The Pennsylvania State University
The Graduate School

REVERSE NEAREST SOCIAL GROUP QUERY

A Thesis in
Computer Science and Engineering
by
Nitish Upreti

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

August 2015

The thesis of Nitish Upreti was reviewed and approved* by the following:

Wang Chien Lee
Associate Professor of Computer Science and Engineering
Thesis Advisor

Kamesh Madduri
Assistant Professor of Computer Science and Engineering
Committee Member

Lee Coraor
Associate Professor of Computer Science and Engineering
Director of Academic Affairs

---

*Signatures are on file in the Graduate School.

Abstract

Spatial data management issues, widely studied in the past few decades, have posed great challenges to researchers. A variety of efficient queries for calculating containment, intersection, nearest neighbors, reverse nearest neighbors among others have been proposed and studied. However, in recent years, increasing smartphone proliferation and web connectivity has lead to a new wave. The increasing popularity of modern LBSNs (Location Based Social Networks) such as Facebook, Foursquare, Periscope, Meetup among others that have never before existed. These services allow users to instantly share their location experiences around this activity with others. Location Based Service providers thus have a lot of user generated socio-spatial data that needs to be stored, processed and analyzed. We strongly believe that novel socio-spatial storage and querying systems are required to meet this need in the near future. In this work, we particularly focus on the challenge of quantifying influence in the socio-spatial domain. The idea is to find groups that satisfy spatial (proximity) and social (group cohesion) constraints and thus can be considered in the influence set for a given query point. To solve this problem, we propose the idea of **Reverse Nearest Social Group (RNSG) Query**. This socio-spatial query is strongly related to Reverse Nearest Neighbor (RNN) query that has been of much interest to spatial data researchers in the past. Unlike the Reverse Nearest Neighbor query that finds individuals having query point as the Nearest Neighbor, RNSG query finds all social groups that satisfy k-core constraint and have their farthest member (individual with maximum euclidean distance to the query point) as a Reverse Nearest Neighbor of the query point. Compared to conventional RNN queries, RNSGQ is much more challenging because of the added social constraint. To tackle this challenging research problem, we propose an algorithm **R-SAGE***(Rnn-based Social-Aware Group discovEry Algorithm)* to find all RNSG(s) efficiently. Experiments conducted on real world datasets show that the proposed R-SAGE algorithm outperforms the other baselines significantly. The proposed solution thus has huge implications on solving various problems in a variety of existing and upcoming domains such as Location Based Systems, Decision Support Systems, Social Marketing, Continuous Referral Systems to name a few. The efficient RNSGQ framework facilitates solving these novel socio-spatial problems and thus coping up with the recent challenges.

Table of Contents

List of Figures

List of Tables

Acknowledgments

My deepest gratitude to my advisor, Dr. Wang Chien Lee, for the time and effort he invested in me. Without him, I would have never come this far in my Master thesis. I really appreciate his encouragement and patience throughout the years, not to mention his excellent guidance and mentorship. I would also like to thank Dr. Chih-Ya Shen, Dr. Kamesh Madduri and The Pervasive Data Access (PDA) group at Penn State that provided me much needed input as I refined my research idea. I am deeply grateful for their support.

Chapter 1

Introduction

With the availability of affordable and portable smartphones, user generated data and content has exponentially increased in the past years. This massive growth in data generation needs a perfect amalgamation of software infrastructure, intelligent algorithms, networking capabilities and hardware resources in place to cope up with. Organizations are thus increasingly investing a large amount of resources to efficiently retrieve massive datasets on demand to drive business solutions and insights. Round the clock, Computer Scientists are working on solving hard problems; designing efficient solutions to slay the data complexity that arise with this tremendous scale.

Within the realm of our data rich world, spatial data representing information about physical location and shape of geometric objects is very popular. Also, in the past decade, Location Based Services (LBS) that offer a rich personal experience by making use of user's location and social graph data have become immensely popular. Internet is swamped with plethora of Location Based Services. Some of the popular genres (not exhaustive) are : recommending social activities, requesting a nearby business or service, locating friends, meeting new people, receiving location based notifications and map directions. For every genre there exists multiple applications that provide overlapping service. In this crowded space, some applications rise to attain massive scale and popularity. Boasting millions of active users, these largely populous networks usually vary significantly in the context and scope of their application but ubiquitously stem from a clever use of socio-spatial data. The new and exciting opportunities with these LBSNs are however accompanied by a variety of challenges that have never been realized before.

Korn and Muthukrishnan [16], described how Reverse Nearest Neighbor can intuitively quantify the notion of influence set for a query point. Thus in the past, Reverse Nearest neighbor queries have received considerable attention due to their importance in several applications involving notification systems, decision support systems, profile-based marketing among others. The problem of finding influence set occurs frequently in spatial domain and has wide implications. Adding to its significance is the fact that this problem can also be considered as a spatial primitive for solving even more complex queries. If we define proximity based on some similarity function, we can further use it in various generic scenarios. Some examples where an RNN query could be useful are: A restaurant wants to determine which customers are likely to visit a newly opened store? An RNN query can identify this important piece of information. Another example could be a scenario where a gaming company might send out emails to gamers that are highly likely to enjoy a new title based on game's genre and their existing likings.

However, as RNN focuses only on the spatial domain, it is unable to model scenarios where social groups are more relevant. This is feasible when users are simply interested in finding groups of individuals rather than a single individual under the influence. Therefore, we might want to use this information for finding social groups under the influence of a query point.

Combining the strengths of RNN and social groups can enable new applications in the domains discussed above. Similar to RNN query, a large number of radically different scenarios can be modeled in the socio-spatial domain using a generic similarity function and group definition. Thus efficient socio-spatial queries involving influence can be used to solve variety of problems that might not fit in a typical LBSN scenario, but nevertheless are fundamentally alike. To provide concrete examples, we extend some of originally documented examples for RNN queries and in addition provide more examples where a notion of social group provides novel use.

- **Building Decision Support Systems :** In a marketing application, adopting an outlet location over another based of geographical proximity of customers is highly critical. An important task is thus to find customer groups that will find a facility in a given location more convenient than their existing choices.

- **Building Continuous Notification Systems :** A referral service notifies users when the a service he or she is interested in changes. Combining RNN and social graphs, we can enhance LBSNs to send group invites to individuals who are likely to find a newly open business location(example: restaurant) appropriate in future. A mobile discount coupon can be pushed when the group visits the outlet.

- **Profile Based Marketing :** Companies willing to push new services to users based on a profile can utilize a social influence query. For example : AT&T launches a new service where all customers in a *family data plan*(social group) can upgrade to a higher data cap. Modeling customers and service into a socio-spatial domain, the social influence query will return user groups that are in the influence region i.e. will find the service useful.

- **Organizing *Meetups* for Social Groups :** Websites such as Meetup [3] provide an online social networking environment which facilitates offline meetings of people based on similar likes, interests and backgrounds. The success of a *meetup* group depends on how often individuals meet in the offline world. Inherently, knowing atleast some of the members in such offline interactions facilitates a good social environment and encourages participation. The location of event is another very important factor, as remote distances discourages people to attend. Thus, when organizing such a *meetup* given an event location, we want to send out invites to individuals belonging to cohesive social groups (to maintain familiarity) that are collectively closest to the rendezvous location.

It is quite essential to realize the subtle complexity of our problem statement. As we are working in a socio-spatial domain, designing any efficient solution needs to consider both these factor to derive an optimal solution. We cannot simply daisy chain a RNN algorithm with a community detection algorithm to obtain an optimal solution. Designing an efficient solution, needs fresh ingenuity as to balance tradeoffs in both the domains.

1.1 Overview of the Problem

While a lot of work has been done in the area of Reverse Nearest Neighbor Query, a little attention has been paid to idea of RNN in context of social groups. Our goal is thus focused on designing the Reversed Nearest Social Group Query (RNSGQ), which integrates the idea of RNN and a social cohesive group to capture the spatial influence and the spreading of influence among a set of individuals. Next, we list some of the challenges involved in deployment of a hypothetical system. We demonstrate that the problem is non-trivial and then formally capture our requirements in a mathematical statement.

1. **Accurate Representation of a real world scenario**: As to the best of our knowledge, there is no existing work that successfully quantifies influence on social groups, the quality of our work is dictated by how well we define the problem. There are two questions that are of particular interest : a) How do we accurately model a social group in our problem ? b) How do we choose a distance function that accurately quantifies influence ?

2. **Scalable Performance**: Scalability is one of the key attributes that defines the ingenuity of a solution. In context of LBSNs, which usually have a large number of active users, the requirement is even more pressing.

3. **Online Workloads with Dynamic Data** : We might encounter further challenges when the underlying dataset for RNSGQ is frequently changing. Also, it is quite plausible that our designed system is expected to respond to queries in an on-line environment where maintaining low query latency is critical.

Before presenting a mathematical formulation of the problem, we discuss an informal example to better convey the intuition behind our work. We hope that it will build a better understanding of the problem and the associated challenges for the readers.

Figure 1.1 is an extremely simple example of a scenario where RNSGQ can be useful. Let us assume that two triangular points S and S' are restaurants and the six circular points a, b, c, d, e and f are user locations in a LBSN. The dashed lines between points show the corresponding user friendship. With RNSGQ, we could look at the figure and ask : *Which groups are in the influence set for S or S'?*. If S or S' represent competing restaurants, given a choice, which ones

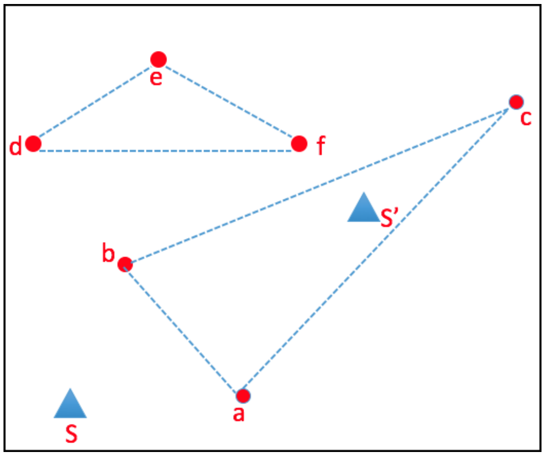


Figure 1.1. RNSGQ: A simple example

are to be more likely chosen by a group? For a marketer or a manager, having this information can be critical while making key business decisions in a competitive environment. However, it should be apparent that before we can answer this question, we need to address the potential ambiguity in the idea of *influence* and *social groups*. To answer this question without explaining what we imply by a *social group* or what defines *influence set* is improbable. Thus a definitive mathematical formulation of *"problem of influence"* and *"social group"* precisely conveying our intent is central to solving the problem.

1.2 Research Contribution and Organization

Thesis Contribution: We now highlight the achievements of the work. In summary, the key contributions of our research are:

1. Developing and understanding the notion of Reverse Nearest Social Groups. We propose an answer for the question : How do we concretely define the notion of influence for social groups? Our work thus successfully formalizes the idea of "influence sets" in context of social groups.
2. Providing an in-depth analysis of finding Reverse Nearest Social Groups for a large socio-spatial network and proposing *Social First* and *Spatial First* approaches for solving the problem.
3. Proposing a novel algorithm **R-SAGE***(Rnn-based Social-Aware Group discovEry Algorithm).*
4. Providing comprehensive theoretical and experimental analysis of proposed solutions.

In the remainder of this paper, we continue to describe the idea behind RNSGQ. We use a mathematical formulation to concretely define our problem in Chapter 2. This also help us build a fundamental understanding of Spatial Querying and Social Cohesiveness. The Chapter 3 further provides a comprehensive survey of Spatial Indexes, Spatial Queries, Social Measures and leads to a discussion on Socio Spatial queries. Next, Chapter 4 analyses the problem statement in depth and presents related findings. The learnings are then used to present *Social first* and *Spatial first* approaches for solving the problem. In Chapter 5, we build on *Spatial first* and *Social first* approaches to propose a solution referred to as R-SAGE*(Rnn-based Social-Aware Group discovEry Algorithm).* Theoretical and Experimental analysis is then presented in Chapter 6 and Chapter 7. We use an extensive comparison of performance to demonstrate the effectiveness of our finally proposed solution relative to the baselines. Finally, Chapter 8 summarizes the work and distills our learning.

Chapter 2

Problem Formulation

With a necessary background on socio-spatial queries, we formulate our research problem formally. Consider a social network $G = (V, E)$, where V is the set of clients and E is the set of undirected edges representing the friendships among these clients. In this network G, each $v \in V$ is also associated with a location l_v. Let S denote a set of server objects sharing the spatial plane, where each $s \in S$ is defined with a location l_s. We denote the spatial distance between two objects u, v as $DIST(u, v)$. A RNSGQ (q, k) comes with two parameters, where the first parameter $q \in S$ is the spatial query point and the second parameter k is the social constraint for clients in V. The RNSGQ(q, k) thus aims to find : The Set W of all Reverse Nearest Social Groups (RNSGs) for $q \in S$, such that 1) each $W_i \in W$ satisfies the k-core [9] constraint, i.e. each member in W_i has edges connecting to at least other k members in W_i. 2) For each W_i, the farthest actor $f \in W_i$ from q has q as its nearest neighbor i.e. $\exists f \in W_i$ such that $DIST(f, q) >= DIST(v_j, q) \wedge f$ is a RNN of $q, \forall v_j \in W_i - \{f\}$ (thus *minimizing the maximum distance(MINMAXDIST) to q)* and 3) each W_i is maximal i.e. there exists no $W_j \supseteq W_i$ such that W_j satisfies conditions 1) and 2).

To quantify spatial proximity of a group, there are several probable solutions. We can design spatial functions that could seek to *minimize*: *MINDIST* (minimum distance of group to query point), *AVGDIST* (average distance of group to query point) or *MAXDIST* (maximum distance of group to the query point). We claim that even though these functions are quite useful, they do not perfectly fit our scenario. This can be attributed to the fact that for the entire group to be in the influence set of the query point, none of these measure ensure that every individual in a group is in the influence region. To understand the idea further, let us look at a simple example in Figure 2.1. For this example, we will assume k value to be 1.

A group defined as B, C and D will be in the influence set of $S1$ if we use a spatial influence function based on *AVGDIST* (minimizing the average distance of group to query point). However, can we really consider D to be in the influence of $S1$? Intuitively, D is in close proximity to $S2$. Similarly, a group defined as F, G and H will be in the influence set of $S3$ if we use a spatial influence function based on *MINDIST* (minimizing the minimum distance of group to query point). However, a similar dilemma persists i.e. can we really consider H to be in the influence of $S3$ (instead of $S1$).Finally, a group defined as A, E and I will be in the influence set of $S4$ if we simply use *MAXDIST* (minimizing the minimum distance of group to query point) without introducing the idea of a *farthest actor*. Again, intuitively A is in the influence of $S2$ (instead of $S4$) by proximity.

