a particular order to maximize the possibility of converging early-on with the AkNN result-set for each user. Particularly, the heuristic strategy attempts to refine the pruning threshold θ_c as soon as possible, by beginning the exploration of the S_c set from O_c, then if necessary proceed to K_c and finally, if again necessary, proceed to the exploration of B_c. Our experimental evaluation in Section 5 reveals that the internal pruning heuristic provides a significant improvement to both Prox and Akin.

5. Experimental Evaluation

To evaluate our proposed algorithms we conduct a set of experiments, using three traces of real datasets, in comparison with existing state-of-the-art algorithms. We run all experiments on a virtual octa-core computing node.

The goal of this evaluation is to compare the overall efficiency of our proposed algorithms with existing state-of-the-art algorithms in centralized AkNN query processing. Efficiency is determined by the running time and, in case of running the algorithm on a mobile device, by the energy consumed. Since our algorithms are centralized, both running time and energy consumed are proportional to the CPU time needed for the computation.

5.1. Datasets

In our experiments we use the following realistic and real datasets (depicted in Figure 6):

Oldenburg (realistic): The initial dataset is generated with the Brinkhoff spatio-temporal generator [5], including 5K vehicle trajectories in a 25km x 25km area of Oldenburg, Germany. The generated spatio-temporal dataset is then decomposed on the temporal dimension, in order to generate realistic spatial datasets of 100, 1000 and 10K users.

Geolife (realistic): The initial dataset is obtained from the Geolife project at Microsoft Research Asia [43], including 1.1K trajectories of users moving in the city of Beijing, China over a life span of two years (2007-2009). Similarly to Oldenburg, the generated spatio-temporal dataset is decomposed on the temporal dimension, in order to generate realistic spatial datasets of 100, 1000 and 10K users.

Rayzit (real): This is a real spatial dataset of 20K coordinates captured by our Rayzit service during February 2014. We intentionally do not scale this dataset up to more users, in order to preserve the real user distribution.

Figure 6 (second row) shows the population histograms for the three respective datasets, when split into nine equi-width partitions. The standard deviation among the buckets for a total population of 10K objects is: i) 900 objects in Oldenburg, ii) 2K objects in Geolife, and iii) 33 objects for Rayzit.

5.2. Evaluated Algorithms

We first evaluate the proposed algorithms in order to experimentally validate the ideas and superiority of our propositions. Our compared algorithms are:

Proximity: This algorithm is proposed in our previous work [8] for answering AkNN queries in an operational cellular network, which exploits the natural partitioning determined by the Network Connectivity Points (cell towers). We implement *Proximity* on top of the grid partitioning pre-processing step discussed in Section 4.2.

Prox: This algorithm implements the optimized candidate set bound, introduced in Section 4.3 and founded on the observation that the K_c' set inside a k+-heap could have been constructed from objects that reside both inside a cell and outside a cell.

Akin: This algorithm implements the optimized candidate set construction algorithm, introduced in Section 4.4. Note that it does not make use of the k+-heap structure, rather uses a heap constructed in linear time.

