Efficient Methods for Finding Influential Locations with Adaptive Grids

Da Yan, Raymond Chi-Wing Wong, Wilfred Ng
The Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
{yanda, raywong, wilfred}@cse.ust.hk

ABSTRACT

Given a set S of servers and a set C of clients, an optimal-location query returns a location where a new server can attract the greatest number of clients. Optimal-location queries are important in a lot of real-life applications, such as mobile service planning or resource distribution in an area. Previous studies assume that a client always visits its nearest server, which is too strict to be true in reality. In this paper, we relax this assumption and propose a new model to tackle this problem. We further generalize the problem to finding top-k optimal locations.

The main challenge is that, even the fastest approach in existing studies needs to take hours to answer an optimal-location query on a typical real world dataset, which significantly limits the applications of the query. Using our relaxed model, we design an efficient grid-based approximation algorithm called FILM (Fast Influential Location Miner) to the queries, which is orders of magnitude faster than the best-known previous work and the number of clients attracted by a new server in the result location often exceeds 98% of the optimal. The algorithm is extended to finding k influential locations. Extensive experiments are conducted to show the efficiency and effectiveness of FILM on both real and synthetic datasets.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Spatial databases and GIS*

General Terms

Algorithms, Design, Performance

1. INTRODUCTION

Optimal-location queries are important in a lot of real life applications. For example, a company can issue an optimal-location query to find an influential location to locate its next chain store, so as to maximize its potential revenue.

CIKM'11, October 24–28, 2011, Glasgow, Scotland, UK.

Informally, given a set S of servers and a set C of clients, an optimal-location query returns a location where a new server can attract the greatest number of clients. In the previous example, chain stores correspond to servers while customers correspond to clients, and the company wants to find a location that can maximize the number of customers attracted by its new chain store.

Optimal-location queries can also be used in many other applications, such as profile-based marketing and emergency schedules. However, most of the previous studies assume that the set of existing servers and clients are static, since even the state-of-the-art algorithm for processing optimal-location queries [1] takes hours to answer such queries on typical real world datasets. Thus, the existing approaches that have long response time are not practical to a fast-evolving environment, where the servers and clients may move and the server and client sets may change as time goes by.

For instance, let us consider a military application where the location of each soldier and each military vehicle is tracked by GPS. Suppose there are medical workers, apparatus and instruments sufficient to set up 10 field hospitals, and the goal is to locate these field hospitals in a military area to serve as many soldiers as possible. As the battlefront changes over time, the locations of soldiers and vehicles will also change, and some field hospitals have to be relocated by military transport aircrafts.

Assume that there exists a coordination center, which is responsible for adjusting the location of a field hospital, if it finds that the hospital can no longer serve sufficient soldiers. Then, the center has to find the optimal location as soon as possible, since fast medical support is critical for the wounded soldiers. Instead of spending hours to run the existing algorithms to find an optimal location, it is more desirable to find a slightly less optimal location within a few seconds, so that the relocation of the hospital can be carried out in time. Thus, our goal is to find a near-optimal location efficiently.

To realize this goal, we propose an approximation algorithm called FILM that is able to return a small region, where each location is a sub-optimal one, within seconds to a couple of minutes, which is orders of magnitude faster than the state-of-the-art algorithm that returns an optimal one. We also consider the related problem of locating k new servers in order to maximize the total number of clients attracted by these servers "collectively". It is worth mentioning that [1] proposes to find k locations, each of which attracts as many clients as possible. However, since the k lo-

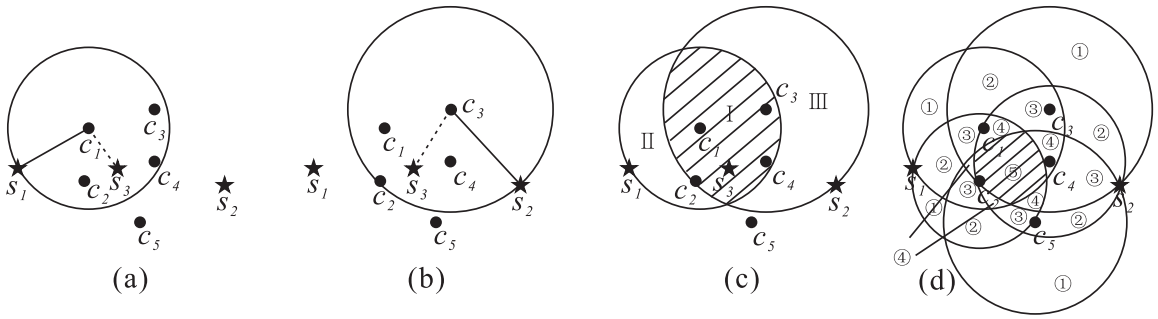


Figure 1: Illustrations of Nearest Location Circle (NLC)

cations are independently found in [1], they may share a lot of attracted clients and thus the total number of clients attracted "collectively" by these k locations may not be large. To solve this problem, we extend FILM to finding k influential locations that collectively attract as many clients as possible.

The rest of the paper is organized as follows: Section 2 formulates the problem of finding influential locations. Section 3 reviews the related work, with an emphasis on the state-of-the-art algorithm proposed recently. Our grid-based algorithm, FILM, is described in Section 4. Extensive experiments are conducted in Section 5 to evaluate the efficiency and effectiveness of FILM on both real and synthetic datasets. Finally, we conclude our paper in Section 6.

2. PROBLEM DEFINITION

2.1 Optimal-Location Query

We now use the example in Figure 1 to illustrate the idea of optimal-location queries. Figure 1(a) shows a spatial setting with a set of two servers (s_1 and s_2), and a set of five clients (c_1, c_2, c_3, c_4 and c_5). Suppose that we need to choose a location for a new server s_3. *Where should it go in order to attract as many clients as possible?* To answer this question, we first need to define the condition under which s_3 attracts client c_i as follows:

Definition 1 (ABUF). *The attraction buffer (ABUF) of a client $c \in C$, denoted ABUF(c), is the union of all the locations where a new server s can attract c.*

Therefore, s_3 attracts c_i if and only if its location is within $ABUF(c_i)$. Previous work assumes that a client always visits its nearest server, which we call the *nearest neighbor assumption* (or the *NN assumption*). Under the NN assumption, when the distance metric is L_2-norm (Euclidean distance), the ABUF of a client is actually a circle as defined below:

Definition 2 (NLC). *The nearest location circle (NLC) of a client $c \in C$, denoted NLC(c), is the circle centered at the location of c with radius $|c, NN(c)|$, where NN(c) is the nearest neighbor of c in the server set S, and $|\cdot|$ stands for the Euclidean distance.*

We illustrate the concept of NLC by our running example in Figure 1. Assume that currently $S = \{s_1, s_2\}$, and thus S does not contain the new server s_3 to set up. Therefore, s_3 is not involved in the computation of the NLC of a client. Figure 1(a) shows the NLC of c_1 (note that c_1 is closer to s_1 than to s_2). However, if s_3 is set up at the location as shown in Figure 1(a), it becomes the new nearest neighbor (NN) of c_1 and thus attracts c_1. This gives rise to the following observation:

Observation 1. *If a new server is set up at a location within the NLC of a client $c \in C$, the server will attract c.*

The same principle also holds when the NLC of c_3 is considered (Figure 1(b)). When we consider only two clients c_1 and c_3, we have the corresponding NLCs as shown in Figure 1(c). In this figure, we can see that region II is only covered by c_1's NLC and thus a new server there will only attract c_1 but not c_3. Similarly, a new server in region III will attract c_3 but not c_1. In the overlapping region I, it is obvious that a new server there will attract both c_1 and c_3. In the region not covered by any NLCs, a new server there will certainly attract no clients. If we assume that there are only two clients c_1 and c_3, the optimal location for s_3 should be within region I. The above discussion can be formalized in the following observation:

Observation 2. *If a new server is set up at a location which is covered by the greatest number of NLCs, the new server will attract the greatest number of clients.*

We conclude that an optimal-location query should return a location which is covered by the greatest number of NLCs. Figure 1(d) shows a more complicated scenario by considering all the clients, namely $c_1, c_2, ..., c_5$. Their NLCs divide the whole space into a number of partitions. In Figure 1(d), each partition is marked with a number denoting the number of NLCs covering the partition. This number corresponds to the total number of clients who have interest in visiting a new server set up in this partition. Given a location l in this partition, we name this number as the *influence value* of l. Obviously, any point in the shaded region in Figure 1(d) is an optimal location.

Now, we generalize the concept of NLC in Observations 1 and 2 to any ABUF, and define the *influence value* of a location l based on ABUF:

Definition 3 (Influence Value). *Given a location l, the influence value of l, denoted inf(l), is the number of clients $c \in C$ such that $l \in ABUF(c)$.*

Definition 4 (Optimal-Location Query). *Given a set S of servers and a set C of clients, the* optimal-location query *returns a location l such that $inf(l)$ is the maximum.*

With a straightforward extension we can also handle the more general case where each client $c \in C$ is associated with a weight $w(c)$. In this case, Definition 3 can be generalized as follows: given a location l, its influence value $inf(l) = \sum_{c \in C:\ l \in ABUF(c)} w(c)$, i.e., we compute the sum of the weights of the attracted clients rather than simply count the total number.

2.2 Finding Influential Location

Although optimal-location queries are useful in real life applications, it is expensive to compute the optimal location, which makes such queries not practical for fast-evolving environments. Another inherent limitation of such queries is that only one (optimal) location is returned as the answer, which is too restrictive. Thus, we propose to study a more general setting of the queries as follows: "finding k influential locations to locate k new servers so as to attract as many clients as possible". To our knowledge, this problem is still not formally addressed in the literature.

To support applications that require short response time, we propose an approximation algorithm called FILM, which is able to return a near-optimal location within seconds to a couple of minutes, and can support the top-k queries mentioned above.

The key to the design of FILM is a generalization of the concept of NLC. The traditional optimal-location queries studied in the previous work adopt NLC as the ABUF. However, the underlying assumption of the traditional optimal-location queries that a client always visits its nearest server (i.e., the NN assumption) may not hold in reality. For example, a restaurant that is 55 meters away from our home and serves better food is more likely to attract us, even if the nearest restaurant is 40 meters away, although we may be reluctant to go to a restaurant 500 meters away. This also holds for long distances. For example, for people living in the outskirt of a city who want to drive to a supermarket in the downtown area for shopping, a supermarket 5 miles away that sells more products is not less attractive than a supermarket 4 miles away.

In order to tolerate other close servers besides the NN, [1] proposes to set the radius of the NLC of a client c to $|c, \mathrm{NN}_k(c)|$, where $\mathrm{NN}_k(c)$ is the k-th nearest server of c, and k is a user-specified parameter. However, this radius relaxation still lacks flexibility: for one client, its second NN may be just a little farther away than its NN and the client is very likely to visit it, while for another client, its second NN may be times farther away than its NN, which deters the client from visiting it. We thus adopt a new NLC relaxation called *Relaxed NLC* (RNLC) defined as follows:

Definition 5 (RNLC). *The relaxed nearest location circle (RNLC) of a client c is the circle centered at the location of c with radius* $(1+\alpha) \cdot |c, NN(c)|$, *where* $\alpha > 0$.

The parameter α in Definition 5 is a user-specified parameter that describes how far a client is willing to visit a server. If it is allowed to be set to 0, RNLC is reduced to NLC. However, note that Definition 5 requires $\alpha > 0$. Intuitively, Definition 5 states that a client is willing to visit a server whose distance is at most $(1+\alpha)$ times longer than the distance between the client and its nearest server.

Each client $c_i \in C$ can have an RNLC parameter α_i specific to it, and to differentiate from the traditional optimal-location queries that adopt NLC as the ABUF, we name the optimal-location queries that adopt RNLC as the ABUF to be *Relaxed Optimal-Location Queries*:

Definition 6 (Relaxed Optimal-Location Query). *A relaxed optimal-location query is an optimal-location query given in Definition 4 that adopts RNLC as the ABUF.*

We develop a very efficient approximation algorithm (FILM) to the relaxed optimal-location queries. Although the parameter α can be different for different clients in the relaxed optimal-location queries, we use a fixed α for all clients and prove that as $\alpha \to 0^+$, the influence value of the result location of FILM converges to that of the traditional optimal-location query.

3. RELATED WORK

The optimal-location query was first studied by [2, 1]. [2] proposes an algorithm called *Arrangement*, whose time complexity is exponential to the number of clients, resulting in very poor scalability, while [1] proposes an efficient algorithm called *MaxOverlap* whose time complexity is polynomial to the number of clients. *MaxOverlap* is know to be the fastest algorithm for optimal-location queries. [1] reports that for a small dataset with 250 clients and 500 servers, the Arrangement algorithm runs for more than one day while MaxOverlap finishes the task within 0.1s.

The optimal-location query defined in [2, 1] is based on NLC instead of RNLC. In the following, we briefly introduce the underlying idea of MaxOverlap [1], as the following four steps:

- **Step 1 (NLC Construction):** For each client $c \in C$, find the NLC of c.
- **Step 2 (Intersection Point Computation):** For each pair of NLCs, find the intersection points between the boundaries of these two NLCs.
- **Step 3 (Point Query):** For each intersection point found in Step 2, find all NLCs covering this point.
- **Step 4 (Maximum Influence Value Finding):** find the intersection point which is covered by the greatest number of NLCs, and return the region which is the intersection of all those covering NLCs.

It is shown in [1] that the region returned by MaxOverlap corresponds to the optimal partition in which each location is an optimal one (e.g., the shaded region in Figure 1(d)).