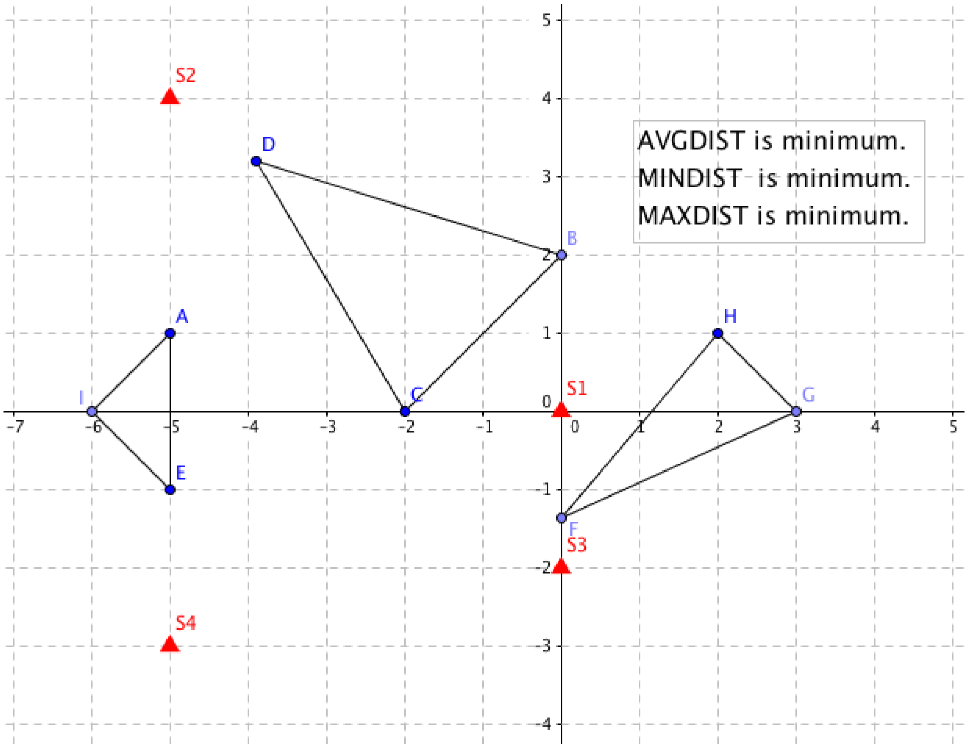


Figure 2.1. Distance Functions for quantifying Influence

The underlying theme is simple: Unlike *MINDIST*, *MAXDIST* and *AVGDIST* that can report groups with members that are unacceptably far away in the influence set, our influence criterion appropriately ensures an upper bound on the distance of groups to the query point. Moreover, after a careful analysis we realize that our influence criterion although similar to simple minimum *MAXDIST* criterion provides an even stricter constraint ensuring all members are in relative proximity.

Similarly, there are many existing popular social cohesive measures such as *diameter, clique, clubs, clans* [19] and *k-plex* [24]. Each group measure aims to capture different aspects of social closeness. However, extracting groups satisfying a specific social cohesive measures are usually hard problems. In contrast, k-core attracts much attention as it is a simple degree-based measure and can be done in polynomial time. It is thus an effective choice for describing social constraint. Moreover, while issuing the query, the parameter *"k"* can be tweaked by the user to strictly or loosely enforce social cohesiveness. It is interesting to note that, we can even take it to the extreme with *"k"* as 0. Our query is then just a spatial Reverse Nearest Neighbor query. The framework we design and propose is not restricted to a particular parameter value for *"k"* and efficiently handles any parameter.

Solution Outline: To design an effective framework for solving Reverse Nearest Social Group Queries, we present a solution framework that attacks the problem using several different strategies. We start the discussion by providing an in-depth analysis of the problem scenario. Insights based on server location and group location drawn from a *farthest actor* analysis are distilled for comprehending problem structure better. From this analysis, we move forward to propose two different approaches for solving our problem : 1) A *Spatial first* approach that prioritizes spatial pruning and 2) A *Social first* approach that in contrast emphasizes on social pruning.

To provide readers with a brief outline, the *spatial first approach* does an iterative exploration based on distance browsing and prioritize pruning based on distance by iteratively consider clients to find social groups. In contrast, the *social first approach* uses a pre-computed k-core to facilitate social pruning and uses divide and conquer for finding social groups (leveraging the fact that actors in different cores do not form a group).

The two approaches help us focus individually on the social and spatial component of the challenge. This serves as a baseline and paves the way for our final solution **R-SAGE***(Rnn-based Social-Aware Group discovEry Algorithm)*. The **R-SAGE** algorithm builds on existing RNN work in [28] and uses two-phased : 1) *discovery* and 2) *computation* to efficiently compute the solution set. Our final proposed algorithm thus greatly improves the scalability and performance as a solution when compared to the previous baselines.

Chapter 3

Related Work

In this chapter, we review all the essential concepts that are relevant to our problem of finding social groups in the influence set for a given query point. We begin our exploration in Section 3.1 by covering the fundamentals of data storage in context of relational databases. Further along, in Section 3.2 we describe some of the foundational data structures that facilitate efficient retrieval of spatial data. Next, Section 3.3 builds up on this previous background to discuss some of the most popular spatial data queries relevant to our work. In Section 3.4, we focus our attention on the problem of Core Decomposition for networks which lays the foundation of social groups in our problem. Finally, we discuss some of the recent work done in the area of multi-domain queries such as *socio-spatial* queries. This is due to the fact that our problem at hand, the RNSGQ is a socio-spatial query.

3.1 Fundamentals of Data Storage

To begin our literature survey, we take a look at some of the popular access methods used by all modern databases. The discussion will help the reader appreciate the science and engineering behind efficient data storage and retrieval. With this discussion, we also want to lay the foundation and trace the roots of modern indexing techniques. In a true sense, we are indeed *standing on the shoulders of giants.*

Development of efficient data structures like B+ Tree (the most ubiquitous variant of original B Tree index [10]) lay the foundation of efficient retrieval of data stored in modern filesystems. While our focus is on spatial data, B+ Tree are still relevant as they can be recognized as an efficient index for one dimensional data. The idea behind the B+ Tree index is to sort data keys based on some relative ordering property. The actual data objects are stored in the leaf node, while the internal nodes of the Tree store pointers for traversal. Each internal node is responsible for key ranges that fall inside some closed interval. A data object, given the corresponding key value can thus be retrieved by a method similar to that of a Binary-Search Tree. However, due to a high fanout degree (number of pointers to child nodes), I/O operations are significantly reduced. For example, Figure 3.1, shows a B+ Tree(degree 3) indexing data values 1 to 9. In addition to finding data value for a corresponding key, sequential scanning of the dataset is quickly feasible with pointers in leaf node. Although B+ Trees are highly efficient for fast retrieval of data objects, their inherent limitation of indexing only in a single dimensional domain makes them somewhat less relevant in spatial data storage. Researchers have tried to propose clever indexing techniques that allow B+ Tree to be used for multi dimensional data. One such index structure is **Space Filling Curves** [22]. The idea is to map every point in space to a unique number. A curve then defined a linear ordering of all points. The points are indexed with B+ Tree and can be used for efficient retrieval.

Next, in the upcoming Section 3.2, we will discuss organization techniques designed specifically for spatial domain. These techniques are optimized for multidimensional data and outperform the likes of B+ Tree.

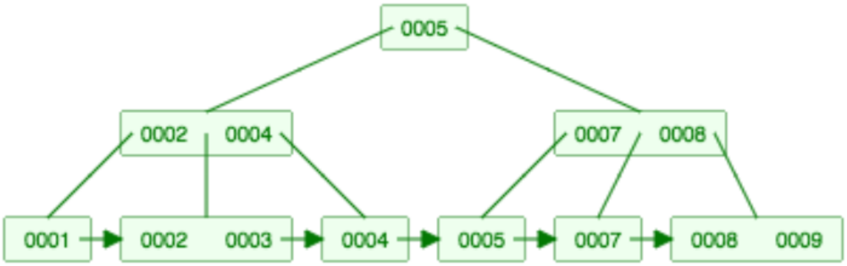


Figure 3.1. B+ Tree Example

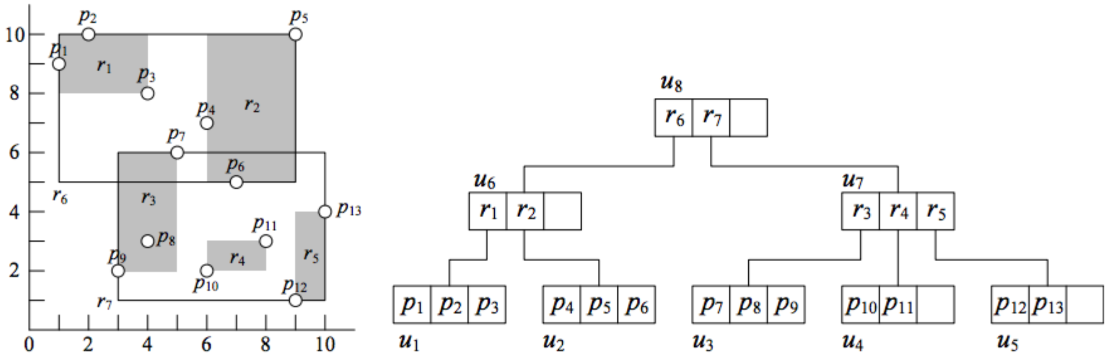
Hash Index deserver a quick mention as they can are used for equality comparisons in databases. These indexes, however do not support range queries and can be used only for simple key-value lookups. Another property of Hash Indexes require that whole keys be used to search for a qualified database row unlike B+ Trees where a leftmost prefix is sufficient [4].

3.2 Spatial Data Index

Spatial Index optimize queries on objects defined in a geometric space. These objects can be anything like points, lines and polygons and are essentially characterized by a location and geometric extent. Spatial queries can then be designed based on a variety of predicates such as relative location and geometric relationship between these objects. Some of the popular examples are: Spatial Selection, Intersection, Nearest Neighbor, Spatial Joins among others. The area is challenging, as dimensionality and extent complicates indexing, retrieving and updating data.

The central theme behind most of the popular Spatial Access Methods is to group nearby objects in space into disk pages. These pages are then organized in some hierarchical fashion in an index. Another common technique used is to define approximations of spatial objects (commonly referred to as Minimum Bounding Rectangles [31]) that can be used to filter unwanted results and reduce query execution costs. We will next cover some of the access methods used for indexing multidimensional data, particularly the R-Tree.

R-Tree [14] are among the most popular spatial access method used in commercial databases. Proposed by Antonin Guttman in 1984, they offer efficient storage with minimal update overhead. Organized in a multilevel hierarchical fashion where every node has a minimum and maximum capacity, objects are always stored at the lowest level of the Tree. A MBR approximates the smallest rectangle that contains all of the associated data points. As levels fill up, a recursively merging and splitting technique is used to solve the problem of overflow and underflow. The bounding boxes are used to decide whether or not to search inside a subtree. Most of the nodes in the tree are thus never read during a search. The idea behind R-Tree index is overall quite similar to that of B+ Tree discussed previously. However, unlike the B+ Tree structure where subsets of the data key range are disjoint, the MBRs associated with sibling R-Tree nodes frequently overlap. Consequently, a query might traverse multiple paths in the R-Tree to return meaningful results. Figure 3.2 demonstrates an example of hierarchical spatial organization with Figure 3.2(a) showing the data points along with corresponding MBRs and Figure 3.2(b) the R-Tree indexing this data. The Tree groups points p_1, p_2, p_3 in the leaf node u_1 which is itself a child of internal node u_6. This MBR r_1 stored contains all these nodes.



(a) Data and MBRs

(b) The structure

Figure 3.2. R-Tree Example

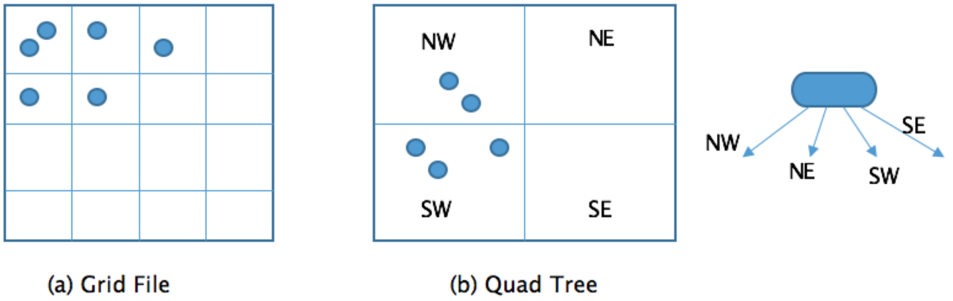
Since the work done by Guttman on R-Tree, multiple variants of the general R-Tree data-structure have been proposed. The most popular variants being the R+ Tree [25] and the R*-Tree [11]. The newer data structures defer from the original R-Tree work by optimizing on the heuristics used when grouping objects together in an MBR. The work done originally in R-Trees attempted to minimize MBR area using various heuristics that trade effectiveness for speed. In contrast, R*-Trees consider a variety of features when making decisions about how to group, split, and merge data objects. A revised node split algorithm and concept of forced reinsertion at node overflow is used to improve split heuristic performance. We will be using R*-Tree variant as the primary spatial indexing method in this work.

The database community has collaboratively designed many other multiple spatial indexes.

Some of the other popular indexes are Grid File [20], Quad Trees [13] and k-d Trees [12]. We briefly touch upon the central themes of these data structures.

Grid File : The grid file divides the entire domain space into multiple cells and uses a grid of n-dimensions(n representing number of keys to reference a single point). A grid directory is used to store mappings between cells and disk blocks similar to hashing. To answer a spatial query, cells that intersect with a query are determined. Next the grid directory are used to fetch data points corresponding to these cells.

Quad Trees : A quad Tree recursively divides space into four quadrants (Figure 3.3). The regions can have arbitrary shapes and need not be symmetric. The partitioning is done in a way that each leaf node has sufficient space for pointers to disk blocks. Similar to R-Trees, spatial queries are evaluated by finding blocks that overlap with query area. However, a major issue with Quad Tree is the fact that due to regular partitioning into four blocks, the utilization of leaf nodes is non-uniform.



(a) Grid File

(b) Quad Tree

Figure 3.3. Other Spatial Data Structures

K-d Trees : K-d Trees are a variant of binary Trees that provide for indexing in k dimensions. The space is recursively by choosing a coordinate as a basis. Points to the left in spatial domain are represented by the left subTree and points right of the hyperplane are represented by the right subTree.

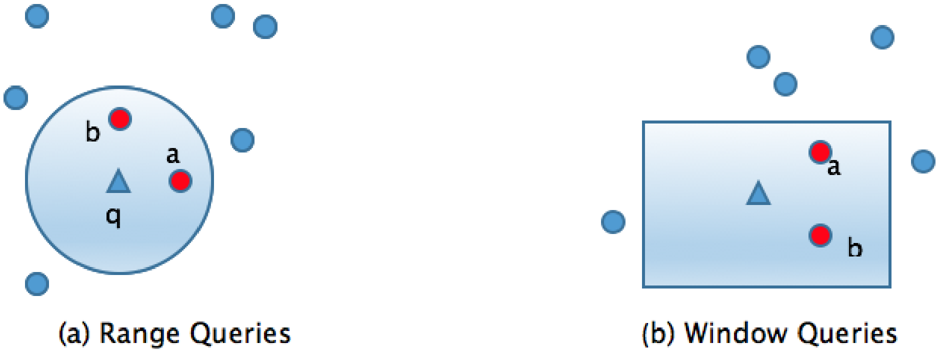
3.3 Spatial Data Query

This section discusses traditional popular spatial queries. Some of these include: spatial selection, nearest neighbor and reverse nearest neighbor. When discussing these queries, we will assume R-Tree or its variants being used for underlying indexing. Most of these spatial queries are now considered as building blocks for designing any modern complex query. To provide readers with some necessary context, we refer to query point and its competing counterparts as *servers* (providing a service requiring the notion of influence) and *clients* as individuals under the proposed influence.