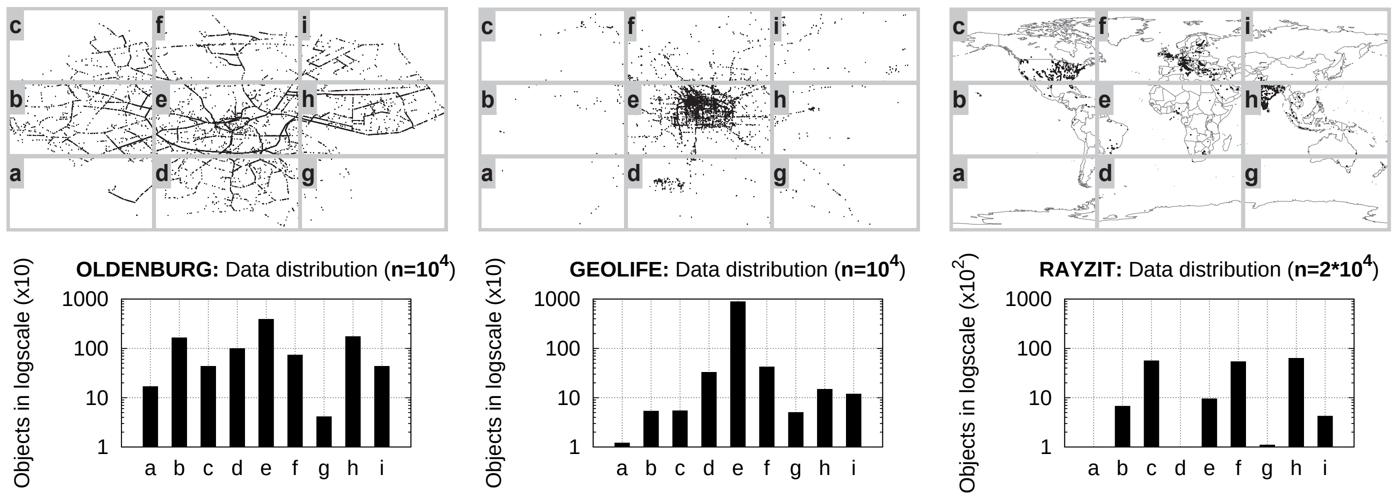


Figure 6: Datasets (top row) and population histograms (bottom row) for an indicative 3x3 partitioning.

Prox+: This is the same algorithm as *Prox* but with the internal pruning strategy described in Section 4.5. This optimization allows the final step to terminate earlier.

Akin+: This is the same algorithm as *Akin* but with the internal pruning strategy described in Section 4.5. This optimization allows the final step to terminate earlier.

We also take existing state-of-the-art algorithms for answering a kNN query for a single user, including Yu *et al.* [39] and Mouratidis *et al.* [26]. We adapt them to answer an AkNN query. In addition, we compare the adaptation of existing work to our best algorithms.

YPK and **CPM**: These methods iteratively enlarge a range search to find the *kNN* for the user (see Figure 3b). The search space starts from the cell of the user and iteratively visits neighboring cells until at least k neighbors are found. It is guaranteed that no further neighboring cell can have a user that is closer. For our experiments, we use this adaptation as a baseline for comparison.

In Table 3, we summarize the time complexity for each of the above algorithms according to the respective data distribution, where "Best" denotes the uniform distribution and "Worst" the worst distribution that corresponds to each algorithm.

Table 3: Algorithm Complexities under Best-case and Worst-case distributions.

Algorithm	Best	Worst
***Proximity, Prox,* Prox+**	$O(n^{1.5} log\sqrt{n})$	$O(n^2)$
Akin, Akin$^+$	$O(n^{1.5})$	$O(n^2)$
CPM/YPK	$O(n^{1.5})$	$O(n^{2.5})$

5.3. Evaluation Metrics

CPU time is the metric we use for our evaluation. All algorithms under evaluation are centralized and, thus, do not require any communication. Therefore, CPU time captures both the running time and the energy consumed by the computing node (e.g., mobile device) to run the AkNN algorithm. The CPU times are averaged over five iterations and are measured in milliseconds. Note that all figures are plotted with the time (y-axis) in *log-scale*, thus the actual difference in efficiency between the algorithms is larger than it visually appears.

