# Collective-*k* Optimal Location Selection

Fangshu Chen[1]**, Huaizhong Lin[1] Jianzhong Qi[2], Pengfei Li[1], and Yunjun Gao[1]

[1] College of Computer Science and Technology, Zhejiang University, P.R. China
`youyou_chen@foxmail.com, linhz@zju.edu.cn, lpff1218@sina.com, gaoyj@zju.edu.cn`

[2] School of Computing and Information Systems, University of Melbourne, Australia
`jianzhong.qi@unimelb.edu.au`

**Abstract.** We study a novel location optimization problem, the *Collective-k Optimal Location Selection* (C$k$OLS) problem. This problem finds $k$ regions such that setting up $k$ service sites, one in each region, collectively attracts the maximum number of customers by proximity. The problem generalizes the traditional influence maximizing location selection problem from searching for one optimal region to $k$ regions. This increases the complexity of the problem. We prove that the C$k$OLS problem is NP-hard, and propose both precise and approximate algorithms to solve this problem. The precise algorithm uses a brute-force search with a pruning strategy. The approximate algorithm adopts a clustering-based approach, which restricts the combinational search space within representative regions of clusters, and has a bounded approximation ratio. Extensive experiments show that the approximate algorithm is effective and efficient, and its output results are close to optimal.

**Keywords:** BRNN, $k$ optimal location, clustering

## 1 Introduction

Given a set of service sites $P$ and a set of customer points $O$, for a service site $p \in P$, a *Bichromatic Reverse Nearest Neighbor* (BRNN) query finds all customer points $o \in O$ whose nearest neighbor in $P$ is $p$. The set of BRNN points of $p$ is called the *influence set* of $p$, and the cardinality of this set is called the *influence value* of $p$. Influence maximization has been a common goal in traditional location optimization problems. A basic problem is the MaxBRNN problem [2][3][4][7][8][17][18][19], which finds a region $S$ in which all the points have the maximum influence value, i.e., setting up a new service site within this region will attract the maximum number of customers by proximity. Here, a point is said to attract a customer if a service site set up at this point will become the new nearest neighbor of the customer.

We generalize the MaxBRNN problem to find $k$ regions such that setting up $k$ new service sites, one in each region, collectively attract the maximum number of

** This work is partially done when Fangshu is visiting the University of Melbourne.

customers by proximity. We call our generalized problem the *Collective-k Optimal Location Selection* (C$k$OLS) problem. The distinctive feature of this problem is that it optimizes the combined impact of $k$ selected points rather than the impact of any single point. As a result, we need to check every combination of $k$ regions to find the optimal result, which leads to a much higher computation complexity than that of the traditional MaxBRNN problem. Although there are existing algorithms for MaxBRNN [4][7][8][17][18][19], a straightforward adaptation of these solutions is inefficient for C$k$OLS because there are too many region combinations to be considered.

Meanwhile, C$k$OLS has many real life applications in service site planning and emergency scheduling. For example, consider a food truck chain which wishes to deploy a number of food trucks in the city CBD. It is preferable to deploy the food trucks within the regions which collectively attract the largest number of customers possible.

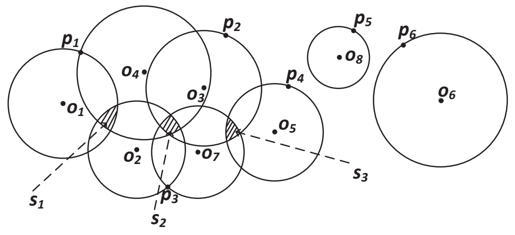


Fig. 1: An example of C$k$OLS

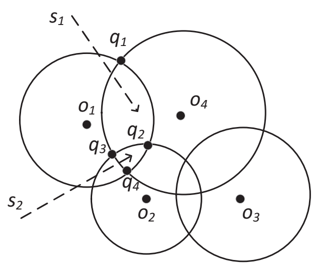


Fig. 2: An example of regions covered by NLCs

To define and compute the optimal regions, we make use of a concept called the *Nearest Location Circle* (NLC) [18]. An NLC is a circle centered at a customer point $o$ with the radius being the distance from $o$ to its nearest service site. For any customer point $o$, if we build a new service site $p$ outside the NLC of $o$, $p$ cannot attract $o$ for that there already exists a service site closer to $o$. As a result, to attract more customer points, the new service site should be located within as many NLCs as possible. The objective of C$k$OLS then becomes finding $k$ regions covered by the maximum number of NLCs.