We already informally discussed *Spatial Selection* queries in context to B+ Trees. We quickly review this query for sake of completeness. The Spatial Selection query involve returning objects in a plane with respect to a reference object and a predicate. In Spatial Selection MBRs stored in internal R-Tree nodes are iteratively tested against the query predicate. The child nodes whose parents qualify this MBR filtering step are further inspected. Most of the times, a predicate like intersection or containment is tested. The query finally retrieves all objects that qualify from the MBRs tested

Range queries return all objects in the dataset that lie within a specified area. The query is framed as searching from a central query point q along with supplemental range parameters given by a predicate P. The two most common categories include the range query and the window query.

Range Queries retrieve objects inside a circular region centered at the query point q with a radius r. Similarly *Window Queries* return all objects inside of a rectangular window around a query point q with extents given by length l and a height h. We visualize these ideas in Figure 3.4.



(a) Range Queries

(b) Window Queries

Figure 3.4. Spatial Range and Window Queries

In both of Figures 3.4(a) and (b), data points a and b satisfy the range and window query respectively. All other data points are simply filtered out by the predicate.

Some potential applications for these queries are :

- A restaurant wants to find all individuals within a range. A range query can accomplish this task.
- A user might want to explore all monuments in a tourist city area. Here the user is interested in a region that can be modeled as a spatial window.

To process Range and Window queries, the key idea is to search all the child nodes of a R-Tree whose MBR overlap with the queried search region. When traversing the leaf node satisfying the property, the algorithm will include all the objects in the result. This can be done in a BFS(Breadth First Search) or a DFS(Depth First Search) fashion.

Another fundamental spatial query is the Nearest Neighbor(NN) query that retrieves objects closest to a given query point q in the spatial domain. This problem is extremely important as most of the other Spatial data queries use it as a foundation when designing more complicated spatial queries. Researchers refer to the problem as finding the k-Nearest Neighbor of a query point. Here the parameter k implies that we return k closest objects to the query point. Finding Nearest Neighbor is can then be considered a special case where $k = 1$.

Let us assume that we have a query point q and we consider all other data points in a set C. The k-Nearest Neighbor query returns a set c, where $|q, c| \leq |q, c'|$, $\forall c' \in C$. Several techniques have been proposed in the past literature for processing Nearest Neighbor information. Most of the efficient techniques assume that the data points are indexed by an R or R*-Tree. These efficient solutions employ a technique popularly referred to as *distance browsing* [15] in the literature. The idea behind this technique is to processes points in order of minimum distance from the query point q. The algorithm makes use of a priority queue to examine data points in an increasing order of Euclidean distance. It is intuitive that the first k objects encountered are part of the result set.

We provide a visualization for Nearest Neighbor(1-NN) query in Figure 3.5.

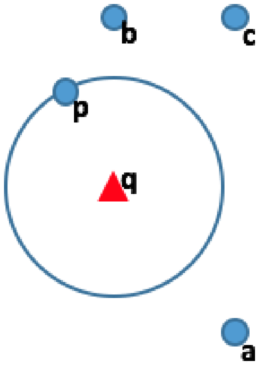


Figure 3.5. Nearest Neighbor Query

The point p is the Nearest Neighbor and discovered first by the distance browsing query with q as query point. Other points a, b and c are not discovered.

The RNN or Reverse Nearest Neighbor query is highly critical as it closely resembles our problem of finding influence set consisting of social groups. In contrast, the RNN problem instead focuses only on the spatial domain and aims to find individual objects that have the query point as Nearest Neighbor. Similar to the Nearest Neighbor problem, we have a general problem defined as RkNN and a more specific variant RNN where the value for k is 1.

Let us now define the RNN query for a point q. The RNN result set consists of a set c such that all points in this set are closer to query point q than any other data point in C. We can thus define the solution set having all elements $c \in C$ with $|c, q| \leq |c, c'|$, $\forall c' \in C$. The RNN query thus identifies all objects that have the query point closest by Euclidean distance. Unlike NN query type, the cardinality of an RNN result set varies. It is also possibly for the result set to be empty(where none of the points have query point as Nearest Neighbor). It is also important to note that we present a monochromatic (clients and servers belonging to the same set) view of RNN query in the problem definition. In addition to this definition, a similar bi-chromatic variant can also be formulated where clients and servers belong to different sets.

We provide a visualization for Reverse Nearest Neighbor(Rk-NN) query in Figure 3.6. The point p and r have q as their Nearest Neighbor. Hence they are the Reverse Nearest Neighbor for q.

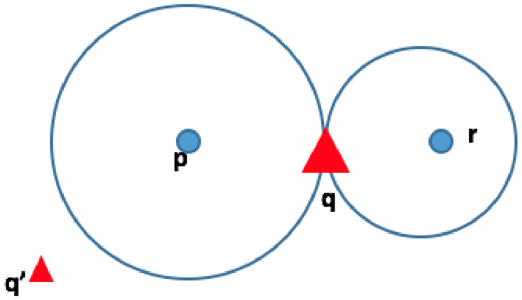


Figure 3.6. Reverse Nearest Neighbor Query

As we discussed in Chapter 1, we can use the RNN query for solving scenarios involving influence in spatial domain. The granularity of influence set are individual point instead of social groups as we are dealing with just the spatial domain.

A lot of methods in the past have been proposed to compute RNN queries. Some of the past influential works are [17], [27], [29], [32] and [28]. As we will present to the readers in upcoming chapters, the work done by Stanoi et al. in [28] is most relevant to our current problem. Our proposed solution R-SAGE builds up on this existing work to solve for Reverse Nearest Social Group Query.

3.4 Core Decomposition of Graphs

Another idea that is central to our problem of finding Reverse Nearest Social groups is that of k-core decomposition. This idea of a *core* was originally proposed by Seidman in [23]. The problem defined a core as a *cohesive subgroup* of individuals within a network where a *cohesive subgroup* consists of individuals with strong direct relationship. Here is a precise definition as originally formulated in [23]: Let $G = (V, L)$ be a graph. V is the set of vertices and L is the set of lines (edges or arcs). We will denote $n = |V|$ and $m = |L|$. A subgraph $H_k = (W, L|W)$ induced by the set W is a k-core or a core of order $k \iff \forall v \in W : deg_H(v) \geq k$, and H_k is the maximum subgraph with this property. The core of maximum order is also referred as the main core. The core number of vertex v is the highest order of a core that contains this vertex.

As we discussed in Chapter 1, k-core is central to our problem formulation as we define a social group as an entity where every individual is atleast familiar to k other people in the group.

This is a natural way to model our problem, as we can vary the value of k to dictate group cohesion. A higher value for k will dictate a tightly knitted group whereas a lower k value would promote a set of loosely connected individuals.

Figure 3.7 demonstrate the example of cores in a network. In this example, point I has a core number of zero. Point G has a core value of one and F, H get a value of two. Rest of the points A,B,C,D and E lie in a core of value three.

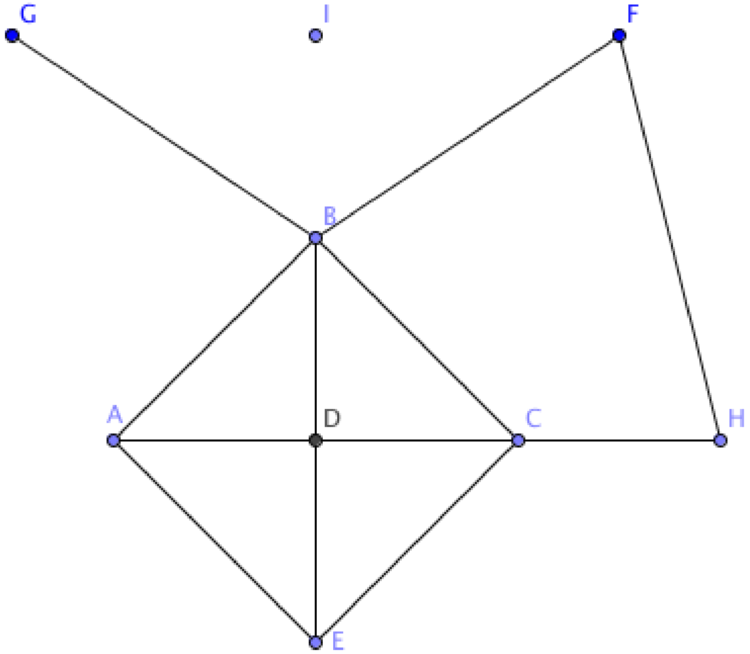


Figure 3.7. k-Core example

The problem of core decomposition received little attention until Vladimir et al. in [9] proposed an efficient solution for calculating maximal core in a graph. The work uses a recursively technique to delete all vertices and edges of degree less than k, resulting in a remaining graph which is a k-core. In the upcoming chapters, we will use k-core based constraint to model social groups. However, it is important to note that in addition to satisfying the social property, when calculating influence set in RNSGQ, it is also critical to ensure that all individuals in k-core also satisfy the spatial proximity constraint.

3.5 Multi domain query problems

We can classify the problem of finding Reverse Nearest Social Group for a query point as a *multi-domain* query. This is due to the fact that it involves working with spatial as well as social domain. In this section, we quickly mention some of the past research conducted in the socio-spatial domain. This will give readers a flavor of problems that involve juggling multiple domains. As we discussed previously, these problems have received considerable attention in the past half a decade because of a boom in the area of Location Based Social Networks. Our research is a further stepping stone in this direction.

One of the foundational work in Socio-Spatial group query is by Yang et al. [33]. The research work titled *On socio-spatial group query for location-based social networks* aims to enhance organizing impromptu activity by issuing invitations in accordance with the locations of attendees and the social relationship among them. The work is relevant to us as it involves a similar tradeoff in spatial-social domain when selecting a group of attendees close to a rally point while ensuring that they have a good social relationship. The problem is formulated in [33] as: Given a social graph $G = (V, E)$, where each vertex $v \in V$ is a candidate attendee with a location l_v, and any two mutually acquainted vertices u and v are connected by an edge $e_{u,v}$. A Socio-Spatial Group Query $SSGQ(p, q, k)$ finds a set F of p vertices from G where the total spatial distance from the vertices in F to the rally point q, i.e., $v \in Fd(v, q)$ is minimized and each vertex on average can share no edge with at most k other vertices in F.

To solve this problem, authors propose an approach titled as *SSGSelect* that uses concepts defined as *Distance Ordering* (ordering points by distance), *Socio-Spatial Ordering*(ordering points both by socio-spatial properties), *Distance Pruning*(pruning points not satisfying spatial constraint), and *Familiarity Pruning*(pruning points not satisfying social constraint) for finding an optimal solution. The authors demonstrate that *SSGSelect* is highly efficient and outperforms manual coordination and other baselines significantly.

Another significant work in Socio-Spatial domain is by Nikos et al. [7] in their work: *A general framework for geo-social query processing.* This work proposes a general framework to tackle processing Socio-Spatial queries using data management techniques and novel algorithmic design. The work demonstrates how a complex socio-spatial query can be executed by combination of primitive queries issued to the social and geographical modules that allows perfect separation of concerns. The work proposes two GeoSN queries *Range Friends* (finding friends of a user within a given range) and *Nearest Friends* (finding the nearest friends of a user to a given query point). In addition to *Range Friends* and *Nearest Friends*, the work proposes a *Nearest Star Group* query which similar to our proposed problem, and aims to find a star subgraph of the social network that minimizes aggregate (euclidean) distance to query point. In the problem formulation, a star subgraph implies a social group where four users are connected by a star's center i.e. a common friend. We would like to point out to the readers that the proposed query *Nearest Star Group* is fairly limited in comparison to Reverse Nearest Social Group Query as it enforces a very strict restriction on the group structure.

Another substantial work in Socio-Spatial group queries has been proposed by Shen et al. in [26] titled *Socio-Spatial Group Queries for Impromptu Activity Planning.* This work introduces the idea of *Multiple Rally-Point Social Spatial Group Query*(MRGQ), to find an appropriate activity location for a group of attendees in proximity with tight social relationships. An MRGQ includes four parameters, p(size of the answer group), Q(list of activity locations), k(familiarity constraint) and t(spatial radius) of the query. The problem sets to find a set $F \subseteq V$ where

$|F| = p$ and minimize the total spatial distance from F to q, i.e., $\sum_{v \in F} d_{v,q}$, where $d_{v,q} \leq t$, $\forall v \in F$, and unfamiliar $(v, F) \leq k^3$, $\forall v \in F$. The authors demonstrate that the problem is NP-hard but can be solved efficiently for small number of attendees. A solution based on Integer Linear Programming optimization model is proposed, followed by a algorithm which uses search space exploration and pruning to reduce runtime for finding the optimal solution.

Summarizing, in this chapter, we presented research in relational data storage and retrieval that led to the design of a variety of efficient access methods. We then discussed the evolution of access methods specialized for spatial data. This was followed by studying queries in spatial and then social-spatial domain to trace the genesis of Reverse Nearest Social Group Query.

Finally, concluding our discussion, we can suggest that although many of the discussed socio-spatial queries differ significantly in terms of their objectives, there is a common theme that transcends all of them. All Socio-Spatial queries present unique challenges for algorithm designers as they involve exercising innovative ideas in multiple domains for efficient pruning and thus finding optimal / efficient solutions. As Reverse Nearest Social Group Query is a Socio-Spatial query, we will face similar challenges when designing an efficient solution.

Chapter 4

RNSGQ Processing

To lay a foundation, we introduce the computational framework necessary for supporting RNSGQ and state any underlying assumptions. We then provide a comprehensive analysis to further delve in and understand our problem throughly. As we will demonstrate, this analysis helps us to understand both the social and spatial aspects of approaching the problem. Armed with this knowledge, we then present two baseline solutions based on these approaches. The chapter concludes with us reinforcing the lessons learnt and paving way for designing an efficient solution. The work thus serves as a groundwork to address challenges involved due to tradeoff in spatial and social domains.

Although the query is designed to work with complex LBSN storing a large amount of data, we make minimal set of assumptions when designing a system model for RNSG query. In Chapter 3, we presented an overview of Spatial Data Storage. For the analysis and baseline solutions we present in this chapter, we will assume access to an R*-Tree data structure and support for fundamental queries like Distance Browsing [15] and Nearest Neighbor Queries [21]. We make no assumptions as to if the data resides in disk or memory. The R*-Tree can be chosen to implement as an in-memory or disk resident data structure.

4.1 Problem Analysis

To answer this problem and analyze possible solution strategies, we will again use a simple example to present some very important observations that will help us shape the problem solution. With this suitable background, we then proceed to define the plausible problem solutions.

Consider Figure 4.1 with a set of Server points $S1, S2, S3$ and client points $v1, v2, v3$ and $v4$ where $S1$ is our query point. Let us assume that we want to find all the RNSG(s) that satisfy the 1-core constraint. For Figure 4.1, the set of points $v1, v2, v3, v4$ together, or a subset can be a potential Reverse Nearest Social Group for S1.