Although MaxOverlap uses a pruning rule to avoid evaluating the intersection points that are guaranteed not to be optimal, it is still quite slow in practice. The time complexity of MaxOverlap is shown to be $O(|C| \log |S| + \ell^2 |C| + \ell |C| \log |C|)$ in [1], where ℓ is the average length of the overlapping lists of the clients. MaxOverlap is very expensive if ℓ is large, and since $\ell = O(|C|)$, the time cost of MaxOverlap is super-quadratic to the number of clients.

Unlike MaxOverlap that works on Euclidean distance, [6] uses Manhattan distance as the metric. In this case, the ABUF of a client is a square. [6] uses plane-sweep to evaluate the optimal-location query. Since [6] focuses on *constrained optimal-location query* that finds an optimal location within a query region rather than globally, it elaborates on the disk-based index to fetch only the ABUFs involved.

Note that MaxOverlap can easily be extended to handle the constrained optimal-location query, by filtering out the intersection points that are outside the query region.

Some recent work [4, 5] considers the capacity of each server. Other related work includes [8, 9, 10], all of which are based on the idea of *bichromatic reverse nearest neighbor* [7]. However, they find the most influential server among the existing servers, while we find an influential location for a new server, which is a fundamentally different problem from theirs.

4. FILM ALGORITHM

Although MaxOverlap [1] is the state-of-the-art algorithm for the evaluation of optimal-location queries, its efficiency is still far from satisfactory for most practical applications. This is because MaxOverlap needs to compute and evaluate a huge amount of intersection points between the boundaries of the NLCs. This observation reveals that, in order to find influential locations much more efficiently, we have to abandon the idea of checking the intersection points but to develop a new approach for finding such locations.

The idea of FILM is inspired by the way people solve the optimal-location query manually. If we are given a paper and a pencil, we will solve the optimal-location problem by drawing the NLCs of all clients and then find the most overlapped partition, as illustrated in Figure 1. Since the "paper" to draw on is a continuous space, and we need to find the most overlapped region/location within short time, we partition the "paper" space using grids and manipulate on the grid cells. FILM returns a grid cell in which each location is an influential one, as an approximation of the optimal location.

In the sequel, we first present in Section 4.1 our idea of FILM when there is only one grid partitioning imposed on the space. This paves the way of introducing the complete FILM algorithm in Section 4.2, where we show the necessity of using a hierarchy of multiple grids to improve the performance of finding an influential location.

4.1 Grid-based Algorithm

4.1.1 Basic Algorithm

Similar to MaxOverlap[1], the first step of FILM is also to obtain the NLCs of all the clients. To get the NLCs, we first bulk-load a balanced kd-tree on the server points in S, denoted as T_S^{kd}. Then, for each client $c \in C$, we find the server s nearest to c by an NN query on T_S^{kd} and get the NLC radius $|s, c|$. We adopt a kd-tree instead of an R*-tree since operations on kd-tree have time bounds. T_S^{kd} is freed from memory after we have computed all the NLCs, since it is no longer needed.

Algorithm 1 Basic Algorithm of FILM

```
1: for each client c ∈ C do
2:    Compute its NLC NLC(c)
3: end for
4: for each client c ∈ C do
5:    for each grid cell g that overlaps with NLC(c) do
6:       counter(g) ⟵ counter(g) + 1
7:    end for
8: end for
9: return arg max_g counter(g)
```

After getting all the NLCs, we "draw" them on the grid partitioning of the space as shown in Algorithm 1, where each grid cell is a small square and its side length is called the *grid side length*. A counter $counter(g)$ is attached to each grid cell g to record the number of NLCs overlapping with the cell, which is initialized to 0. After all the NLCs are drawn, the grid cell with the maximum counter value is returned. In the cases where the clients are weighted, Line 6 of Algorithm 1 becomes $counter(g) \longleftarrow counter(g) + w(c)$, where $w(c)$ is the weight of client c.

4.1.2 Analysis

In this subsection, we justify that Algorithm 1 returns a grid cell in which each location is an influential one, and as the *grid side length* approaches 0, the influence value of a location in the result cell approaches the maximum influence value.

To prove this, we can view Algorithm 1 as an approximation algorithm to the relaxed optimal-location query defined in Definition 6. We now present the following theorem:

THEOREM 1. *If a grid cell g overlaps with the NLC of client c whose radius $r \geq \delta\varepsilon$ (ε is the grid side length, $\delta > 1$), then any location in g is within the RNLC of c with $\alpha \geq \frac{\sqrt{2}}{\delta}$.*

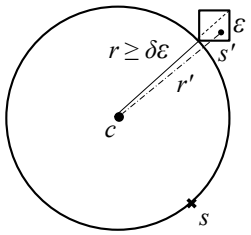


Figure 2: Illustration of Theorem 1

PROOF. To illustrate the idea of Theorem 1, Figure 2 shows the NLC of a client c, whose nearest server is s. Assume that the grid side length is ε and the radius of the NLC is $r \geq \delta\varepsilon$. Figure 2 is an extreme case for a grid cell g to overlap with the NLC of c. The point in g that is farthest from c is the upper right corner of g, denoted g_{ur}. It is obvious that, for any point s' in g, if we denote $d = |c, g_{ur}|$ and $d' = |c, s'|$, we have $d' \leq d$. Since $d = |c, g_{ur}| = r + \sqrt{2}\varepsilon$, we have $d' \leq r + \sqrt{2}\varepsilon$.

Note that Figure 2 is the ***extreme*** case, and for any grid cell g' closer to c than g (and hence overlaps with $NLC(c)$), we must also have $\forall s' \in g'$, $|c, s'| \leq r + \sqrt{2}\varepsilon \leq r + \sqrt{2}(\frac{r}{\delta}) = (1 + \frac{\sqrt{2}}{\delta})r$.

Therefore, if a grid cell g overlaps with the NLC of c, then g is within the RNLC of c with $\alpha \geq \frac{\sqrt{2}}{\delta}$ (see Definition 5), which proves Theorem 1. □

The parameter δ in Theorem 1 has a straightforward geometric meaning: for a grid with grid side length ε, $\delta\varepsilon$ defines the lower bound of the radius of any NLC "drawn" on it. δ is also important for the grid hierarchy described later.

Next, we describe the relationship of our new parameter δ with the parameter α in Definition 5: Given the user-specified RNLC parameter α, we require $\delta \leq \frac{\sqrt{2}}{\alpha}$ according to Theorem 1. On the other hand, smaller grid side length

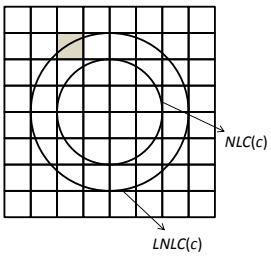


Figure 3: Underestimated Cell

ε leads to better approximation, and thus we want $\delta \leq \frac{r}{\varepsilon}$ to be as large as possible. As a result, the optimal setting is $\boldsymbol{\delta = \frac{\sqrt{2}}{\alpha}}$.