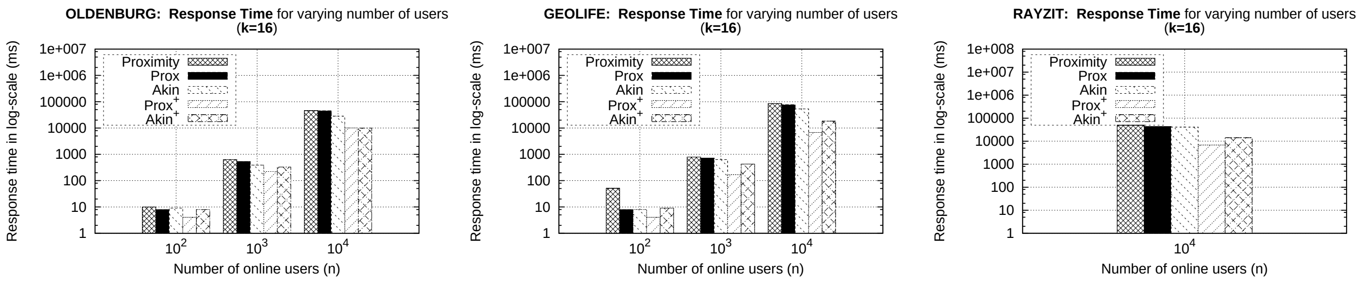
5.4. Control Experiments

In this experimental series, we evaluate our proposed optimizations. We increase the workload by growing the number of online users (n) exponentially using $n = 10^2, 10^3, 10^4$ for all datasets.

All plots in Figure 7 show that Proximity without any optimizations has the worst performance. They also show that using bulk heap construction (i.e., *Akin*) instead of a k+-heap (i.e., *Prox*) we achieve better performance. Looking at the '+' optimization (i.e., the internal pruning strategy), it is evident that *Akin*$^+$ and *Prox*$^+$ outperform their counterparts (i.e., *Akin* and Prox) every time.

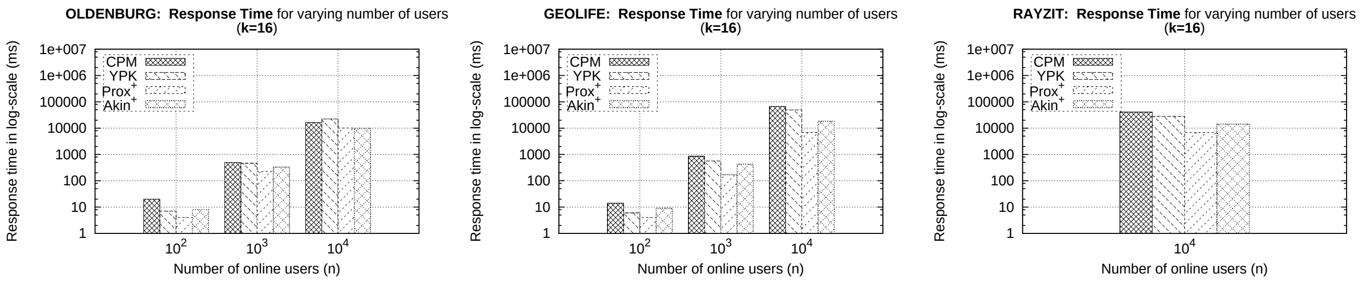
It is interesting to notice that *Akin*$^+$ does not outperform *Prox*$^+$, even though *Akin* outperforms *Prox*, and *Akin*$^+$ outperforms *Akin*. This is happening because the pruning power inside the candidate set of *Akin*$^+$ is smaller compared to *Prox*$^+$. As described in Section 4.4, *Akin* includes also internal objects in the computation of the K_c set resulting in a K_c area that is *narrower and closer* to the cell border. Therefore, when the kNNs of an internal object can not be guaranteed by the objects inside the cell (O_c set), then there is a much higher chance for *Akin*$^+$ to continue the kNN search in the much larger boundary set (B_c) instead of guaranteeing the kNNs by just expanding to the K_c set, as opposed to *Prox*$^+$.

Comparing the various datasets, we conclude that the more skewed the dataset is (e.g., Geolife) the more improvement the speed-up achieved by our optimizations. This stands also when we look at the growing workload. Generally, the proposed optimizations in this paper always outperform our original algorithm Proximity, specifically for large workloads and skewed datasets where they reach a 10% speed-up (Figure 7b).



(a) (b) (c)

Figure 7: CPU time for all algorithms using the datasets: a) Oldenburg; b) Geolife; and c) Rayzit. The plots show that: i) Proximity without any optimizations has the worst performance; ii) Internal Pruning (using the "+") has a higher impact on Prox rather than on Akin, making Prox+ the algorithm with the best CPU-time performance; iii) the more skewed the dataset is (e.g., Geolife) the more improvement the speed-up achieved by our optimizations.



(a) (b) (c)

Figure 8: CPU time for the best algorithms using the datasets: a) Oldenburg; b) Geolife; and c) Rayzit. The plots show Prox+ and Akin+ outperform adapted state-of-the-art AkNN query processing algorithms YPK and CPM that apply iterative deepening principle rather than bulk computation of the search space.