To study the problem further, we first investigate the concept of a *farthest actor*. For a given

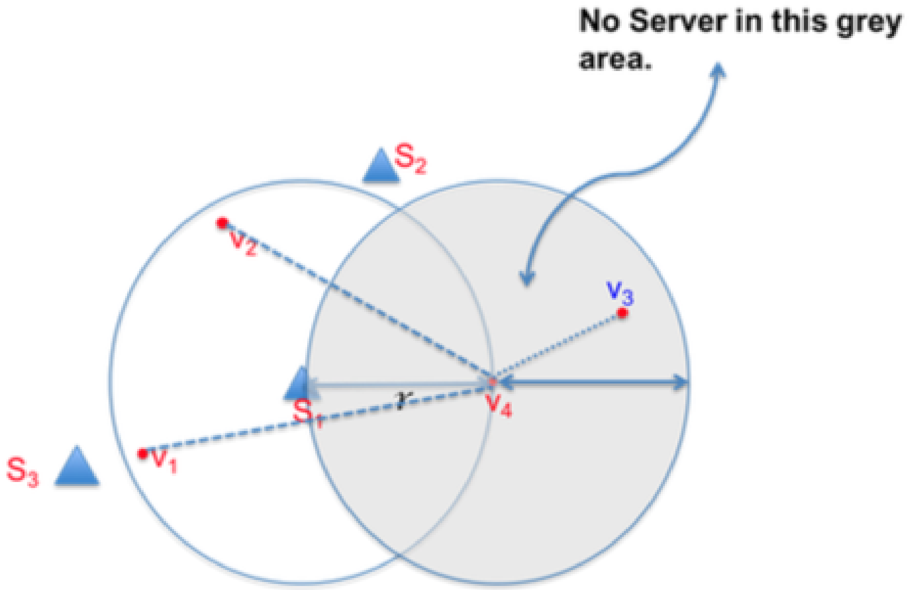


Figure 4.1. RNSGQ Analysis

query point q, we define *farthest actor* as a client v_i that is a Reverse Nearest Neighbor (RNN) for q and is also situated at a distance (euclidean) greater to q than any other RNN client $v_j \in V - v_i$ that are part of the same k-core.

In Figure 4.1, we can prove that $v4$ is a *farthest actor*. Comparing $v4$ to other client points, $v2$ is not a RNN of S_1, $v1$ is a RNN but is situated at a distance less than $v4$. In contrast, the point $v3$ is situated at a farther distance but does not have S_1 as a RNN. Thus $v4$ is RNN in the core situated at a maximum distance(satisfying all criteria discussed above). As we will demonstrate later, finding all *farthest actor* given a query point is a non-trivial problem and one of the key challenges that we need to overcome for finding RNSGs efficiently. To contribute further insights, understand *farthest actor* and quantify influence, we now propose the following statements:

Influence Proposition 1: There exist no server within radius of distance r for the actor $v4$ (denoted as $d(v4, r)$).

Proof: Let us assume that there does exist a server S within radius of distance r for the actor $v4$. If this is true then server S is the Nearest Neighbor of $v4$ (by definition). However, this contradicts from our initial assumption of S_1 being the Nearest Neighbor of $v4$. Therefore we can conclude that there exist no servers within $d(v4, r)$ (marked as gray in Figure 4.1).

Influence Lemma: For a given query point q, if a client v at a distance r is a *farthest actor* for q, then a group W_i consisting of v and all v_i that lie within $d(q, r)$ and are part of the same k-core form a Reverse Nearest Social Group for q.

Proof: From Influence Proposition 1, we can safely conclude that any server $q^{'}$ in set $S - q$

will have a distance more than r to v. Thus a group W_i consisting of v and all v_i within $d(q,r)$ will have *MAXDIST(Maximum Distance)* $(Wi, q) = r$ but *MAXDIST* $(Wi, q) > r$. We can safely conclude that *MAXDIST* $(Wi, q) < MAXDIST(Wi, q)$, as the *MAXDIST* is being dictated by the farthest actor v. In our example Figure 4.1, *MAXDIST* $(\{v1, v2, v4\}, S1) <=$ *MAXDIST*$(\{v1, v2, v4\}, S - \{S1\})$. As the group is a nearest neighbor according to *MAXDIST*, *farthest actor* and is also assumed to be satisfying the core constraint, we can qualify it as a RNSG.

We now take a step back and ask For each potential group and a given query point, how can we find all the farthest RNN actors? The key to solving the RNSG query is thus to effectively examine and identify these client points. With that in place, we could then for each potential group find all the points $vi \in V$ that lie at a distance less than or equal to $d(q,r)$ and along with the *farthest actor* v satisfy the social constraints.

As the problem of finding RNSG(s) for a query point involves finding data points that satisfy the spatial as well as the social constraints, an efficient algorithm needs to effectively consider the distinct spatial and social layers. If we simply choose to prioritize one over another, we will not return a correct result set. With this necessary background, we now propose two basic processing approaches to solve this problem.

The spatial approach is based on an iterative space expanding technique based on distance browsing, where we prioritize pruning based on distance and iteratively consider clients to find RNSG.

To guide an efficient exploration of the search space, at each iteration, we want to incrementally consider client according to their distance from the query point S. For achieving this, we leverage the distance browsing technique in Spatial index R*-Tree. The distance browsing technique can be efficiently used to discover points in ascending or descending order of distance from a query point.

Let us take the example of Figure 4.2 to understand how we want to approach finding RNSG(s) for query point S. Remember the idea for us is to find all the possible farthest RNN actor points that will help us reach our end goal. To begin with, we use distance browsing, to discover the point 1 at first. Once we have a newly discovered point, we would want to check if this point 1 is a Reverse Nearest Neighbor for S and also a potential *farthest actor*? We can easily check the former condition by issuing a Nearest Neighbor query for 1 using our R*-Tree Spatial index. However, as at the moment, we do not know what points lie further? Thus a key challenge is to figure out if the point is indeed a *farthest actor*? How do we resolve this dilemma?

To answer this question, we need to discover all possible points that could potentially act as the *farthest actor* in the connected component that contains 1. To do this, we propose a Breadth First Search traversal to explore all the points reachable from 1 and maintain a list of all these points sorted by decreasing distance to S i.e. $\{6, 5, 2, 4, 3\}$. Now, to discover a maximal group, we iterate from farthest point in the list and check is the point satisfies the RNN constraint. In Figure 4.2, point 6 and 5 fail to satisfy this property as they have $S3$ and $S2$ as their respective Nearest Neighbors. Point 2 is a RNN point but does not satisfy the core constraint as its degree

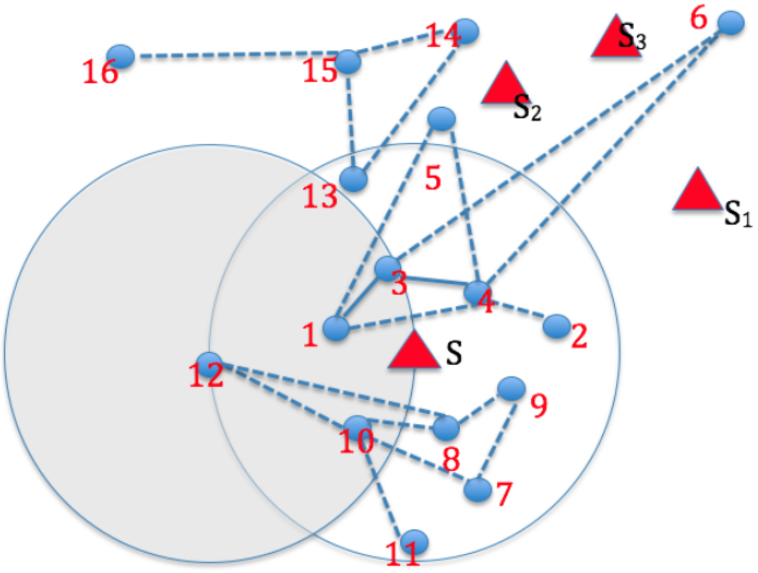


Figure 4.2. Socio Spatial Example

is 1.

Moving ahead and checking client 4, the point satisfies the *farthest actor* criteria. At this point, we are halfway through our goal, we have found the *farthest actor* point and can now look for a potential RNSG for S. Thus, we propose to perform a social pruning (running k-core algorithm) on the rest of subset $\{4, 3, 1\}$ dictated by Lemma 1. As these points satisfy the spatial and social constraint, we report the set as one of our solution.

Similarly, we discover 9 as the next closest point, and further discover all the connected points $12, 11, 7, 10, 8, 9$. The *farthest actor* which is a RNN of S is 12. After performing the social pruning on subset, we filter out 11 and we have $12, 7, 10, 8, 9$ as our resultant group.

Finally, for the next group discovered starting from 13 i.e. $\{16, 14, 15, 13\}$ none of the points satisfy the *farthest actor* constraint. Hence we reject all the data points without even checking for social constraints. Algorithm 1 compiles all of the steps clearly.

We now discuss the other school of thought in solving the RNSG problem. As discussed previously, in this approach, we use a pre-computed k-core to facilitate social pruning and use a technique similar to *divide and conquer* for solving the problem. We refer to this approach as *divide and conquer* as it stems from the idea that actors in different cores do not form a social group.

Let us use Figure 4.2, to understand the Social approach for solving the problem. We start by running a k-core algorithm on graph G to prune all elements that do not satisfy the k-core constraints. In Figure 4.2, as our constraint require a 2-core, we prune $\{16, 2, 11\}$ as they have a core value of 1. The idea for this approach is to next traverse each core to find the maximal social groups. We propose to iterate on every core actor in descending order of distance from S. This is due to the fact that one of our preliminary goals is to find the *farthest actor* for S that satisfies the

Algorithm 1 Spatial First Approach for Reverse Nearest Social Group Query

```
$V \leftarrow$ Set of all clients
$S \leftarrow$ Query Point Location
for all client points $v \in V$ in ascending order of distance from $S$ do
    if NotProcessed(v) then
        Do a BFS starting at v to :
        ⇒ Find the entire connected component.
        ⇒ Maintain list $L$ of clients in the connected component in descending order of distance.
        for all $l \in L$ do            // To find the farthest actor with S as its Nearest Neighbor
            if (Degree(l) < k || NearestNeighbor(l) != S ) then
                Ignore and mark as processed.            // Not a part of our result set
            else
                Perform k-core computation for elements in $L[l..(l.length)]$.
                if (IsKCore($L[l..(l.length)]$)) then
                    Return the core as a solution.
                end if
                Mark all points in L as processed.
            end if
        end for
    end if
end for
```

RNN constraint. Iterating in descending order of distance reduces any un-necessary computation and is more intuitive way to solve the problem. To iterate over clients in descending order of distance, we can use a variant of distance browsing approaches to consider clients in descending order. For Figure 4.2, we start with the core $C1 = \{6, 5, 4, 3, 1\}$ and inclemently check to find the *farthest actor* satisfying the RNN constraint. For this core, the farthest RNN actor is 4 giving us a subset $\{4, 3, 1\}$. As we are considering a subset of a core, we again need to check again if the subset satisfies the core constraints. To do this we run a k-core algorithm again to check if $4, 3, 1$ make a 2-core. As they satisfy the group constraints, they are one of the RNSG for S. Next we consider core $\{12, 7, 10, 8, 9\}$, as 12 does satisfy the RNN constraint, we can simply result this as one of our answers. Finally we consider the core $\{14, 15, 13\}$ but as none of the point satisfy the RNN constraint, we skip the clients. We now summarize the Social First approach in Algorithm 2.

Algorithm 2 Social First Approach for Reverse Nearest Social Group Query

```
$C \leftarrow$ Set of all clients
$S \leftarrow$ Query Point Location
Execute k-core decomposition on the entire graph to retrieve a core $V$ from all points in $C$.
while (NotEmpty($V$)) do            // Consider every $v \in V$ in descending order of distance
    if (NearestNeighbor(v) != $S$) then
        Ignore and delete $v$ from $V$.
    else            // $v$ is a NN and is assumed to be situated at a distance $r$ from $S$
        Find and return the sub-core $K$ consisting of $v$ and points in $d(S, r)$.
        Delete all elements in $K$ from $V$.
    end if
end while
```

We discussed both the Spatial First and Social First approach in the last section. The individual algorithms give us some insights on their respective advantages and disadvantages. However some of the subtle nuances of algorithms might remain hidden unless pondered on. In this section, we thus explicitly discuss the merits and de-merits for both approaches.

The Spatial First approach provides excellent distance browsing capabilities that allow us to explore groups nearby the query point. As all the reverse nearest social groups are expected to minimize the distance to query point, this is a huge advantage. The advantages discussed earlier are also the reason behind some of the major shortcomings for the approach. As the algorithm aims to explore groups iteratively according to increasing distance, this makes identifying *farthest actors* difficult. Moreover, using BFS to find other group members causes huge performance penalties in densely connected social graphs with few very large connected components.

The Social first approach as discussed in last section, provides an excellent *divide and conquer* strategy for simply pruning a large number of clients based on their social connectivity. This provides a substantial advantage as k-core computation has a linear runtime and scales wells for large number of clients. Finding *farthest actors* is relatively straight forward too as we can consider every core in decreasing order of distance from query point. Facilitating spatial pruning using the Social Pruning approach is although quite cumbersome. Once a core is identified, in order to compute *farthest actors*, a large number of Nearest Neighbor queries have to be issued. This can be very expensive. Moreover, to check if k-core subgroups satisfy core constraints, we have to repeatedly check for social constraints. Thus performing k-core decomposition repeatedly. This can be expensive.

Analyzing merits and demerits of the baselines help us in determine recurring challenges in designing an efficient solution. It is apparent from the past few sections that given a query point for computing RNSG(s) effectively, we should be able to effectively compute: Given a set of nodes, do they satisfy spatial constraints? and Given a set of nodes (that satisfy spatial constraints as dictated by farthest actor), do they also satisfy k-core constraints?

As repeated execution of these steps lie at the crux of our algorithm, a rough breakdown of the cost of our query will give us further insights. It is important to note that these steps are a part of both Spatial First as well as Social First approach. The total cost of baseline algorithms can thus be broken down into the following individual components: 1) Finding Farthest Actors which is very costly as we were issuing a large number of Nearest Neighbor queries. This is also due to the fact that social network graph consist of few very large connected components, and the algorithms end up traversing all nodes of a large graph. Thus to find all other elements, we need to visit the entire connected component. 2) Finding all other plausible group members (the ones reachable from farthest actor and within a distance bounded by the farthest point) which is again costly. 3) Checking if these nodes together satisfy Social Constraints which is costly too. This is due to the fact that for checking social cohesiveness, we need to check for k-core constraint. When done repeatedly, it is computationally expensive.

Recalling our past discussions of baselines, an interesting observation is the fact that: our solution revolves around finding *farthest actors*. The *farthest actors* are interestingly a subset of

Reverse Nearest Neighbors for the query point. However, we do not directly use any ideas from research in efficiently calculating Reverse Nearest Neighbors to calculate Reverse Nearest Social Groups. The baselines instead use a more primitive repeated execution of Nearest Neighbor queries. This is because we are trying to approach the research problem distinctively in Spatial and Social domain. In the next chapter, we will build on the observation of calculating RNN points to efficiently calculate Reverse Nearest Social Groups for a given point.

Chapter 5

Improving RNSGQ Computation

In this chapter, we propose an efficient algorithm defined as R-SAGE*(Rnn-based Social-Aware Group discovEry Algorithm)* to compute all Reverse Nearest Social Groups. As a reminder to our readers, we list the key challenges for efficiently doing so. Going ahead, our proposed solution will address all these challenges by combining pruning in spatial and social domain. The challenges are: 1) Finding all farthest actor effectively, 2) Quickly locating the rest of qualifying group members as dictated by Lemma 1 and 3) Efficiently checking if a set of clients satisfy the core constraints.

To communicate our ideas with the reader, a narrative style similar to past few chapters is used. We follow through an example to explain our vision for the problem solution. The algorithm R-SAGE is further divided into two distinct phase to convey the major learnings: 1) A discovery phase for finding relevant server and client data points. In this phase, to keep track of clients socio-spatial properties, we propose a spatially aware disjoint set data-structure. This data-structure keeps track of clients according to the connected components they lie in and their respective distances to the query point. 2) Once the discovery phase terminates, a computation phase is employed for calculating the exact social groups.