Although α is a more intuitive parameter that indicates the maximum distance a user can tolerate, we use δ as the user-specified parameter instead, due to its more straightforward geometric meaning. In fact, given α, we can always set $\delta = \frac{\sqrt{2}}{\alpha}$, and thus setting both parameters are essentially equivalent.

If a grid cell g does not overlap with the NLC of a client c, we do not add $counter(g)$, although g may overlap with the RNLC of c (e.g., the gray cell in Figure 3). Therefore, the counter value of a grid cell g computed by our approach is a conservative estimation of the number of clients that a new server in g attracts, for the relaxed optimal-location query.

However, as $\delta \to +\infty$ ($\alpha = \frac{\sqrt{2}}{\delta} \to 0^+$), the boundary of RNLC approaches NLC and the probability that the above underestimation happens for a grid cell approaches 0. Therefore, we have the following theorem:

THEOREM 2. *The counter value of a grid cell g is a conservative estimation of the number of clients that a new server in g attracts, and this value approaches the exact value as $\delta \to +\infty$.*

Thus, the influence value of a location in the result cell of Algorithm 1 approaches the maximum influence value, as $\delta \to +\infty$.

4.1.3 Implementation Details

We first introduce our grid structure as shown in Figure 4. The whole space is divided into a number of grid cells with grid side length ε. Each cell is indexed by two integers $\langle i, j \rangle$, such that its lower left vertex coordinate is $(i\varepsilon, j\varepsilon)$. For example, Cells I, II, III, IV and V in Figure 4 are indexed as $\langle 0, 0 \rangle$, $\langle -1, 0 \rangle$, $\langle 1, 0 \rangle$, $\langle 0, 1 \rangle$ and $\langle -4, -4 \rangle$, respectively.

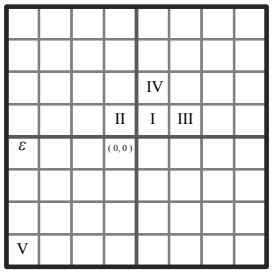


Figure 4: Grid Cell Indexing with Coordinates

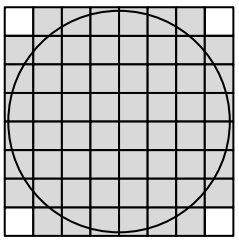


Figure 5: Grid Cells with Counters Added

Line 5 in Algorithm 1 aims at obtaining all the grid cells overlapping with an NLC. To find these cells efficiently, given an NLC with center $c = (x, y)$ and radius r, we define the *candidate cell set* of the NLC to be the set of cells $\{\langle i, j \rangle \mid \lfloor x - r \rfloor < i < \lfloor x + r \rfloor, \lfloor y - r \rfloor < j < \lfloor y + r \rfloor\}$. The round down operation $\lfloor x \rfloor$ is used because we index each cell by its lower left vertex.

For example, all the cells in Figure 5 compose the *candidate cell set* of the NLC in the figure. To see how to find this set, we note that the NLC has center $(0, 0)$ and radius r such that $3\varepsilon < r < 4\varepsilon$, and therefore the *candidate cell set* of the NLC is computed as $\{\langle i, j \rangle \mid \lfloor 0 - r \rfloor < i < \lfloor 0 + r \rfloor, \lfloor 0 - r \rfloor < j < \lfloor 0 + r \rfloor\} = \{\langle i, j \rangle \mid -4 < i < 3, -4 < j < 3\}$.

However, only the gray cells in Figure 5 are those that really overlap with the NLC. Therefore, after obtaining the *candidate cell set* of an NLC, we need a filtering step to find all the cells overlapping with the NLC. For each cell $g = \langle i, j \rangle$ in the *candidate cell set*, we determine whether it overlaps with an NLC with center $c = (x, y)$ and radius r, by checking whether $MINDIST(g, c) \leq r$, where *MINDIST* is defined similar to [11]:

$$MINDIST(g, c) = \sqrt{(x_g - x)^2 + (y_g - y)^2}$$

$$x_g = \begin{cases} i\varepsilon, \text{ if } i\varepsilon > x \\ i\varepsilon + \varepsilon, \text{ if } i\varepsilon + \varepsilon < x \\ x, \text{ otherwise} \end{cases}, \quad y_g = \begin{cases} j\varepsilon, \text{ if } j\varepsilon > y \\ j\varepsilon + \varepsilon, \text{ if } j\varepsilon + \varepsilon < y \\ y, \text{ otherwise} \end{cases}$$

Note that each grid cell can be represented as a key-value pair, where the key is the cell index $\langle i, j \rangle$ and the value is the counter of the cell. Since it may be extremely memory-inefficient to materialize all the cells in the grid (especially if the cells are very small compared to the whole space involved, or if most of the space is not covered by any NLC), we organize the cells by a balanced search tree, where for a cell $\langle i, j \rangle$, i serves as the primary key and j serves as the secondary key.

Only those cells that overlap with at least one NLC are stored in the tree. Algorithm 2 shows the operations done

Algorithm 2 Grid Cell Processing

```
1: if g overlaps with NLC then
2:    if g ∈ GRID then
3:       counter(g) ⟵ counter(g) + 1
4:    else
5:       Create grid cell g
6:       counter(g) ⟵ 1
7:       Insert g into GRID
8:    end if
9: end if
```

for each cell g in the already obtained *candidate cell set* and the NLC NLC in question, assuming $GRID$ to be the binary search tree storing the cells. Line 1 of Algorithm 2 is checked using *MINDIST*.

Although there are many efficient realizations of a balanced search tree (e.g., [12]), we simply use the "map" container from the C++ STL library, which is actually a red-black tree.

4.2 Adaptive Gird Hierarchy

To see the necessity of a hierarchy of multiple grid structures, let us first analyze the characteristics of NLCs on real datasets.

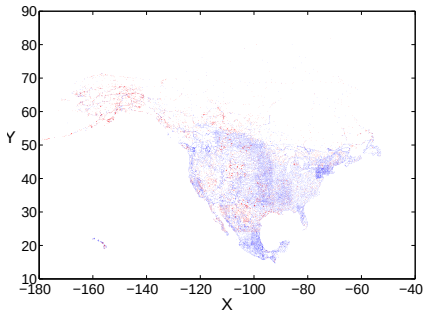


Figure 6: North American Dataset

Figure 6 shows the North American dataset from [14], where the 9203 red points are cultural landmarks and the 24493 blue points are populated places. We treat the cultural landmarks as servers and the populated places as clients, and assume that people tend to visit the cultural landmarks close to them. After all the NLCs are computed, we find that the radii of the NLCs span several orders of magnitude. For example, the smallest NLC of the North American dataset has radius 1.23×10^{-3} while the largest is 9.505. We find similar patterns in all the datasets we use.

As a result, if we use only one grid, of which the grid side length is suitable for the smallest NLC, "drawing" larger NLCs would involve quite a lot of cells, which degrades the performance. Thus, we need to adapt the grid structure to the NLC size automatically, which results in the grid hierarchy of FILM.