For example, in Fig. 1, there are 8 customer points denoted by $o_1, o_2, ..., o_8$ and 6 existing service sites denoted by $p_1, p_2, ..., p_6$. The NLC of each customer is represented by the circle enclosing the customer point. The NLCs partition the space into regions. Three shaded regions $s_1, s_2$, and $s_3$ are covered by multiple NLCs. Region $s_2$ is covered by 4 NLCs. A new service site in $s_2$ will attract the 4 corresponding customers $\{o_2, o_3, o_4, o_7\}$. Similarly, each of regions $s_1$ and $s_3$ are covered by 3 NLCs and a new service site in them will attract 3 customers. Assuming that we want to build 2 new service sites in 2 different regions. The optimal result in this case is $\{s_1, s_3\}$ which attracts 6 customer points together:

$\{o_1, o_2, o_3, o_4, o_5, o_7\}$. In comparison, the combination $\{s_1, s_2\}$ attracts 5 customer points, $\{o_1, o_2, o_3, o_4, o_7\}$; and $\{s_2, s_3\}$ also attracts 5 customer points. As we can see from Fig. 1, the single optimal region is $s_2$ (attracts 4 customer points which is more than any other regions). This region, however, is not part of the optimal regions for the C$k$OLS problem when $k = 2$.

To solve the C$k$OLS problem, we propose two algorithms. The first algorithm is a precise algorithm. It computes all intersection points of NLCs, and obtain the influence value of each intersection point (i.e., number of customers whose NLCs encloses the intersection point). Then, these intersection points are used to represent the corresponding intersection regions. The algorithm sorts the intersection points in descending order of their influence value, and enumerate all combinations of $k$ intersection points to find the optimal combination. We choose the point with largest influence value first to form a combination. During the process of checking all combinations, we can prune the combinations that are unpromising. The second algorithm adopts an approximate strategy to reduce the computation complexity. The algorithm also computes all the intersection points first, but then adopts a clustering method to find a small set of candidate representative points (one from each cluster). At last, it chooses the optimal combination of $k$ representative points to form the final result. Our contributions are summarized as follows:

1. We propose and formulate the C$k$OLS problem, and prove that the problem is NP-hard.
2. We propose both a precise and an approximate algorithm to solve the C$k$OLS problem. The precise algorithm uses a brute-force search with a pruning strategy. The approximate algorithm adopts a clustering-based approach, which restricts the combinational search space within representative regions of clusters, and has a bounded approximation ratio.
3. We conduct extensive experiments over both real and synthetic datasets to evaluate the proposed algorithms. Experimental results verify the effectiveness and efficiency of the proposed algorithms.

The rest of the paper is organized as follows. Section 2 discusses related work. Problem statement is presented in Section 3. Section 4 and Section 5 describe the proposed algorithms and analyze their complexity. Section 6 presents experimental results and Section 7 concludes the paper.

## 2 Related Work

The MaxBRNN problem was first introduced by Cabello et al. in [2][3]. They call it MaxCOV and propose an algorithm in two-dimensional Euclidean space.

Wong et al. propose the MaxOverlap algorithm to solve the MaxBRNN problem [18]. MaxOverlap uses region-to-point transformation to reduce the optimal region search problem into an optimal intersection point search problem. The intersection points are computed by the NLCs of customer points, and the optimal intersection points are those covered by the largest number of NLCs. In another paper [17], Wong et al. extend the MaxOverlap algorithm to $L_p$-norm in two- and three-dimensional space. An algorithm called MaxFirst is presented by Zhou et al. [19] to solve the MaxBRNN problem. MaxFirst partitions the data space into quadrants, computes the upper and lower bounds of each quadrant's BRNN, and recursively partitions the quadrants with the largest upper bound until the upper bound and lower bound of some quadrants come to the same value. Liu et al. propose an algorithm called MaxSegment [7] for the MaxBRNN problem where the $L_p$-norm is used. Lin et al. present an algorithm called OptRegion [8] to solve the MaxBRNN problem. The algorithm employs the sweep line technique and a pruning strategy based on upper bound estimation to improve the search performance.