5.1 Calculating Reverse Nearest Social Groups with R-SAGE

Discovery Phase: The key intuition behind this approach is to estimate a region where the social groups in the influence set will lie. Along with the spatial estimation, we also want to discover the social graph for each qualifying node and temporarily index them in a fashion so as to aid in checking core constraints quickly.

Following key steps are performed in this phase:

1. Calculating a bounded region for the query point where all the social groups and competing servers will lie.

2. Retrieving all Servers and Clients in the region.
3. Indexing clients in order of increasing distance and by social connectivity to facilitate quick checking of social constraints in the next phase.

RNN Background With these goals in mind, we build on the state of the art algorithm proposed in [28] that calculates all Reverse Nearest Neighbors for a query point effectively. Discussed in length at Chapter 5, *farthest actor* is among one of the RNN point, thus this technique shares high relevance to our work. We adapt and improve on the technique to efficiently compute social groups as defined in the scope of our work rather than individual client points. To provide our readers with the necessary background, let us first quickly discuss the major ideas of the previous work. We strongly recommend the readers to refer to this work for an in-depth look. After this brief discussion, while discussing RNSGQ, we will provide a step by step example to further build on the idea to solve for our problem statement. The algorithm proposed by Stanoi et al. [28] can be summarized as:

1. Finding an Approximate region *Approx(q)* where RNN points lie. This region will contain an influence set with either the exact same points as defined by a precise RNN region or a Superset of these points i.e. including some incorrect points that are not in the influence set.

   Following Figure 5.1, to find this approximate region, four Nearest Neighbor servers (S1, S2, S3 and S4) in each quadrants are discovered. Since the NN points are closer to Q, they restrict the influence region for Q. However, this region is still an approximation as we are missing out on points like S5 which is not a Nearest Neighbor but still restricts the influence set. An approximate influence region defined as $\{A1, A2, A3, A4\}$ is obtained by calculating the intersection of bisectors of point $|QS1|, |QS2|, |QS3|$ and $|QS4|$.

2. Next, using clever analysis and by simply looking into the approximate region, another region *Refined(q)* is calculated. This region is theoretically guaranteed to consist of all other competing servers that can potentially affect the influence region of query point. Once the region is defined, all the servers in this region are retrieved.

   The key insight to finding *Refined(q)*, comes from the fact that any Server Point whose bisector (with query point) intersects with the lines defined by A1, A2, A3, A4 constricts the influence region further. Figure 5.2 defines the four circles with diameter equaling the distance between query point Q and $\{A1, A2, A3, A4\}$ respectively. Figure 5.3 builds on the circles constructed to define an MBR consisting of all four circles. Stanoi et al. prove that if bisector of the midpoint for line segment joining an arbitrary server S and q lies outside the MBR of Circles (as denoted by M1, M2, M3 and M4 in Figure 5.3) then the point cannot constraint influence region for q.

   Assuming the coordinate of a circle i be defined as $c_i x$ and $c_i y$, the authors propose that the MBR dimensions can be further calculated as:

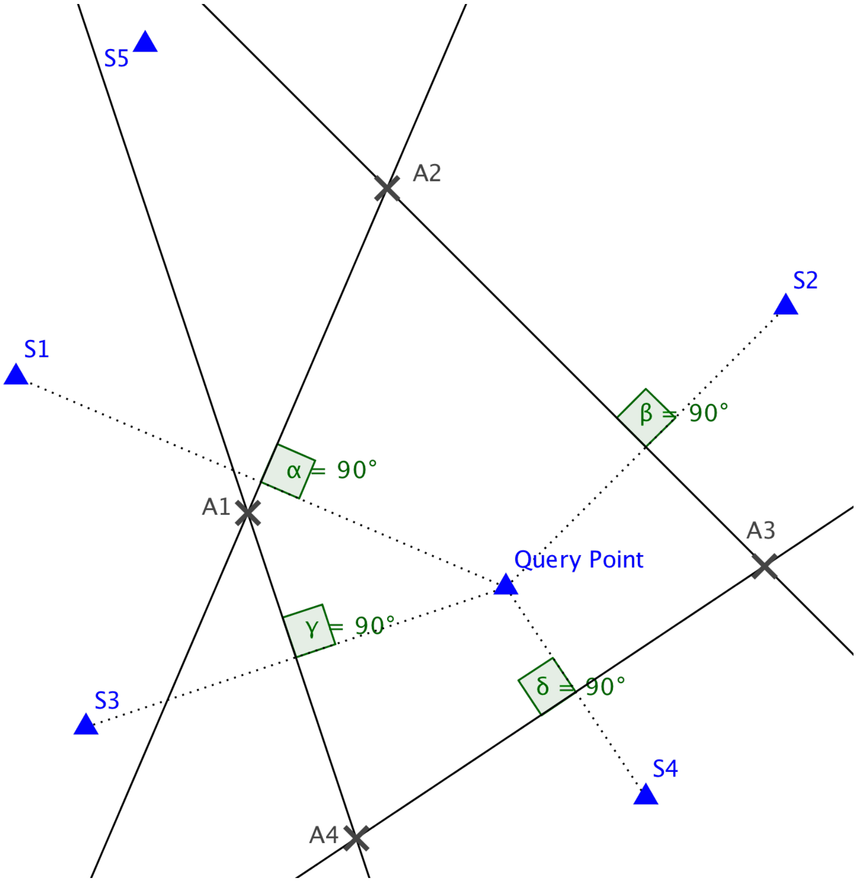


Figure 5.1. Server with Nearest Neighbors in all four Quadrants

$$Low(x) = min((c_1x - r_1), (c_2x - r_2)...(c_ix - r_i))$$
$$Low(y) = min((c_1y - r_1), (c_2y - r_2)...(c_iy - r_i))$$
$$High(x) = max((c_1x + r_1), (c_2x + r_2)...(c_ix + r_i))$$
$$High(y) = max((c_1y + r_1), (c_2y + r_2)...(c_iy + r_i))$$

The area defined as *Refined(q)* is derived the MBR as:

$$Refined(q) = 2 * low_x - q_x, 2 * high_x - q_x, 2 * low_y - q_y, 2 * high_y - q_y$$

The complete Proof for the work is presented in [28].

3. Finally all RNN points in the approximate region are retrieved and verified to be in the influence set by comparing the distance to competing server points retrieved further in *Refined(q)* in Step 2.

Spatially Aware Union-Find Data Structure: Union Find data structure have been classically proposed in computer science to keep track of a set of elements partitioned into a number of non-overlapping subsets. This classic data-structure can thus be used to dynamically track the connected components of a graph as vertex are added as they model disjoint set conceptually.

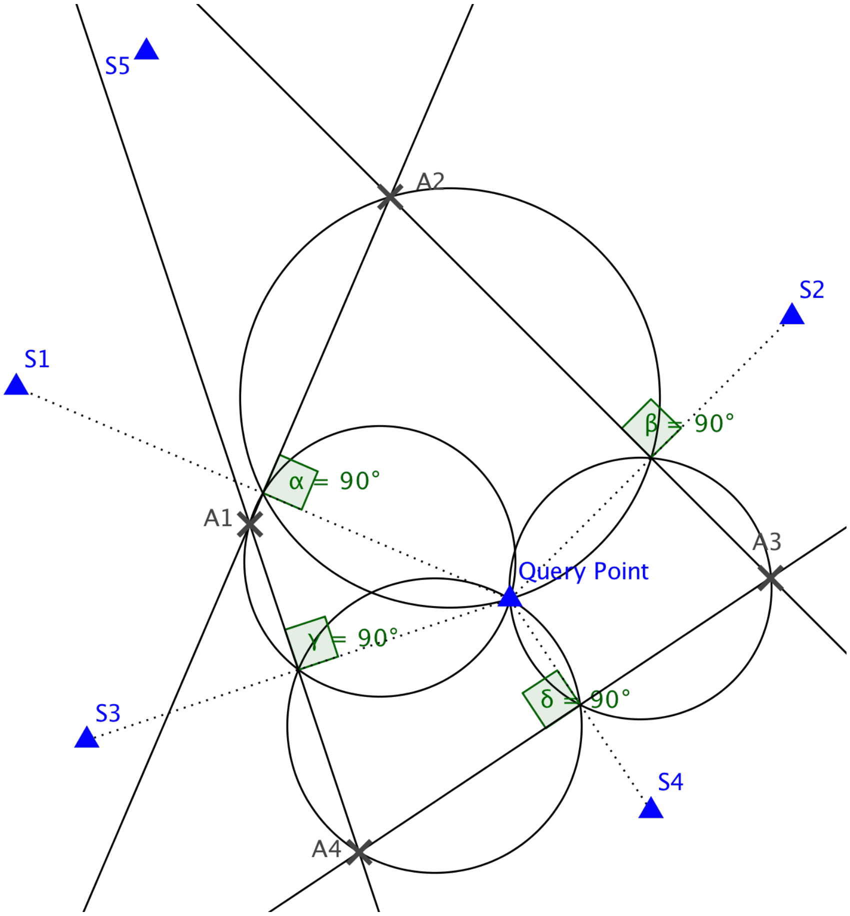


Figure 5.2. Nearest Neighbors with corresponding circles

Fundamentally, the classic Data Structure supports the following two operations:

1. **Find**: This operation can be used to check the set in which an element lies.
2. **Union**: This operation can be used to join / union two disjoint subsets into a single large subset.

As we will propose next in the section, during the discovery phase, we want to keep track of all clients and the social groups they lie in. Thus a union-find based data structure seems a good fit. However, our use case is slightly different from the classic data-structure. In addition to simple *union and find* operations, we need to support:

- **Iterate**: Given a client as *farthest actor*, this operation can be used to iterate on all other clients that lie in the same connected component within a distance bounded by the queried *farthest actor*.

We will use a simple example to demonstrate the idea behind a spatially-aware Union-Find Data-Structure. Walking through an example, we will augment a simple Union-Find data-

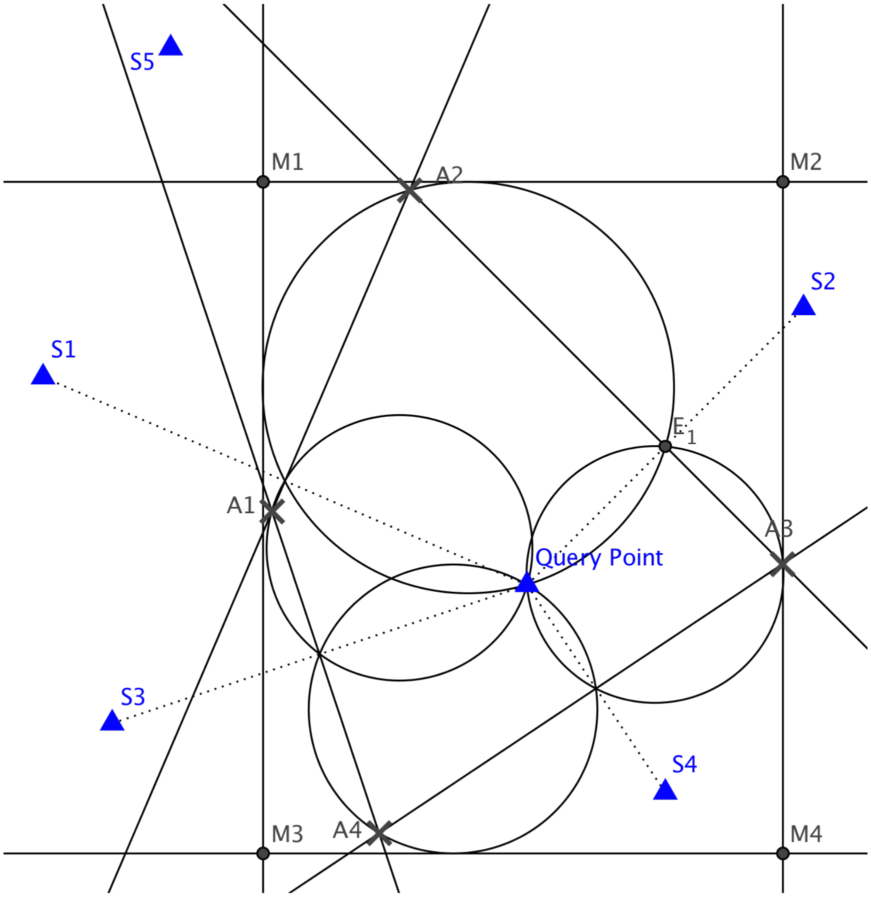


Figure 5.3. MBR bounding area for Reverse Nearest Social Groups

structure into a Spatially aware counterpart. Let us assume a scenario where we start with five nodes $0, 1, 2, 3$ and 4 each in a different connected component. This is shown in Figure 5.4.

Figure 5.4. Union-Find: Five different Sets / Connected Components

Next, we merge 2 with 3 to make a single set (imagine a scenario where 2 and 3 share an edge).

Finally in Figure 5.6, we merge 1 with the set consisting of 2 and 3 to insert it into a connected component.

Figure 5.7 shows the state of Union Find data-structure, where reference to parent points are shown corresponding to the state described in Figure 5.4, 5.5 and 5.6. The readers are urged to note that the Union Find data-structure also supports height information to support faster query runtimes. However this information has been omitted in the current figure for the sake of brevity.

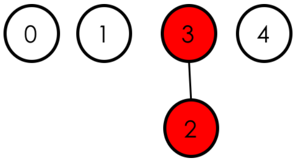


Figure 5.5. Union-Find: Four Connected Components

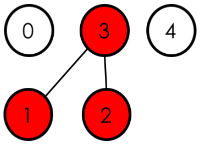


Figure 5.6. Union-Find: Three Connected Components

To augment the data-structure with additional spatial information, we propose to make the following set of changes: Our implementation maintains double pointers for every node i.e. every parent node points to all immediate child nodes. Thus after using the *Find* operation to locate the parent node, we can traverse the child pointers to iterate on all other nodes in a connected component. In this example, this would imply storing 2 and 1 as child nodes for 3. Next, every node stores its distance to query point thus allowing to filter nodes that lie beyond a *farthest actor*.

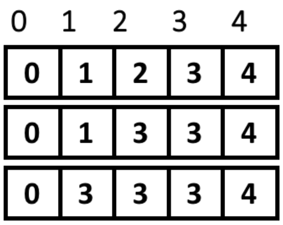
The resulting data-structure is visualized in Figure 5.8. Note that the first two rows for parent pointers are shown for completeness and demonstrating the evolution of the data-structure with join operations (they are not part of the final snapshot). Extra meta-data in the form of distance is stored along with every client. Moreover, each client also a list of its immediate child nodes.

Discovery Phase example for RNSGQ The adoption of the technique for calculating RNSGQ is now explained with an example: Figure 5.9 shows a simple scenario with our query point Q and five other serves S1, S2, S3, S4 and S5. Clients location H1 to H15 are also present along with the server points. We assume the value for k defining core constraints is 2.

We use the distance browsing technique with R*-Tree to visit data points in ascending order. As discussed previously, we are presently interested in pruning search space, discovery of data points in the region and indexing them by social characteristics. Thus the key goals for this phase are: 1) Finding Servers in all four quadrants that constrict influence region of Q and 2) Finding all clients that lie in this region. The idea is to incrementally maintain connected components information. A spatially aware Union Find structure is employed the purpose (as described in the previous paragraph). This will help in checking for social constraints faster.

We want to accomplish all this by visiting data points just once and henceforth saving computation and I/O costs. We trace the example in Figure 5.9 to show a step by step computation of our result set:

1. We start by visiting point H7, moving to H11, H10, H8, H6 and H1. Ids of these elements are inserted into the spatially aware union find data-structure UF.