To achieve reasonable approximation quality, the grid cells we choose should be sufficiently small compared with the size of an NLC "drawn" on it. Therefore, we set a lower-bound to the size of any NLC handled by a grid as follows, where δ is the same parameter as in Theorem 1 and is the user-specified parameter of our grid hierarchy:

DEFINITION 7. *Given a grid structure with grid side length ε, any NLC that is "drawn" on it should have radius $r \geq \delta \cdot \varepsilon$.*

However, a grid structure of too fine granularity is not necessary for large NLCs, and will degrade the evaluation performance. Thus, we also set an upper-bound to the size of any NLC handled by a grid:

DEFINITION 8. *Given a grid structure with grid side length ε, any NLC that is "drawn" on it should have radius $r < \delta^2 \cdot \varepsilon$.*

To sum up, for a grid structure whose grid side length is ε, we only draw on it those NLCs with radius r satisfying $\delta \cdot \varepsilon \leq r < \delta^2 \cdot \varepsilon$.

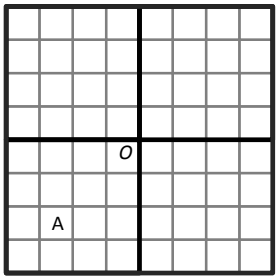


Figure 7: Two Consecutive Grids in the Hierarchy

FILM uses a set of grids such that consecutive grids have grid side lengths being different by a factor of δ. Figure 7 shows an example space with 2 consecutive grid structures, where $\delta = 4$ and the smaller grid side length ε_1 is of the lower level grid, and the large grid cells drawn with bold lines whose grid side length is $\varepsilon_2 = 4\varepsilon_1$ are of the higher level grid. Note that low level grids are just refinements of the upper level ones, e.g., each large grid in Figure 7 contains exactly $\delta^2 = 16$ small grids.

According to our grid hierarchy, each NLC is handled by exactly one grid structure in the hierarchy. For example, in Figure 7, the grid with grid side length ε_1 can only handle NLCs with radius $4\varepsilon_1 \leq r < 16\varepsilon_1$, and the grid with grid side length ε_2 can only handle NLCs with radius $4\varepsilon_2 \leq r < 16\varepsilon_2$, i.e., $16\varepsilon_1 \leq r < 64\varepsilon_1$.

4.2.1 Algorithm for Mining k Influential Locations

The algorithm of FILM consists of 2 steps: 1)build grid hierarchy from NLCs; and 2)evaluate the influence value estimation of each grid cell and pick the maximum one.

The first step is detailed as follows: The obtained NLCs are first sorted in non-decreasing order of the radius. Then a pass through the sorted list allocates the NLCs to the corresponding grids based on the bounds in Definition 7 and 8. The result of the first step is shown in Figure 8, where we obtain a list of grids (i.e., $GList$), and each grid element in $GList$ records: 1)len: the grid side length, 2)$[start, end]$: the set of NLCs allocated to the grid, and 3)$tree$: the binary search tree holding the grid cells of the grid, which is initialized to an empty tree.

An important issue is the choice of the value of the grid side length ε_{min} of the lowest level grid. We choose $\varepsilon_{min} = r_{min}/\delta$ where r_{min} is the radius value of the smallest NLC. In this way, the lowest level grid handles the NLCs with radius $\delta\varepsilon_{min} \leq r < \delta\varepsilon_{min}^2$, i.e., $r_{min} \leq r < \varepsilon_{min} \cdot r_{min}$.

After obtaining $GList$, the NLCs allocated to each grid are "drawn" on that grid as described in Section 4.1, and the counter information of the grid cells are stored in the binary search tree of that grid.

So far, the grid hierarchy has been constructed from the NLCs, and we are ready to evaluate the influence value estimation of each grid cell using our grid hierarchy —— a basic operation for the second step of FILM. The key to this operation is the following observation of our grid hierarchy:

OBSERVATION 3. *Given a grid cell $g = \langle i, j \rangle$ with grid side length ε, we can identify the cell in a upper level grid with grid side length ε' that contains g, as $g' = \langle \lfloor i\varepsilon/\varepsilon' \rfloor, \lfloor j\varepsilon/\varepsilon' \rfloor \rangle$.*

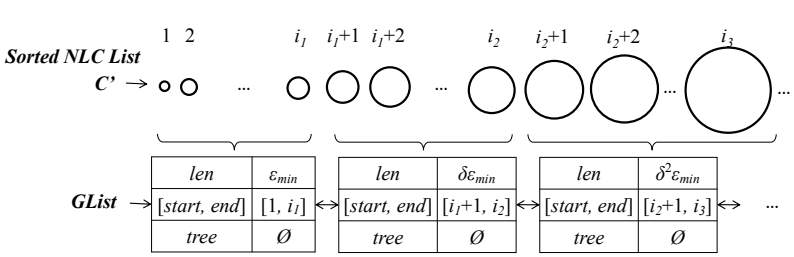


Figure 8: NLC Allocation to Grids

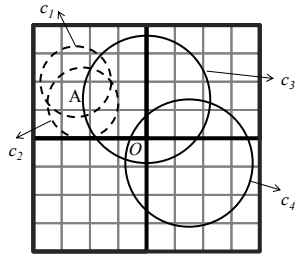


Figure 9: Estimation Integration Among Grids

We call g' the *covering cell* of g in this case. For example, the covering cell of Cell $A = \langle -3,\ -3\rangle$ in Figure 7 is the lower left large cell: $\langle \lfloor -3\varepsilon_1/\varepsilon_2 \rfloor, \lfloor -3\varepsilon_1/\varepsilon_2 \rfloor\rangle = \langle \lfloor -3/4 \rfloor, \lfloor -3/4 \rfloor\rangle = \langle -1,\ -1\rangle$. Therefore, our structure hierarchy allows us to find the covering cell of a grid cell in an upper level grid structure in O(1) time.

Since each grid only handles the NLCs of a subset $C' \subseteq C$ of the clients, the counter value of a grid cell g is just a conservative influence value estimation on this subset C'.

Figure 9 shows some NLCs on the grid hierarchy of Figure 7, where the two small NLCs c_1 and c_2 are drawn on the lower level grid, and the two large NLCs c_3 and c_4 are drawn on the higher level grid. After the grid hierarchy is built, the counter value of Cell A is 2 due to its overlap with c_1 and c_2, and the counter value of the upper left large cell (denoted as g_{ul}) is also 2 due to its overlap with c_3 and c_4.

The counter of Cell A does not take c_3 and c_4 into consideration. Note that c_4 does not even overlap with Grid A. However, the clients of c_3 and c_4 can also be attracted by a new server in Cell A. This is because c_3 and c_4 overlap with g_{ul}, the *covering cell* of Cell A, and according to Theorem 1, all the locations in g_{ul} are in the RNLCs of c_3 and c_4, including all the locations in Cell A.