5.5. Comparison Against Existing Work

In this experimental series, we compare our best algorithms (i.e., *Prox+*, *Akin+*) against the state-of-the-art (i.e., *YPK*, *CPM*). Again, we increase the workload of the system by growing the number of online users (n) exponentially using $n = 10^2, 10^3, 10^4$ for all datasets (other than Rayzit, which is kept at 10^4 as explained previously).

Figure 8 shows that there are only two instances where a state-of-the-art algorithm (i.e., YPK) outperforms any of our algorithm (i.e., *Akin*$^+$), i.e., for the lightest workloads $n = 10^2$. Looking at the workload, it shows that the our algorithms achieve greater speed-up as the workload increases. Comparing datasets, it is evident that the more skewed the dataset the greater the speed-up achieved by our algorithms.

It can clearly be observed, that the algorithms proposed in this paper outperform existing approaches for any workload and skewness of dataset. The more skewed the dataset is (i.e., Geolife) the more improvement the speed-up of our algorithms, specifically they reach a 10% speed-up (Figure 8b). This is in line with our theoretical comparison in Table 3, where the time complexity of the state-of-the-art is greater for the worst-case data distribution.

6. Conclusions

In this paper, we develop techniques that generate the kNN graph of an arbitrary crowd of smartphone users that interconnect through a short-range communication technology, such as, Wi-Fi Direct, 3G/LTE direct or Bluetooth v4.0 (BLE). We present two efficient algorithms, namely *Akin+* and *Prox+*, optimized to work on a resource-limited mobile device. These algorithms partition the user space and compute shared candidate sets per partition. *Prox+* uses a custom heap data structure to update the candidate set as new users are inserted, whereas *Akin+* uses a bulk bottom-up construction of a simple heap to compute the candidate set once all users have been inserted. Our experiments verify the theoretical efficiency and shows that *Prox+* and *Akin+* are very well suited for large scale and skewed data scenarios.

Acknowledgments

This work was supported by an *Appcampus Award* (Microsoft, Nokia and Aalto University, Finland). It has also been supported by the third author's startup grant at the Univ. of Cyprus, EU's FP7 *"Mobility, Data Mining, and Privacy"* project, EU's COST Action MOVE *"Knowledge Discovery for Moving Objects"* and an industrial award by MTN Cyprus.

References

[1] C. C. Aggarwal "An Introduction to Social Network Data Analytics". *Springer USA*, 2011.

[2] P. Andreou, A. Pamboris, D. Zeinalipour-Yazti, P. K. Chrysanthis, G. Samaras. "ETC: Energy-Driven Tree Construction in Wireless Sensor Networks". In *Proceedings of the 10th International Conference on Mobile Data Management (MDM'09)*, pp. 513–518, 2009.

[3] R. Benetis, C.S. Jensen, G. Karciauskas, and S. Saltenis. "Nearest neighbor and reverse nearest neighbor queries for moving objects". In *Proceedings of the 2002 International Symposium on Database Engineering & Applications (IDEAS'02)*, pp. 44–53, 2002.

[4] B. Bollobs. "Modern graph theory". *Springer Science & Business Media*, Vol. 184, 1998.

[5] T. Brinkhoff. "A framework for generating network-based moving objects". *Geoinformatica*, Vol. 6, pp. 153–180, 2002.

[6] P.B. Callahan. "Optimal parallel all-nearest-neighbors using the well-separated pair decomposition". *In Proceedings of the 34th IEEE Annual Foundations of Computer Science (SFCS'93)*, pp. 332–340, 1993.

[7] G. Chatzimilioudis, D. Zeinalipour-Yazti, D. Gunopulos. "Minimum-hot-spot query trees for wireless sensor networks". In *Proceedings of the 9th ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE '10)*, pp. 33–40, 2010.

[8] G. Chatzimilioudis, D. Zeinalipour-Yazti, W.-C. Lee and M.D. Dikaiakos. "Continuous all k-nearest neighbor querying in smartphone networks". *In Proceedings of the 13th IEEE International Conference on Mobile Data Management (MDM'12)*, pp. 79–88, 2012.