0	1	2	3	4
0	1	2	3	4
0	1	3	3	4
0	3	3	3	4

Figure 5.7. Union-Find: Timeline for Connected Component Information

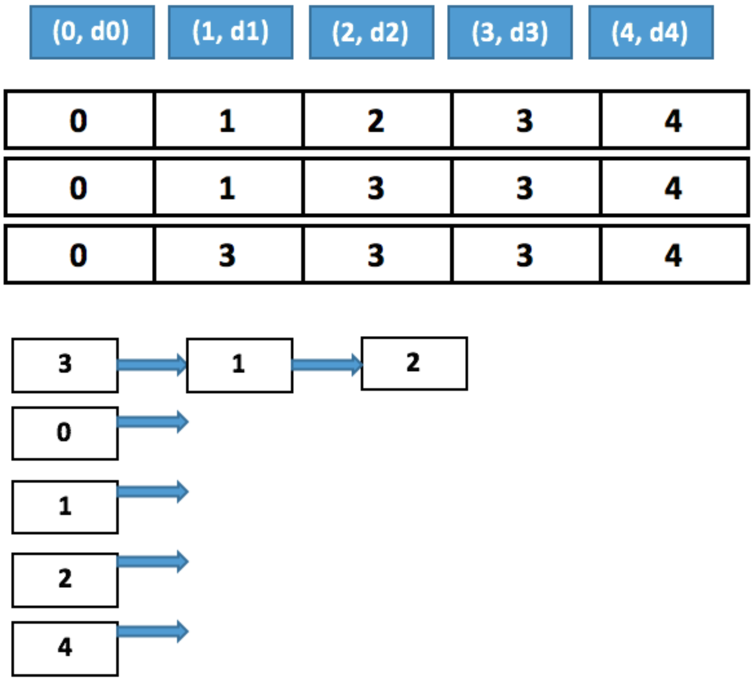


Figure 5.8. Design of Spatially Aware Union-Find Data Structure

2. We now encounter S4. As it is a server, we check the quadrant it lies and then add it to the Server list (no server found in quadrant 4 yet).

3. H5, H2 are next discovered point and added to UF.

4. We now have an interesting case, the point is H9 which is in fourth quadrant and farther from the NN Server S4. Can we simply prune this point?
The answer is NO, the pruning is not possible as H9 is connected to H12 (which we havent discovered yet). Even though H9 is not a RNN point, it is part of social group where farthest actor H12 could be a potential RNN for Q. Thus the point is simply added to UF.

5. Server S2 is discovered and added to Server list (Quadrant 2).

6. Further along we find points H3 and H4. As degree of H3 $<$ core constraint, it is pruned. H4 is added to UF.

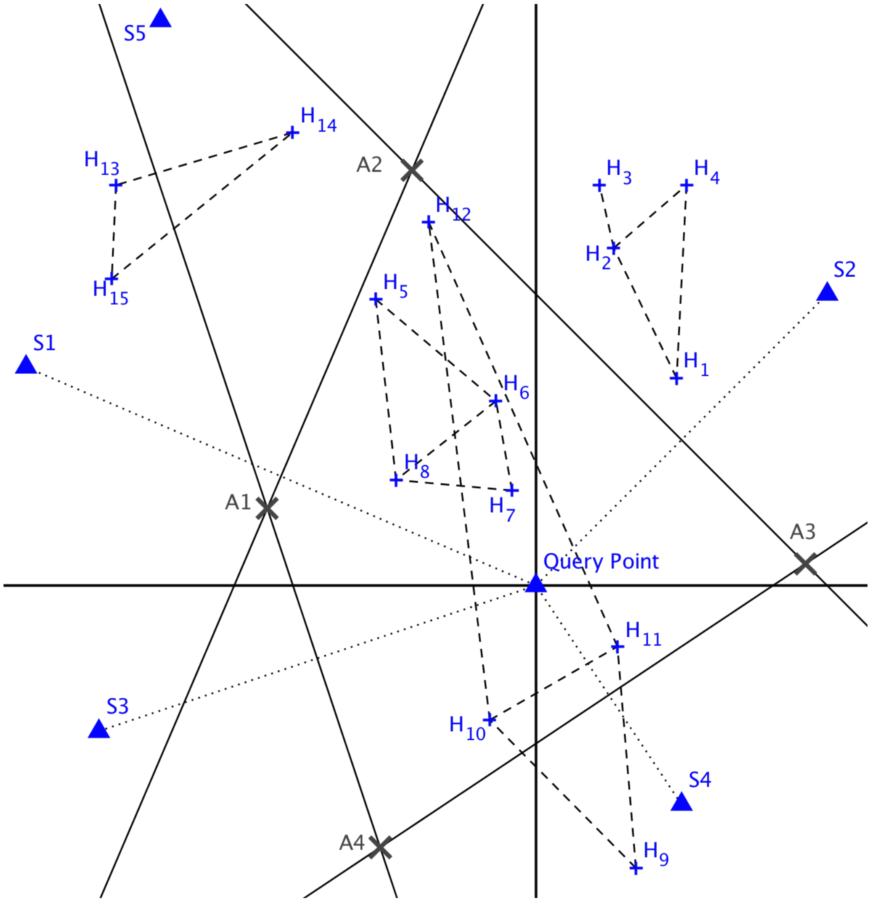


Figure 5.9. Reverse Nearest Social Group Query Example

7. Server S3 in 3rd quadrant and S1 in 1st quadrant are discovered and added to server list. At this point, we have discovered NN in all four quadrants. Consequently, we can compute a *Refined(q)* and as discussed by authors that restricts the possible area for finding competitive serves. However, along with restricting server distance, there is another crucial spatial pruning implication. This brings us to Lemma 2.

Lemma 2: *For a given query point Q and an approximate region Approx(q), defined by four vertex A1, A2, A3 and A4. If D is maximum distance among d(Q,A1), d(Q,A2), d(Q,A3) and d(Q,A4). Then, there does not exist any group member further away from D.*
Proof: As per Lemma 1, a group consist of a *farthest actor* and all members satisfying social constraints that lie within d (q, r) (where r is the distance of farthest actor from query point) . Thus a group's bound on distance from query point is dictated by the *farthest actor*. We now extend this a step further. In our case, the farthest Reverse Nearest Neighbor situated at a distance D sets a bound on maximum distance of any farthest actor. Thus consequently defining a bound on all groups and their respective members. Figure 5.10

demonstrate the idea visually.

8. We now discover H15, H14 and H13 but choose not to store them in UF as they cannot possibly be in the solution set as dictated by Lemma 2.

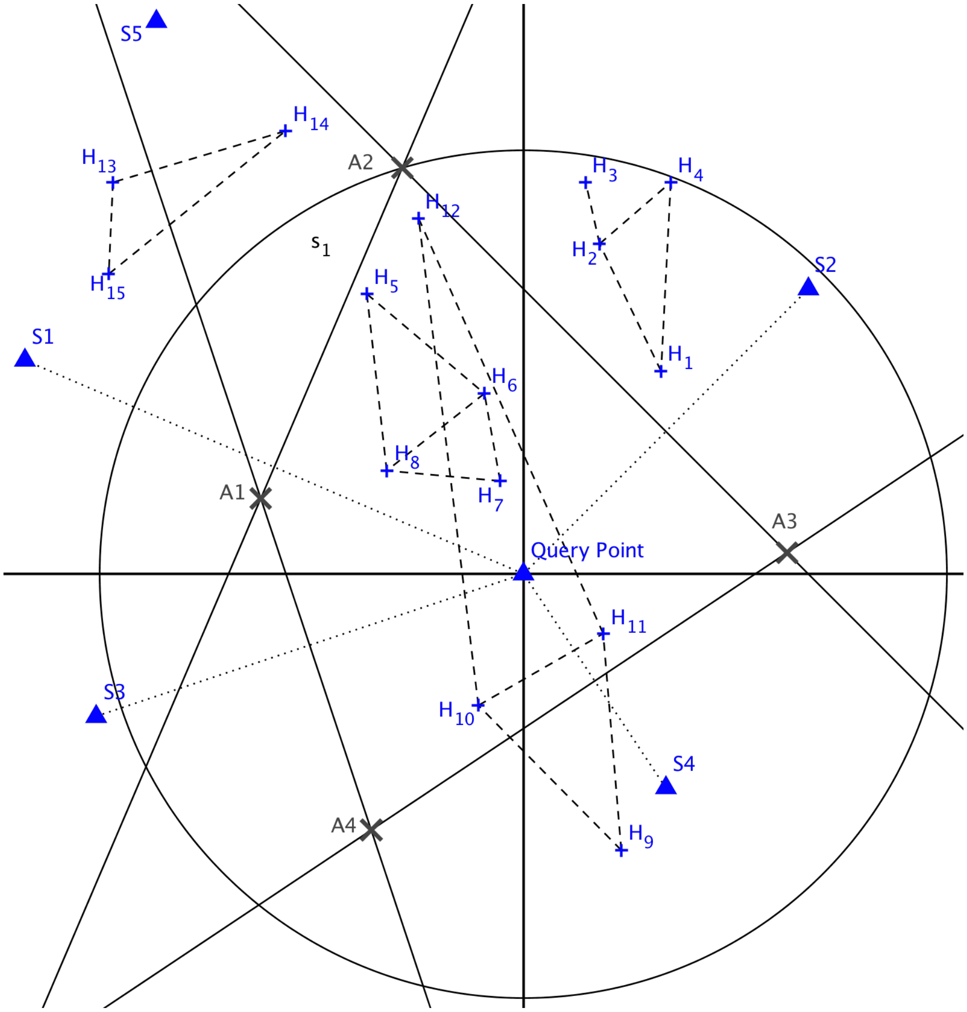


Figure 5.10. Pruning for Reverse Nearest Social Group Query

9. S5 is discovered and added to Server List.

At the end of discovery phase, we have the following memory data-structures with us: UF and Server List. The contents for the example are listed below :
$Server\ List = \{S4, S3, S3, S1, S5\}$
$UF = \{\{H9, H10, H11, H12\}, \{H5, H6, H7, H8\}, \{H1, H2, H4\}\}.$

We omit to include the exact structure of UF as the idea behind Spatially aware Union-Find data-structure has been discussed previously. Moreover, a precise internal structure might vary depending on the implementation details. However, conceptually the idea is simple, we have the connected components stored as disjoint sets. The spatial Union Find data structure as discussed previously allows us to perform following operations quickly:

1. Find: Quickly find which connected component a point lies in.
2. Iterate: Quickly iterate on all members in a connected component within a distance bounded by *farthest actor*.

Computation Phase: With all the necessary information, we are now armed to calculate the RNSG(s) for query point Q. All the information is encoded in Server List and UF.

The highlights for this step are :

1. Iterate on all connected component (any order) but visiting individual points in descending order of distance in UF to check for all farthest actor (by comparing the distance with q and the other list of Servers). It is important to note that sorting is required in order to visit nodes in decreasing order of distance. This requires a small amount of extra work in the size of a set (running a fast sorting algorithm). However, as in our scenario the list is small due to pruning of search space to Refined(q), this step is not computationally expensive.
2. Finding elements that lie within the group for this point and checking for core constraints. The state of the art algorithm for finding core number runs in linear time. This is computationally cheap but can get expensive if executed repeatedly on large graphs. Similar to the previous point, although we use this subroutine as it is, we only run the k-core algorithm on a very small pruned input set. Thus our implementation is again very fast.

Tracing the example presented in Figure 5.9: We start with H4 and compare the point distance to servers in Server List and conclude that the point is not a farthest actor. H2 and H1 fail on similar ground. Client H5 satisfies the NN criteria. Retrieving rest of group members in the connected component H6,H7,H8 we find that they also satisfy the core constraint and are thus reported as social group. H12 and H9 fail to satisfy the farthest actor criteria. H10 and H11 although do satisfy the spatial constraint but fail to be a part of the result as social criterion is not satisfied.

The algorithm highlights another key strength of our solution discussed previously in Chapter 1. As our algorithm performs excellent pruning in both social and spatial domains, it is highly suitable for an online environment where maintaining low-latency is critical. Moreover as we do perform any pre-processing that imposes a strict notion of a server and a client to pre-construct an index, the roles of server and client can be chosen during the runtime.

5.2 Algorithm for Reverse Nearest Social Group Query

Now that our readers are familiar with the workings of our proposed algorithm for Reverse Nearest Social Group Query, we formally define the R-SAGE algorithm. The two phases as discussed previously are packages into seperate subroutines.

When executing the R-SAGE algorithm, we make the following set of simple assumptions. **IsKCore**: We have an access to a subroutine *IsKCore* that checks whether a graph satisfies the k-Core constraint as proposed in [9]. **IsSever**: This subroutine checks whether a point is a server and returns true / false accordingly. **IsQuadrantNN**: The subroutine *IsQuadrantNN* computes which quadrant a point lies in and then further ensures that no other point was previously discovered in this quadrant (thus ensuring the discovered point is Quadrant's Nearest Neighbor).

Our proposed algorithm R-SAGE*(Rnn-based Social-Aware Group discovEry Algorithm)* can thus easily locate all groups in the influence set without any pre-computation. This is feasible only because of the novel insights applied by the technique in both spatial as well as social domain.

Algorithm 3 Reverse Nearest Social Group Query

```
procedure DiscoverServerClients(RTree)
    $NNS, SL, UF \leftarrow \{\}$
    $discoveredServer \leftarrow$ FALSE    // Flag to exit loop when all Servers have been found
    $ApproxQ \leftarrow \{-\infty, \infty, \infty, -\infty\}$
    $RefinedQ \leftarrow \{-\infty, \infty, \infty, -\infty\}$
    repeat
        $\forall v \in RTree$ in increasing order of distance
        if ( IsServer(v) && IsQuadrantNN(v) ) then    // Check if v is quadrant's NN server
            NNS.insert(v)
            if (NNS.length() = 4) then    // We have all Nearest Neighbor
                $ApproxQ \leftarrow$ intersection of bisectors $|Q - L[0]|, |Q - L[1]|, |Q - L[2]| \& |Q - L[3]|$
                Calculate center for MBR circle $(c_1x, c_1y)...(c_4x, c_4y)$
                Calculate radius for MBR circle $r_1...r_4$
                $Low(x) \leftarrow min((c_1x - r_1), (c_2x - r_2), (c_3x - r_3)), (c_4x - r_4))$
                $Low(y) \leftarrow min((c_1y - r_1), (c_2y - r_2), (c_3y - r_3)), (c_4y - r_4))$
                $High(x) \leftarrow max((c_1x + r_1), (c_2x + r_2), (c_3x + r_3)), (c_4x + r_4))$
                $High(y) \leftarrow max((c_1y + r_1), (c_2y + r_2), (c_3y + r_3)), (c_4y + r_4))$
                $Refined(q) \leftarrow 2 * low_x - q_x, 2 * high_x - q_x, 2 * low_y - q_y, 2 * high_y - q_y$
            end if
        else if ( v $\in$ Refined(q) ) then    // Point lies within Refined(q)
            if ( IsServer(v) ) then
                SL.insert(v)
            end if
            if ( IsClient(v) && v $\in$ MAXDIST(ApproxQ) ) then
                UF.insert(v)
            end if
        else
            discoveredServer = FALSE
        end if
    until $(discoveredServer = FALSE)$
end procedure

procedure ComputeRNSGQuery(NNS,SL,UF,QueryPoint)
    for all Groups $C \in UF$ do
        for all client point $v \in$C in descending order of distance do
            if ( NearestNeighbor(v) = QueryPoint) then
                $C \leftarrow$ UF.find(v)    // Fetch clients in $C$ bounded by distance to $v$
                if (IsKCore($C$) then
                    Return the core as a solution.
                end if
                Mark all points in C as processed.
            end if
        end for
    end for
end procedure
```

Chapter 6

Theoretical Analysis

In Chapter 4 we proposed two baseline algorithms for calculating the Reverse Nearest Social Group Query. Further Chapter 5 proposed an improvised algorithm for calculating social groups in the influence set for a query point. However, we refrained for providing any formal verification to aid in proving the correctness of our algorithms. In this chapter we fill in the missing pieces and provide a strong theoretical foundation for checking correctness of our algorithm. In addition, we also provide a framework for understanding performance characteristics of our solutions as quantifying performance is as critical as correctness. We use several different datasets and criteria to reinforce our results.