Therefore, to get a conservative influence value estimation for cell g in terms of the whole client set C (for the relaxed optimal-location query), we need to sum up the counter values of all the covering cells of g in the upper level grids, besides the counter value of g. For example, the conservative estimation for Cell A in Figure 9 is the sum of its counter value and the counter value of g_{ul}, which is 2+2=4. The covering cell of g in an upper level grid can be identified in $O(1)$ using Observation 3. Algorithm 3 shows how the influence value estimation $\widehat{inf}(g)$ of cell g is computed.

Algorithm 3 Grid Cell Influence Value Estimation

```
1: inf(g) ⟵ counter(g)
2: for each grid GRID of level higher than that of g do
3:    Find the covering cell g' of g in GRID
4:    if g' ∈ GRID then
5:       inf(g) ⟵ inf(g) + counter(g')
6:    end if
7: end for
```

Algorithm 3 is executed on all the cells of all the grids in the hierarchy and the cell with maximum influence value estimation is returned.

However, a more general problem is to find k influential locations that collectively attracts the most clients. The exact problem can be viewed as a special case of the *maximum coverage problem*, where each intersection point between NLC boundaries corresponds to a subset of clients C that a new server there can attract, and the objective is to choose k such points(subsets) that covers as many clients in C as possible.

Although the *maximum coverage problem* is NP-hard [13], the greedy algorithm of choosing a subset which contains the largest number of uncovered elements at each stage, achieves an approximation ratio of $1 - \frac{1}{e}$ [13]. Inspired by this, FILM can be easily extended to find k influential grid cells by performing the following 2 steps at each stage, after the previous cell g_{pre} is picked: 1)find the NLCs overlapping with g_{pre}, and cancel them out from the grid hierarchy, and 2)pick the cell with maximum influence value estimation from the grid hierarchy as the next result cell. Algorithm 4 shows how an NLC NLC is canceled out from the the grid hierarchy, where we assume NLC is allocated to grid GD_{NLC}.

Algorithm 4 Cancel NLC out from the Grid Hierarchy

```
1: for each grid GD of level not lower than GD_NLC do
2:    Find the candidate cell set of NLC in GD
3:    Pick the cells overlapping with NLC into set CELLS
4:    for each cell g ∈ CELLS do
5:       counter(g) ⟵ counter(g) − 1
6:    end for
7: end for
```

Complexity Analysis: For NLC computation, it takes $O(|S| \log |S|)$ to construct the server kd-tree, and then for each client, an NN query is issued on the tree, with a total time of $O(|C| \log |S|)$. According to Definition 8, the *candidate cell set* size of any NLC is upperbounded by $\left(2 \cdot \frac{\delta^2 \cdot \varepsilon}{\varepsilon}\right)^2 = 4 \cdot \delta^4$. Therefore, for all $|C|$ NLCs, there are at most $n = 4 \cdot \delta^4 |C|$ cells stored in our grid hierarchy. For each of the n grids, we need to estimate its influence value using Algorithm 3, the time of which is bounded by $O(l \log n)$, where l is the number of grids in the hierarchy (typically around 6). To sum up, the time complexity of FILM is $O(|S| \log |S| + |C| \log |S| + n \cdot l \log n) = O(|S| \log |S| + |C| \log |S| + \delta^4 (\log \delta^4) l |C| + \delta^4 |C| l \log |C|)$, which is $O(|C| \log |C|)$ to the number of clients, much better than the super-quadratic time cost of MaxOverlap[1].

5. EXPERIMENTS

We have conducted extensive experiments on an AMD Opteron-based Linux CPU Server with 4×AMD Opteron 844 (1.8GHz) CPU and 8GB memory. We implemented

Table 1: Experimental Results on Real Datasets

		MaxOverlap$_{NLC}$		MaxOverlap$_{RNLC}$		FILM				
S	C	Exec Time	Influence	Exec Time	Influence	Exec Time	Inf$_{RNLC}$	Ratio$_{RNLC}$	Inf$_{NLC}$	Ratio$_{NLC}$
NA$_{cl}$	NA$_{pp}$	374.63 s	354	866.98 s	627	5.375 s	395	0.63	332	0.938
CA	GR	4379.65 s	1389	3685.37 s	1558	6.305 s	1460	0.937	1296	0.933
CA	GM	351916.73 s	3872	659814.0 s	5640	6.92 s	4369	0.775	3820	0.987

FILM in C++, and used the program of MaxOverlap provided by the authors of [1] for experimental comparison.[1]

The real datasets we used are obtained from R-tree Portal[14], which are summarized in Table 2. NA$_{pp}$ and NA$_{cl}$ contain 2D points representing the populated places and the cultural landmarks in North American, respectively. We also used the pre-processed real datasets adopted by [1], i.e., CA, GR and GM that contain 2D points representing geometric locations in California, Greece and Germany, respectively. On each dataset, we report our experimental results averaged over 10 runs.

Table 2: Summary of Real Datasets

S	$\|S\|$	C	$\|C\|$
NA$_{cl}$	9203	NA$_{pp}$	24493
CA	62556	GR	23268
CA	62556	GM	36334

As for the synthetic datasets, for each parameter configuration, we generated 5 independent sets of client and server points, and ran the program twice on each set. The reported results are averaged over the 10 runs.

Since we find that the experimental results on weighted clients are similar to those for the unweighted case, we only report the latter.

5.1 Experiments on Real Datasets

Table 1 shows our experimental results on the three datasets in Table 2 when the gird hierarchy parameter $\delta = 4$ (i.e., $\alpha = \frac{\sqrt{2}}{4}$). We denote by MaxOverlap$_{NLC}$ the original MaxOverlap algorithm, and MaxOverlap$_{RNLC}$ the modified MaxOverlap algorithm which uses RNLCs by expanding the radii of NLCs by $\alpha = \frac{\sqrt{2}}{\delta}$.

For FILM, we use inf_{RNLC} to denote the influence value estimation of the result cell, and define $ratio_{RNLC}$ as the ratio of inf_{RNLC} to the influence value of the result of MaxOverlap$_{RNLC}$, which measures the approximation quality of FILM to the relaxed optimal-location query.

To measure the result quality of FILM for the exact optimal-location query, we define inf_{NLC} to be the influence value of the center of the result cell of FILM. Since a user can pick any location in the result cell, we use the cell center to evaluate the result quality for simplicity.

Note that the center may not be the best choice, since many NLCs may just overlap with the boundary part of the cell. However, even in this case, their corresponding clients are still very likely to visit a server in the center since it is just a little farther than their nearest server, which is not considered by the conservative measure inf_{NLC}. We use the ratio of inf_{NLC} to the influence value of the result of MaxOverlap$_{NLC}$ (i.e., the maximum influence) to evaluate the optimality of the result of FILM.