[9] Y. Chen and J.M. Patel. "Efficient evaluation of all-nearest-neighbor queries". *In Proceedings of the 23rd IEEE International Conference on Data Engineering (ICDE'07)*, pp. 1056–1065, 2007.

[10] K.L. Clarkson. "Fast algorithms for the all nearest neighbors problem". *In Proceedings 24th Annual Symposium on Foundations of Computer Science (FOCS'83)*, pp. 226–232, 1983.

[11] C. Costa, C. Anastasiou, G. Chatzimilioudis and D. Zeinalipour-Yazti. "Rayzit: An Anonymous and Dynamic Crowd Messaging Architecture", *In Proceedings of the 3rd IEEE Intl. Workshop on Mobile Data Management, Mining, and Computing on Social Networks (Mobisocial'15)*, Vol. 2, pp. 98-103, IEEE Computer Society, 2015.

[12] R.W. Floyd. "Algorithm 245: Treesort". *Commun. ACM* 7, 12 (December 1964), pp. 701-., 1964.

[13] E. Frentzos, K. Gratsias, N. Pelekis and Y. Theodoridis. "Algorithms for nearest neighbor search on moving object trajectories". *Geoinformatica*, Vol. 11, pp. 159–193, 2007.

[14] H.N. Gabow, J.L. Bentley and R.E. Tarjan. "Scaling and related techniques for geometry problems". *In Proceedings of the 16th ACM symposium on Theory of computing (STOC'84)*, pp. 135–143, 1984.

[15] V. Hautamaki, I. Karkkainen and P. Franti. "Outlier Detection Using k-Nearest Neighbour Graph". In *Proceedings of the 17th International Conference on Pattern Recognition (ICPR'04)*, Vol. 3, 2004.

[16] H. Hu, J. Xu and D.L. Lee. "A generic framework for monitoring continuous spatial queries over moving objects". In *Proceedings of the 2005 ACM SIGMOD International conference on Management of data (SIGMOD'05)*, pp. 479–490, 2005.

[17] Y.-K. Huang, S.-J. Liao and C. Lee. "Efficient continuous k-nearest neighbor query processing over moving objects with uncertain speed and direction". In *Proceedings of the 20th international conference on Scientific and Statistical Database Management (SSDBM'08)*, pp. 549–557, 2008.

[18] G.S. Iwerks, H. Samet and K. Smith. "Continuous k-nearest neighbor queries for continuously moving points with updates". In *Proceedings of the 29th international conference on Very large data bases - Volume 29 (VLDB'03)*, pp. 512–523, 2003.

[19] P. Kolios, A. Pitsillides and O. Mokryn. "Bilateral routing in emergency response networks". In *20th International Conference on Telecommunications (ICT'13)*, pp. 1–5, 2013.

[20] G. Kollios, D. Gunopulos and V.J. Tsotras. "Nearest neighbor queries in a mobile environment". In *Proceedings of the International Workshop on Spatio-Temporal Database Management (STDBM'99)*, pp. 119–134, Springer-Verlag, 1999.

[21] A. Konstantinidis, D. Zeinalipour-Yazti, P. Andreou, G. Samaras and P. K. Chrysanthis "Intelligent search in social communities of smartphone users". *Distributed and Parallel Databases (DAPD13)*, Vol. 31, pp. 115–149, 2013.

[22] 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*, 7, pp. 806–817, 1996.

[23] N. Lathia, S. Hailes and L. Capra. "kNN CF: a temporal social network". In *Proceedings of the ACM conference on Recommender systems (RecSys'08)*, pp. 227–234, 2008.

[24] Y.-B. Lin and I. Chlamtac. "Wireless and Mobile Network Architectures". *John Wiley & Sons, Inc.*, 2000.

[25] W. Liu, J. Wang and S. Chang. "Hashing with graphs". In *Proceedings of the 28th International Conference on Machine Learning (ICML'11)*, pp. 1–8, 2011.