Correctness Proofs: Before we can argue on the correctness of our algorithms, we first need to answer a much more fundamental question, i.e. *Irrespective of any specific technique, what are the fundamental problems any RNSG algorithm must solve in order to derive a correct answer?*. Once we have a precise answer to this question, we can then use it as a measure to prove the correctness for any given algorithm. With this insight, let us proceed to define our correctness framework properties. As dictated by the problem definition (consisting of distance function *MINMAXDIST*, *farthest actor* and *core-constraint*), any solution needs to satisfy three correctness properties:

Property 1: All Reverse Nearest Social Groups consist of a *farthest actor* that is a Reverse Nearest Neighbor of the query point and lies at a distance greater than any other point in the group.

Property 2: All Reverse Nearest Social Groups calculated minimize the maximum distance from a group to query point.

Property 3: All Reverse Nearest Social Groups calculated satisfy core constraints.

Thus for all our solutions, a group in a solution set as specified in Lemma 1: Consists of a *farthest actor* v at a distance r and all v_i that lie within $d(q, r)$ and is a k-core. Lemma 1 further dictates that such a group indirectly satisfies all three properties (a formal proof in Section 4.1).

Thus we can rephrase the two correctness properties for our baseline as: Correctly identifying all farthest actors with corresponding groups having all members within a distance less than the farthest actor from a query point and correctly checking the k-core constraints.

With this background, we now proceed to individually verify the two base line approaches. The Spatial First algorithm iteratively looks for client data points that are situated near the query point. Next, the algorithm starts a BFS to find the entire connected component for this point. These points are then sorted in descending order of distance and iteratively checked for satisfying the farthest actor criteria. An in-depth algorithm explanation has been provided in Chapter 4.

Theorem 1: *Spatial First algorithm finds all Reverse Nearest Social Groups.*
Proof: For a connected component, the algorithm sorts the data points in descending order of distance and performs Nearest Neighbor checks on every point. Thus the farthest actor is calculated correctly. The connected component members are sorted by the distance to query point, once the farthest actor is found, only members of the group at a lesser distance are considered. Thus the farthest actor and the corresponding group within $d(q, r)$ is found correctly. Finally, a k-core check is performed on all qualified elements satisfying Property 3. The algorithm continues to execute until all the *farthest actors* and the corresponding groups are found (by terminating after processing all points), thus finding all qualified groups.

The Social First algorithm starts by running a k-core preprocessing step and retrieving core number for every point. For every corresponding core, data points are considered in descending order of distance and checked for the *farthest actor* criteria. A subset of elements satisfying spatial constraints are then checked for fulfilling the social constraint. An in-depth algorithm explanation has been provided in Section 4.1.

Theorem 2: *Social First algorithm finds all Reverse Nearest Social Groups.*
Proof: A k-core based pruning is performed and cores are then considered in descending order of distance from query point to be checked for farthest actor constraint (by repeated executing Nearest Neighbor query). This allows us to find the *farthest actor* and all elements at a distance lesser from the query point thus satisfying Property 1 and 2. A k-core check is performed on all qualified elements satisfying Property 3. The algorithm continues to execute until all the cores are processed thus finding all qualified groups.

Our final solution R-SAGE builds on the state of art techniques for finding Reverse Nearest Neighbor clients. Along with finding the RNN set, the algorithm keeps track of social connectivity and distance of clients from the query point to allow quick checks on socio-spatial constraints. An in-depth explanation has been provided in Section 5.1.

Theorem 3: *R-SAGE algorithm finds all Reverse Nearest Social Groups.*
Proof: The algorithm finds *farthest actors* correctly by finding the RNN set for query point (satisfying Property 1).The algorithm then uses smart data-structures to consider clients in the

same connected component having distance less than *farthest actors* from the query point. This ensures that Property 2 is satisfied.Finally, all the elements are checked for core-constraints thus satisfying Property 3. The algorithm continues to execute until all *farthest actors* (by repeating the above steps for all qualified connected components) and the corresponding clients have been processed. Thus retrieving all qualified groups.

Chapter 7

Experimental Analysis

Before we take a plunge into the experimental results, we provide an in-depth dataset analysis for the readers. This is important as performance of algorithm is tightly linked to the dataset characteristics. Thus it is critical to have a good sense of the data used for experimentation. We use two real world datasets (popular LBSN *Foursquare* [1] and now defunct *Whrrl* [6]) for our analysis.

A lot of research efforts in the past have been dedicated to analyzing real world social graphs and some of the interesting phenomenons observed that are specific to these networks [30] [18] [8]. It is thus well established that social networks differ greatly from randomly generated graphs. As one of the key implications for RNSGQ is to quantify influence in Location based social network, it makes much sense to use a real world dataset fort this analysis. This helps us to get closer into a real world system when designing a solution.

The datasets have been borrowed from the past research in Socio-Spatial domain [33]. Both of these datasets consist of following attributes: 1) A list of businesses ids along with their location. 2) A list of users and their friendship graph and 3) A list of user check-ins (past history of user visits to businesses).

In order to make the data suitable for our work, we cleanse and preprocessing the data. Here we list the key scrubbing and transformation performed on the original raw data. We remind our readers that the transformation is applied on both the datasets. The transformation involves : 1) Removing all locations missing location(latitude-longitude) data. 2) Removing all users who have past checkins only in businesses with missing location data. 3) Discarding friendship tuples that involve a deleted users. 4) Converting all user ids to a numeric string and 5) Pre-processing checkin history of a client to obtain a single home location by calculating *centroid* point of all past visited locations.

The location data is represented as Latitude and Longitude. A *Plus/Minus* notation is used for encoding extra direction information. We provide a concise review for our readers to quickly remind them of the system : The *Latitude* of a pointmeasures how far north or south of the equator a place is located. To provide an anchor, the equator is situated at 0°, the North Pole at 90° north (or simply 90°, as positive latitude implies north), and the South Pole at 90° south (or 90°). Thus Latitude measurements range from 0° to (+/) 90°. *Longitude* on the other handmeasures how far east or west of the prime meridian a place is located. A minus sign indicates west of prime meridian and a positive sign implies east. The longitude measurements range from 0° to (+/)180°.

Let us start our dicussion by summarizing the key Foursquare dataset attributes in Table 7.1. Next Table 7.2 shows a pattern of check-ins for foursquare and Table 7.3 summarizes information on connected components in the dataset. Finally we summarize the *k*core attributes for the dataset in Figure 7.1.

Attribute	**Count**
Server Count	57117
Client Count	127753
Friendship Count	591876

Table 7.1. Foursquare Dataset Summary

Check-In Count	**Number of Users**
1 to 10	123256
11 to 20	3850
21 to 30	495
31 to 40	89
41 to 50	493
51 to 400	28

Table 7.2. Foursquare Check-In Summary

User Count	**Number of Connected Components**
87229	1
5	1
4	5
3	11
2	34

Table 7.3. Foursquare Connected Component Summary

In our study, Foursquare is the larger dataset. It has a higher client to server ratio. Moreover, the friendship graph is dense. Table 7.2 shows the distribution of check-in counts in the dataset. We observe that most of the clients have a smaller number of check-ins. Further, Table 7.3 shows number of components with a specific user count. We observe that most of the clients are part of a single large connected component with 87229 nodes. Finally, Figure 7.1 summarizes the *k*-Core distribution of the graph. We observe that the largest core has a consists of 87355 nodes.

We now a present a similar analysis for the Whrrl dataset. The key dataset attributes are summarized in Table 7.4. Similar to Foursquare dataset, we use Figure 7.5 to show a pattern of check-ins for Whrrl and Table 7.6 to summarize connected component information. Finally we summarize the *k*core attributes for the Whrrl dataset in Figure 7.2.

In our study, Whrrl is the smaller dataset. Unlike Foursquare, it has a higher server to client ratio. Moreover, the friendship graph is relatively sparse. Table 7.5 shows the distribution of

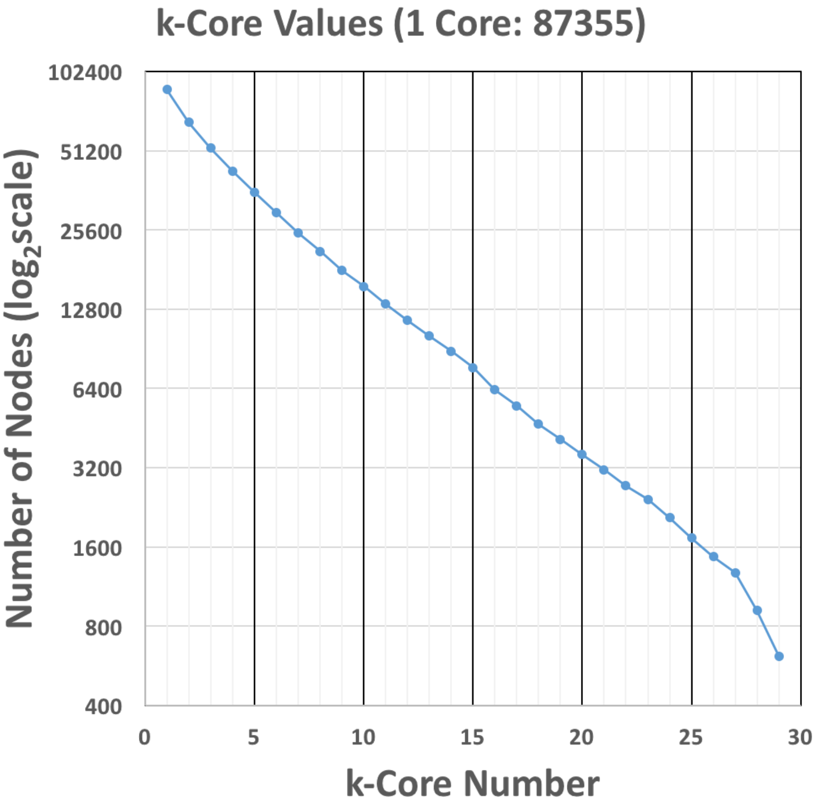


Figure 7.1. Foursquare k-core

Attribute	Count
Server Count	47828
Client Count	5256
Friendship Count	48626

Table 7.4. Whrrl Dataset Summary

check-in counts in the dataset. We observe that most of the clients have a smaller number of check-ins. Next, Table 7.6 shows number of components with a specific user count. Most of the clients are part of a single large connected component with 4871 nodes. Figure 7.2 summarizes the *k*-Core distribution of the graph. We observe that the largest core has a consists of 4871 nodes.

We now divert our attention to performance analysis of baseline algorithms and our final solution. Particularly, we are interested in a relative comparison between the performance of our proposed solution and baselines. For an empirical performance analysis, we implement all the techniques discussed in this work in C++. All experiments are performed on a Unix machine with an 2.4 GHz Intel Core i5 processor and 8 GB of RAM. We use a R*-Tree implementation in C++ from an open source library *libspatialindex* [2]. The Stanfords *SNAP(Stanford Network Analysis Project)* [5] graph implementation is used to store the social groups. Throughout our

Check-In Count	**Number of Users**
1 to 50	4681
51 to 100	244
100 to 150	75
151 to 200	34
201 to 250	14
251 to 450	14

Table 7.5. Whrrl Check-In Summary

User Count	**Number of Connected Components**
4871	1
8	3
5	3
4	10
3	4
2	30

Table 7.6. Whrrl Connected Component Summary

experiments, we make the following assumptions when designing all the algorithms: 1) The clients are stored in a database where the location is indexed by an R*-Tree that resides in memory. 2) The social graph is stored in-memory. 3) We stimulate a cold start for experiments and set the spatial library buffer pool size set to zero and 4) C++11 <chrono> high resolution clock is used to measure all runtimes. We assume a constant CPU workload when running the experimental analysis (as varying load may affect runtime).

The key idea is to measure runtime performance of algorithms for varying value of k. The Figure 7.3 and Figure 7.4 summarizes our result for the two datasets Foursquare and Whrrl. We observe the runtime trend for Foursquare and Whrrl datasets by varying the value of k. This is achieved by executing 1000 random queries for every value of k. A graph is then plotted on a log scale and shows the runtime trends for *Spatial First*, *Social First* and the *R-SAGE* algorithm. Note that the *Spatial First* approach is missing for the Foursquare dataset as it does not retrieve any results in a reasonable amount of a time(60minutes).

We summarize the k-Core trend for Foursquare. The *Social First* starts with a very high runtime with an average of 8.3 seconds per query. However, as we increase the value of k, a sharp decline in the runtime is observed. With a value of k as 25, the average runtime comes down to 2.81 seconds. This can be clearly attributed to a large pruning of clients during preprocessing where a lot of clients are eliminated from the search space as they do not satisfy the core constraint. The *R-SAGE* algorithm is observed to be independent of the value of k as the majority of pruning is performed by restricting the search space and k-Core computation is simply run on a very small subset of clients.

We can now summarize the k-Core trend for Whrrl. The *Spatial First* approach does not show much of a variation with changing value of k. This is because the algorithm is based on a scanning approach and has to traverse the search space for finding a solution set. The runtime is very high and averages around 4.5 seconds. The *Social First* approach, similar to the Foursquare dataset shows a similar trend. The query starts with a high average runtime of 2.36 seconds per query for k value of 5. However, as we increase the value of k to 25, the average runtime

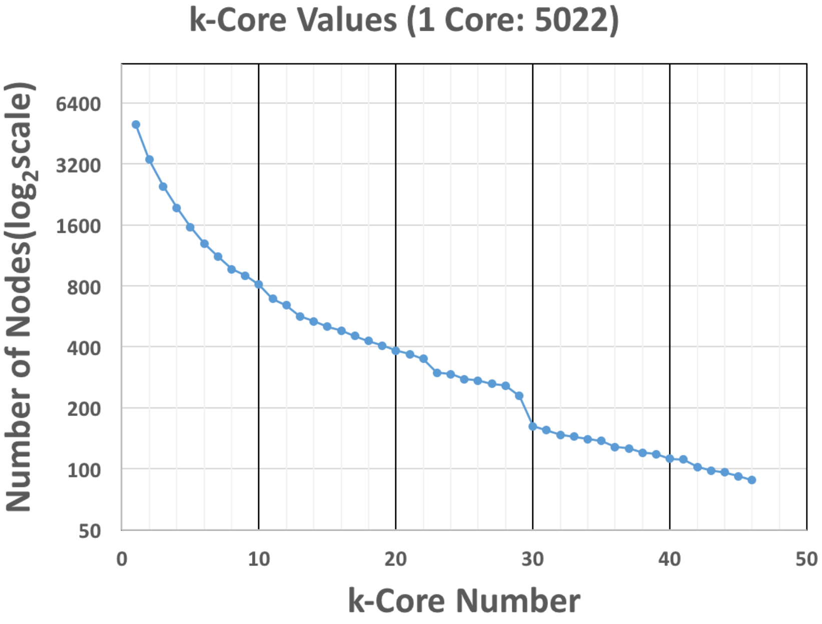


Figure 7.2. Whrrl k-core

decrease to 0.0963 seconds. Finally, the *R-SAGE* shows no correlation to the increasing value of k as discussed previously. We observe an average runtime of 0.051 seconds for k as 5. The value remains relatively unchanged for k as 25 at 0.059 seconds.