[1] http://www.cse.ust.hk/∼raywong/code/maxRNN.zip

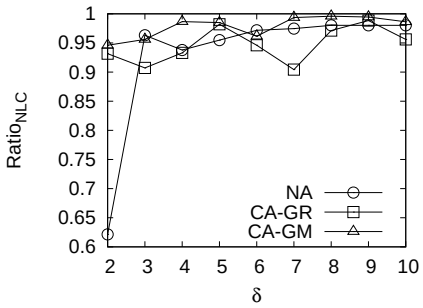


Figure 10: Approximation Quality on Real Datasets

From Table 1 we can see that the execution time of MaxOverlap is very sensitive to the data distribution. For example, MaxOverlap$_{NLC}$ takes 4379.65s to compute the optimal location on CA–GR, but 351916.73s on CA–GM (over 80 times), despite the fact that the number of data points in GM is only 1.56 times that of GR. On the other hand, the execution time of FILM is very short and stable.

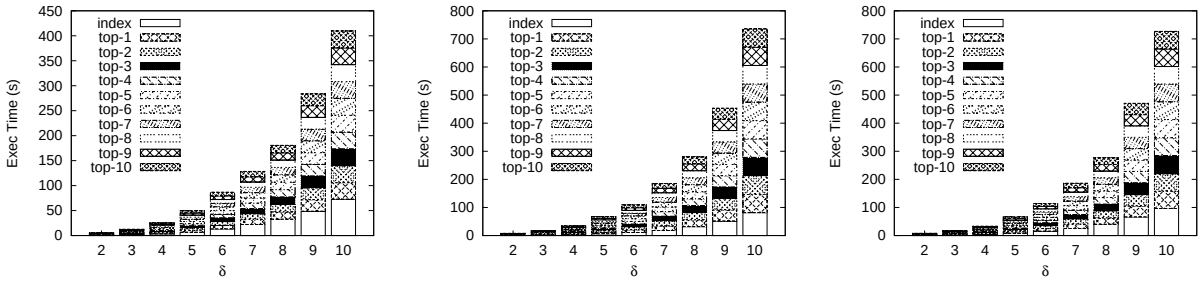
Theorem 2 only guarantees that FILM will be a good approximation algorithm for MaxOverlap$_{RNLC}$ when δ is large. Thus, when $\delta = 4$ as in Table 1, $ratio_{RNLC}$ may not be close to 1 for some datasets (e.g., NA$_{cl}$-NA$_{pp}$). However, from $ratio_{NLC}$ we can see that FILM already approximates MaxOverlap$_{NLC}$ well when $\delta = 4$, i.e., the center of the result cell of FILM is already a quite influential location when NLC is adopted as the ABUF.

Since our goal is to approximate MaxOverlap$_{NLC}$ efficiently rather than to approximate MaxOverlap$_{RNLC}$ for arbitrary δ, FILM works well in this sense and therefore we will focus on $ratio_{NLC}$ in the following discussion. As δ becomes larger, both $ratio_{RNLC}$ and $ratio_{NLC}$ approaches 1, but we cannot set δ too large because of the factor $O(\delta^4)$ in the time and space cost of FILM.

Fortunately, δ does not have to be very large to ensure good approximation quality. Figure 10 shows that Ratio$_{NLC}$ approaches 1 as δ increases, and $\delta = 10$ is already sufficient for FILM to achieve over 98% of the maximum influence value.

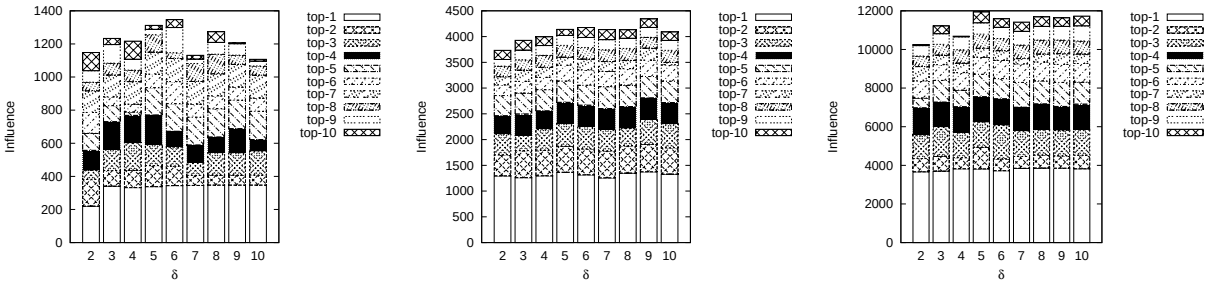
There is a natural measurement of the memory usage of FILM, i.e., the total number of cells stored in the grid hierarchy (denoted N_{cell}). Since each grid cell is composed of a 2-integer index $\langle i, j \rangle$ and a counter, which consumes 12 bytes in a typical computer, the space cost of FILM can be approximated as $12 \times N_{cell}$ bytes.

Figure 13 shows the memory usage of FILM on the 3 real datasets, where we can see that N_{cell} increases fast with the increment of δ. Despite of this fact, a computer with gigabytes of memory like the one we use can easily handle FILM with δ up to 10, which is already sufficient to ensure good approximation.



(a) NA (b) CA-GR (c) CA-GM

Figure 11: Execution Time on Real Datasets



(a) NA (b) CA-GR (c) CA-GM

Figure 12: Influence Value on Real Datasets

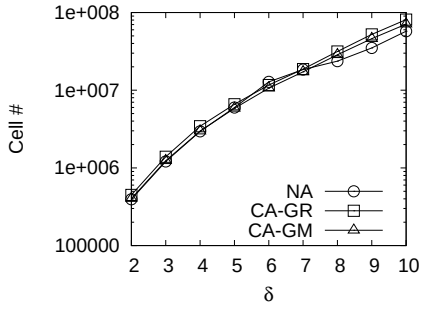


Figure 13: Memory Usage on Real Datasets

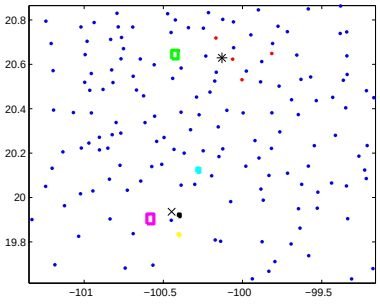


Figure 14: Top-1 Cells on North American Dataset

Figure 11 shows the running time of FILM on the 3 real datasets, where "index" denotes the time to construct the grid hierarchy, and "top-k" denotes the time to cancel out the $(k-1)$-th result cell (if $k > 1$) and find the k-th result cell from the grid hierarchy. We can see that the running time of FILM increases as δ increases, and that it takes FILM seconds to tens of seconds to return an influential cell depending on δ, which is considerably faster than MaxOverlap.

Figure 12 shows the result of FILM that finds k influential grid cells greedily on the 3 real datasets, where "top-k" denotes the number of new clients attracted by a server at the center of the k-th result cell. The number of clients attracted by the servers at the centers of the top-k cells increases with the increment of δ, although the influence value fluctuates at around $\delta = 6$. It is possible that if we randomly pick a location in each result cell rather than the cell center, there would be less fluctuation.