[26] K. Mouratidis, D. Papadias and M. Hadjieleftheriou. "Conceptual partitioning: an efficient method for continuous nearest neighbor monitoring". *In Proceedings of the ACM SIGMOD international conference on Management of data (SIGMOD'05)*, pp. 634–645, 2005.

[27] E.C. Ngai, M.B. Srivastava and L. Jiangchuan. "Context-aware sensor data dissemination for mobile users in remote areas". In *Proceedings of the IEEE INFOCOM'12*, pp. 2711–2715, 2012.

[28] K. Raptopoulou, A.N. Papadopoulos and Y. Manolopoulos. "Fast nearest-neighbor query processing in moving-object databases". *Geoinformatica*, Vol. 7, pp. 113–137, June 2003.

[29] N. Roussopoulos, S. Kelley and F. Vincent. "Nearest neighbor queries". *In Proceedings of the ACM SIGMOD international conference on Management of data (SIGMOD'95)*, pp. 71–79, 1995.

[30] J. Sankaranarayanan, H. Samet and A. Varshney. "A fast all nearest neighbor algorithm for applications involving large point-clouds". *Computers & Graphics*, Vol. 31, Iss. 2, pp. 157–174, 2007.

[31] Y. Tao, D. Papadias and Q. Shen. "Continuous nearest neighbor search". In *Proceedings of the 28th international conference on Very Large Data Bases (VLDB'02)*, pp. 287–298. 2002.

[32] S. Trifunovic, B. Distl, D. Schatzmann and F. Legendre. "Wi-Fi-Opp: ad-hoc-less opportunistic networking". In *Proceedings of the 6th ACM workshop on Challenged networks (CHANTS'11)*, pp. 37–42, 2011.

[33] P.M. Vaidya. "An o(n log n) algorithm for the all-nearest-neighbors problem". *Discrete Computational Geometry*, Vol. 4, pp. 101–115, 1989.

[34] C.-J. Wang and W.-S. Ku. "Anonymous Sensory Data Collection Approach for Mobile Participatory Sensing". *IEEE 29th International Conference on Data Engineering Workshops*, pp. 220–227, 2013.

[35] 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 Transactions of Parallel Distributed Systems, Vol. 16, pp. 193–206, 2005.

[36] X. Wu, V. Kumar, Q. Ross, et al. "Top 10 algorithms in data mining". *Journal of Knowledge and Information Systems*, Springer Verlag, Vol. 14, pp. 1–37, 2008.

[37] C. Xia, H. Lu, B. Chin Ooi and J. Hu. "Gorder: an efficient method for knn join processing". *In Proceedings of the 30th international conference on Very large data bases (VLDB'04)*, pp. 756–767, 2004.

[38] X. Xiong, M. F. Mokbel and W. G. Aref. "Sea-cnn: Scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases". In *Proceedings of the 21st IEEE International Conference on Data Engineering (ICDE'05)*, pp. 643–654, 2005.

[39] X. Yu, K. Q. Pu and N. Koudas. "Monitoring k-nearest neighbor queries over moving objects". In *Proceedings of the 21st IEEE International Conference on Data Engineering (ICDE'05)*, pp. 631–642, 2005.

[40] D. Zeinalipour-Yazti, P. Andreou, P. K. Chrysanthis and G. Samaras. "MINT Views: Materialized In-Network Top-k Views in Sensor Networks". In *Proceedings of the 8th International Conference on Mobile Data Management (MDM'07)*, pp. 182–189, 2007.

[41] D. Zeinalipour-Yazti, P. Andreou, P.K. Chrysanthis, G. Samaras and A. Pitsillides. "The MicroPulse Framework for Adaptive Waking Windows in Sensor Networks". In *Proceedings of the 8th International Conference on Mobile Data Management (MDM'07)*, pp. 351–355, 2007.

[42] J. Zhang, N. Mamoulis, D. Papadias and Y. Tao. "All-nearest-neighbors queries in spatial databases". *In Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM'04)*, pp. 297–306, 2004.

[43] Y. Zheng, L. Liu, L. Wang and X. Xie. "Learning transportation mode from raw gps data for geographic applications on the web". *In Proceedings of the 17th international conference on World Wide Web (WWW'08)*, pp. 247–256, 2008.