We would also like to measure runtime over an extended number of queries. The Figures 7.5 and 7.6 summarizes our result for the two datasets Foursquare and Whrrl. We observe the runtime trend for Foursquare and Whrrl datasets by increasing the number of queries while keeping the value of k constant at 10. A graph is then plotted on a log scale that shows the runtime trends for *Spatial First*, *Social First* and the *R-SAGE* algorithm. Just as we observed when looking at the runtime with respect to k value, the *Spatial First* approach is missing for the Foursquare dataset as it does not retrieve any results in a reasonable amount of a time(60minutes).

The k-Core trend for Foursquare can now be summarized. The *Social First* algorithm has an average runtime of 5.3 seconds for 25 random queries. The runtime shots up a little to 6.7 seconds when executing 1000 random queries. We attribute this to the entropy in the system when executing a large number of queries over an extended period of time. The *R-SAGE* algorithm in contrast shows a clear increase in runtime as we increase the number of queries. This might be a little perplexing for the readers and might seem counter intuitive. The observation however stems from the fact that the algorithm relies on dataset properties (finding NN servers) for pruning a large number of clients and servers. With increasing number of random queries, the algorithm uses outliers as inputs that do not have Nearest Neighbor Servers in all four quadrants restricting the search space. For these cases, the pruning is inefficient and the average runtime is increased.

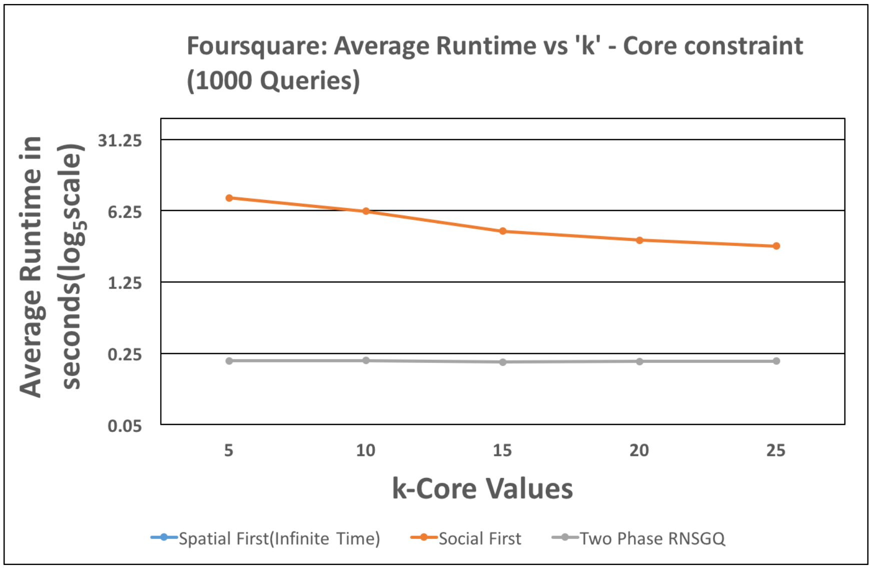


Figure 7.3. Foursquare k-core vs Runtime

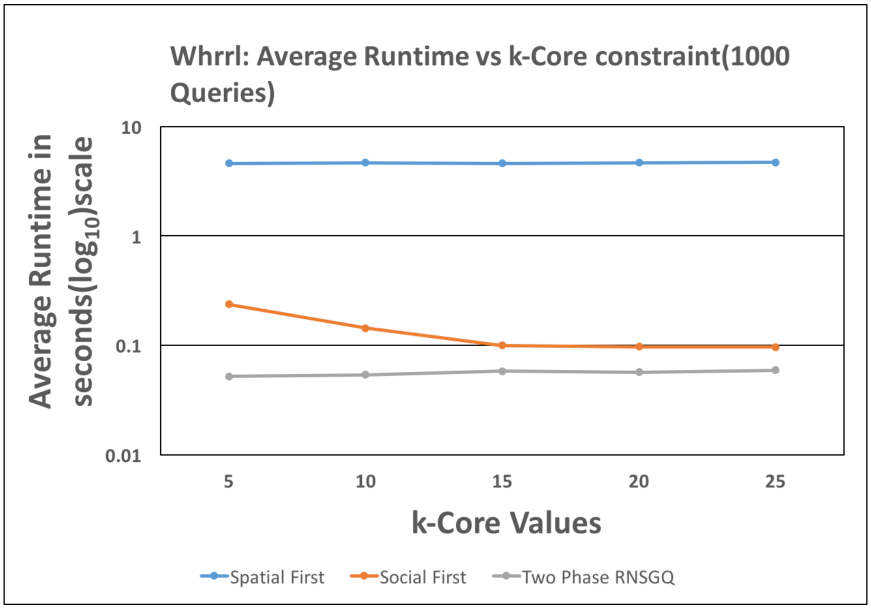


Figure 7.4. Whrrl k-core vs Runtime

To conclude, we summarize the k-Core trend for Whrrl. The *Spatial First* algorithm has an average runtime of 4.6 seconds for 25 random queries. The runtime stays relatively constant around 4.6 seconds when executing 1000 random queries. The *Social First* algorithm has an average runtime of 0.132 seconds for 25 random queries (due to pruning with k value as 10). The runtime stays constant around 0.138 seconds when executing 1000 random queries.The *R-SAGE* algorithm as discussed previously for Foursquare dataset shows an increase in runtime with increasing number of queries. The average runtime is 0.009 seconds for 25 random queries

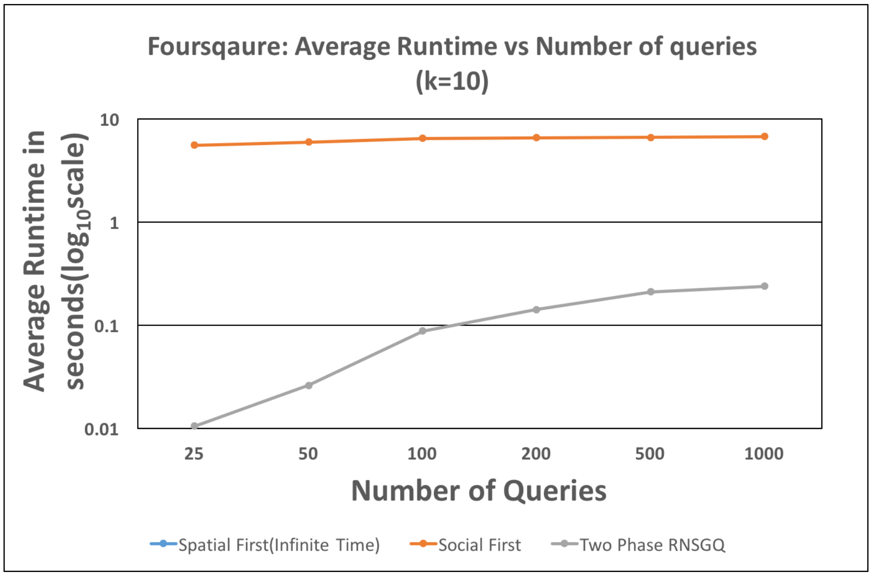


Figure 7.5. Foursquare Query Number vs Runtime

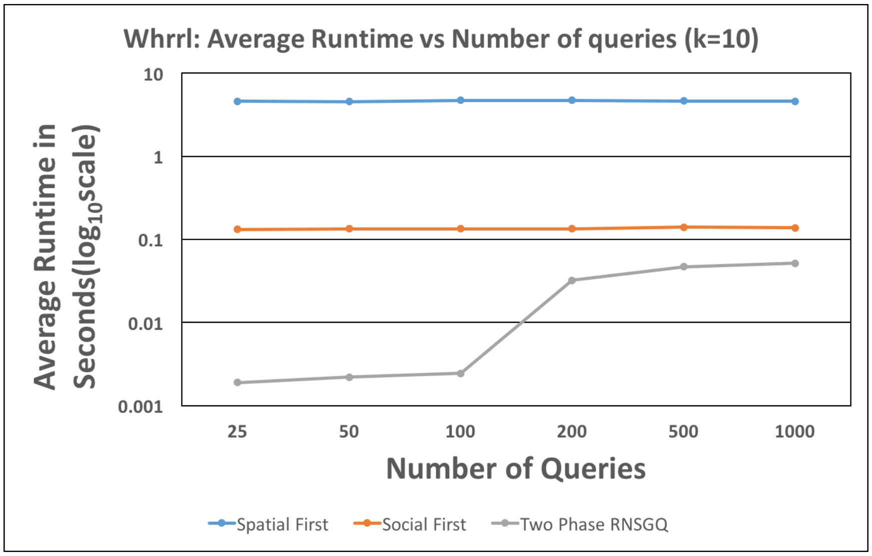


Figure 7.6. Whrrl Query Number vs Runtime

which shoots to 0.051 seconds for 1000 random queries.

Chapter 8

Conclusion

We start our discussion by highlighting the exponentially increase in user generated socio-spatial data and discuss the associated challenges involved in storing and querying massive socio-spatial datasets. We acknowledge a critical need for database designers to tackle novel challenges when designing queries in socio-spatial domain. Next we discuss the idea of quantifying influence in socio-spatial domain. We talk about *influence* for social groups and its ties to Reverse Nearest Neighbor Query in spatial domain. We formulate the problem Reverse Nearest Social Group Query (RNSGQ) mathematically for a concrete definition. For motivating RNSGQ, we propose a variety of applications that span several different domains. Further, we motivate the idea that extending the notion of *influence* to social groups as a natural extension to the RNN problem. With the definition of RNSGQ, we can model further rich problems that involve influence particularly important in the Socio-Spatial domain.

We present a *farthest actor* based analysis to exemplify the idea of minimizing the maximum distance from a group to a query point. Next, we use this analysis to propose two different approaches for solving our problem : 1) A Spatial first approach that prioritizes spatial pruning and 2) A Social first approach that in contrast emphasizes on social pruning. To propose an efficient solution and balancing the tradeoffs in both spatial and social domains, we next propose a two-phase R-SAGE(Rnn-based Social-Aware Group discovEry) algorithm. The proposed algorithm R-SAGE was shown to easily locate all groups in the influence set without any precomputation. This was feasible only because of the novel insights applied by the technique that facilitate excellent pruning. Another key strength of our algorithm relies on the fact that it does not impose a strict notion of a server and a client as no pre-processing is involved. Thus the roles of server and client can be chosen during the runtime.

Concluding, we successfully formulate the notion of Reverse Nearest Social Groups and the idea of quantifying influence set consisting of social groups. The proposed algorithm **R-SAGE** is empirically demonstrated as an efficient problem solution.

Bibliography

[1] Foursquare : http://foursquare.com.

[2] Libspatialindex : http://libspatialindex.github.io.

[3] Meetup : http://meetup.com.

[4] Mysql : https://dev.mysql.com/doc/refman/5.5/en/index-btree-hash.html.

[5] Snap : http://snap.stanford.edu.

[6] Whrrl : https://en.wikipedia.org/wiki/whrrl.

[7] Nikos Armenatzoglou, Stavros Papadopoulos, and Dimitris Papadias. A general framework for geo-social query processing. *Proceedings of the VLDB Endowment*, 6(10):913–924, 2013.

[8] Albert-Laszlo Barabasi. Linked: How everything is connected to everything else and what it means. *Plume Editors*, 2002.

[9] Vladimir Batagelj and Matjaz Zaversnik. An o (m) algorithm for cores decomposition of networks. *arXiv preprint cs/0310049*, 2003.

[10] R. Bayer and E. McCreight. Organization and maintenance of large ordered indices. In *Proceedings of the 1970 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control*, SIGFIDET '70, pages 107–141, New York, NY, USA, 1970. ACM.

[11] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. *The R*-tree: an efficient and robust access method for points and rectangles*, volume 19. ACM, 1990.

[12] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[13] Raphael A. Finkel and Jon Louis Bentley. Quad trees a data structure for retrieval on composite keys. *Acta informatica*, 4(1):1–9, 1974.

[14] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, SIGMOD '84, pages 47–57, New York, NY, USA, 1984. ACM.

[15] Gísli R Hjaltason and Hanan Samet. Distance browsing in spatial databases. *ACM Transactions on Database Systems (TODS)*, 24(2):265–318, 1999.

[16] Flip Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. *SIGMOD Rec.*, 29(2):201–212, May 2000.

[17] Flip Korn and S Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In *ACM SIGMOD Record*, volume 29, pages 201–212. ACM, 2000.

[18] Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a feather: Homophily in social networks. *Annual review of sociology*, pages 415–444, 2001.

[19] Robert J Mokken. Cliques, clubs and clans. *Quality & Quantity*, 13(2):161–173, 1979.

[20] Jürg Nievergelt, Hans Hinterberger, and Kenneth C Sevcik. The grid file: An adaptable, symmetric multikey file structure. *ACM Transactions on Database Systems (TODS)*, 9(1):38–71, 1984.

[21] Nick Roussopoulos, Stephen Kelley, and Frédéric Vincent. Nearest neighbor queries. In *ACM sigmod record*, volume 24, pages 71–79. ACM, 1995.

[22] Hans Sagan. *Space-filling curves*, volume 18. Springer-Verlag New York, 1994.

[23] Stephen B Seidman. Network structure and minimum degree. *Social networks*, 5(3):269–287, 1983.

[24] Stephen B Seidman and Brian L Foster. A graph-theoretic generalization of the clique concept*. *Journal of Mathematical sociology*, 6(1):139–154, 1978.

[25] Timos Sellis, Nick Roussopoulos, and Christos Faloutsos. The r+-tree: A dynamic index for multi-dimensional objects. 1987.

[26] Chih-Ya Shen, De-Nian Yang, Liang-Hao Huang, Wang-Chien Lee, and Ming-Syan Chen. Socio-spatial group queries for impromptu activity planning. *arXiv preprint arXiv:1505.02681*, 2015.

[27] Amit Singh, Hakan Ferhatosmanoglu, and Ali Şaman Tosun. High dimensional reverse nearest neighbor queries. In *Proceedings of the twelfth international conference on Information and knowledge management*, pages 91–98. ACM, 2003.

[28] Ioana Stanoi, Mirek Riedewald, Divyakant Agrawal, and Amr El Abbadi. Discovery of influence sets in frequently updated databases. In *VLDB*, volume 2001, pages 99–108, 2001.

[29] Yufei Tao, Dimitris Papadias, and Xiang Lian. Reverse knn search in arbitrary dimensionality. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 744–755. VLDB Endowment, 2004.

[30] Duncan J Watts and Steven H Strogatz. Collective dynamics of small-worldnetworks. *nature*, 393(6684):440–442, 1998.

[31] Jordan Wood. Minimum bounding rectangle. In *Encyclopedia of GIS*, pages 660–661. Springer, 2008.

[32] Congjun Yang and King-Ip Lin. An index structure for efficient reverse nearest neighbor queries. In *Data Engineering, 2001. Proceedings. 17th International Conference on*, pages 485–492. IEEE, 2001.

[33] De-Nian Yang, Chih-Ya Shen, Wang-Chien Lee, and Ming-Syan Chen. On socio-spatial group query for location-based social networks. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 949–957. ACM, 2012.