Figure 14 shows our results on NA_{cl}–NA_{pp} shown in Figure 6, where the red points are servers, the blue points are clients, the 'x' point is the intersection point returned by Max-Overlap$_{NLC}$, the '*' point is by Max-Overlap$_{RNLC}$, the green cell is the result of FILM when $\delta = 2$, the cyan cell is when $\delta = 4$, the magenta cell is when $\delta = 6$, the yellow cell when $\delta = 8$, and the black cell is when $\delta = 10$. We can see that the top-1 cell approaches 'x' from '*' as δ increases, which is also observed on CA-GR and CA-GM. Note that Figure 14 covers only 0.02% of the whole space in Figure 6.

5.2 Experiments on Synthetic Datasets

Since the effect of δ on the synthetic datasets is similar to that on the real datasets, we only report the results when $\delta = 4$. We generated the synthetic datasets by Uniform distribution or Gaussian distribution, and did two sets of experiments on them: 1)$|C| = |S|$, and 2) varying $|C|/|S|$.

In the first set of experiments, we set $|C| = |S| = 10^2$, 10^3, 10^4 and 10^5. The results show that it takes FILM more time on the more biased datasets of Gaussian distribution. For $|C| = |S|$, both the running time and memory usage of FILM are linear to the dataset size, and so is the running time of MaxOverlap.

We only show our experimental results for the second set of experiments in this subsection to save space. Since $|S|$ and $|C|$ are comparable on the real datasets we use, and in the real life $|C|$ tends to be much larger than $|S|$ for a server to serve many clients, we study the performance of FILM and MaxOverlap with varying $|C|/|S|$.

We set $|C| = 100000$ but use various $|S|$, the results of which are shown in Figures 15 and 16, where "FILM_g" denotes the results of FILM on datasets with Gaussian distribution, "RNLC_u" denotes the results of Max-Overlap$_{RNLC}$ on datasets with Unform distribution, "NLC_g" denotes the results of Max-Overlap$_{NLC}$ on datasets with Gaussian distribution, and so forth.

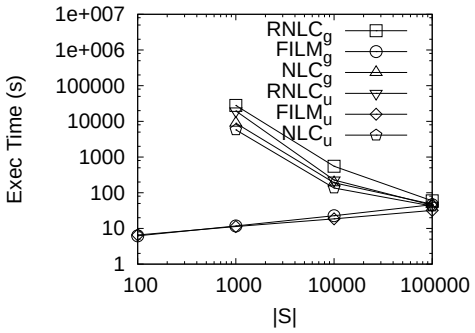


Figure 15: Execution Time on Synthetic Datasets

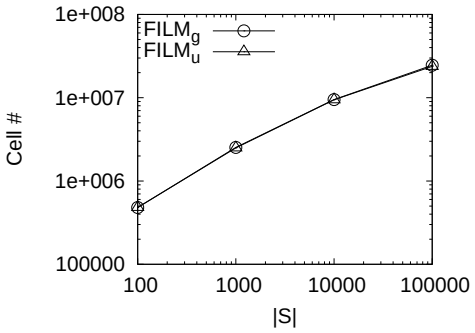


Figure 16: Memory Usage on Synthetic Datasets

We can see from Figure 15 that as $|C|/|S|$ becomes larger, Maxoverlap becomes much more expensive, while the time cost of FILM is insensitive to $|C|/|S|$. For example, when $|C|/|S| = 10$ (i.e., |S|=10000), Maxoverlap already takes hundreds of seconds, which is even worse as $|S|/|C|$ becomes larger (e.g., Maxoverlap takes over 28000s on "RNLC_g" when |S|=1000). When $|S| = 100$, MaxOverlap shows no trend to end after running for a week on any generated dataset of this configuration, and therefore these values are missing. The reason that large $|C|/|S|$ is expensive for MaxOverlap is that there are more intersection points to evaluate, since each NLC tends to be larger. On the other hand, FILM is even faster for small $|S|$ although the NLCs become larger, thanks to our grid structure that is adaptive to the NLC size.

6. CONCLUSION

We designed an efficient influential location miner called FILM, which returns a small grid cell in which all locations have an influence guarantee. In contrast to the existing approaches that return precisely an optimal location at the expense of long running time, our approach returns near-optimal locations in considerably less time. Thus, our approach is practical for many time-critical applications that require short response time of finding influential locations.

7. ACKNOWLEDGEMENTS

This work is partially supported by RGC GRF under grant number HKUST 617610.

8. REFERENCES

[1] R. C.-W. Wong, T. Ozsu, P. S. Yu, and A. W.-C. Fu. Efficient method for maximizing bichromatic reverse nearest neighbor. In *VLDB*, 2009.

[2] S. Cabello, J. M. Diaz-Banex, S. Langerman, C. Seara and I. Ventura. Reverse facility location problems. In *Canadian Conference on Computational Geometry*, 2005.

[3] N. Beckmann, H. Kriegel, R. Schneider and B. Seeger. The R*-Tree: An efficient and robust access method for points and rectangles. In *SIGMOD*, 1990.

[4] L. H. U, M. L. Yiu, K. Mouratidis, and N. Mamoulis. Capacity constrained assignment in spatial databases. In *SIGMOD*, 2008.

[5] R. C.-W. Wong, Y. Tao, A. W.-C. Fu, and X. Xiao. On efficient spatial matching. In *VLDB*, 2007.

[6] Y. Du, D. Zhang, and T. Xia. The optimal-location query. In *SSTD*, 2005.

[7] I. Stanoi, M. Riedewald, D. Agrawal, and A. E. Abbadi. Discovery of influence sets in frequently updated databases. In *VLDB*, 2001.

[8] B. C. Tansel, R. L. Francis, and T. Lowe. Location on networks: A survey. In *Management Science*, 1983.

[9] J. Krarup and P. M. Pruzan. The simple plant location problem: Survey and synthesis. In *European Journal of Operational Research*, 1983.

[10] T. Xia, D. Zhang, E. Kanoulas, and Y. Du. On computing top-t most influential spatial sites. In *VLDB*, 2005.

[11] T. Emrich, H.-P. Kriegel, P. Kröger, M. Renz, and A. Züfle. Boosting spatial pruning: on optimal pruning of MBRs. In *SIGMOD*, 2010.

[12] C. Kim, J. Chhugani, N. Satish, E. Sedlar, A. Nguyen, T. Kaldewey, V. Lee, S. Brandt, and P. Dubey. FAST: fast architecture sensitive tree search on modern CPUs and GPUs. In *SIGMOD*, 2010.

[13] D. S. Hochbaum. Approximating covering and packing problems: Set cover, vertex cover, independent set, and related problems. In *Approximation algorithms for NP-hard problems*, PWS Publishing Company, Boston, pp. 94-143, 1997.

[14] http://www.rtreeportal.org