The maximum coverage problem is a similar problem, which computes a group of locations $p \in P$ (where $P$ is a given candidate service site set) that maximizes the total weight of the clients attracted [14][16]. This problem differs from ours in that, it dose not consider the existing facilities, and hence the solutions are not suitable to our problem.

Another similar problem is the Top-$k$ MaxBRNN query [7][15]. The Top-$k$ MaxBRNN problem simply ranks the regions by the number of BRNNs attracted and returns the Top-$k$ regions. Although it also returns a group of $k$ regions such that setting up a new service site within each region attracts a large number of customers, the Top-$k$ regions may shared many customers attracted, which is different from our C$k$OLS problem. However, the algorithm proposed for Top-$k$ MaxBRNN can also be considered as an approximate method for C$k$OLS. We will compare the proposed approximate algorithm with Top-$k$ MaxBRNN query in the the experiment section. Qi et al. conducted a series of studies [9][10][11] on the Min-dist location selection problems. They find a service site location to minimize the average distance between the service sites and the customers they are serving. This is a different optimization goal and the studies will not be discussed further.

## 3 Preliminaries and Problem Statement

### 3.1 Problem Statement

Given a set of customer points $O$ and a set of service sites $P$, each customer point $o$ is associated with a weight $w(o)$, which indicates the number of customers at $o$. For a point $p \in P$, $BRNN(p, O, P)$ represents the set of customer points that take $p$ as their nearest neighbor in $P$. For a point $s \notin P$, $BRNN(s, O, P \cup \{s\})$ represents the set of points that take $s$ as their nearest neighbor if $s$ is added into $P$. In order to describe the attracted customer points of a new service site, we define *influence set*, *influence list* and *influence value* in Definitions 1 and 2.

**Definition 1.** *[Influence set/value of a single service site] Given a service site $s$, we define the influence set of $s$ to be $BRNN(s, O, P \cup \{s\})$. The influence value of $s$ is equal to $\sum_{o \in BRNN(s,O,P\cup\{s\})} w(o)$.*

**Definition 2.** *[Influence list/value of k points] Let $\mathcal{S}$ be a set of k new service sites. The influence list of $\mathcal{S}$ is the union of BRNN(s, O, P) for every service site $s \in \mathcal{S}$. We denote this list by $U$, $U = \cup_{s \in \mathcal{S}} BRNN(s, O, P \cup \mathcal{S})$. The influence value of $\mathcal{S}$ is the sum of the weights of the customers in $U$, i.e., $\sum_{o \in U} w(o)$.*

**C*k*OLS:** Given a set of customer points $O$ and a set of service sites $P$ in a two-dimensional Euclidean space, the C*k*OLS problem finds $k$ optimal regions, such that setting up $k$ new service sites, one in each region, will collectively attract the maximum number of customers.

### 3.2 Region-to-point Transformation

Wong et al. transform finding optimal regions to finding optimal points for the MaxBRNN problem [18]. We also use this transformation for the C*k*OLS problem. We use the intersection points of NLCs (intersection points, for short) to represent regions. We say that a region is *determined by* the intersection points of the NLC arcs enclosing the region.

Notice that, the influence set of $q_1$ is $\{o_1, o_4\}$ (as shown in Fig. 2), which is different from that of $q_2$ and $q_3$, while $q_2, q_3, q_4$ share the same influence set $\{o_1, o_2, o_4\}$, and region $s_1$ is not an optimal region of C*k*OLS, since $s_2$ attracts more customers. We can see that, if a region is determined by intersection points with different influence sets, it cannot be the optimal region. This is because, a region's influence value is always bounded by the intersection point with the lowest influence value among all the intersection points that determine this region. Next, we introduce Theorem 1 to help us find the optimal region efficiently.

**Theorem 1.** *Any optimal region for CkOLS must be determined by the intersection points with the same influence set.*

*Proof.* For any region $s$ that is determined by intersection points $p_1$, $p_2$, ..., $p_n$ with different influence list, we assume that $p$ is the one with largest influence value in $p_1$, $p_2$, ..., $p_n$. Thus there must exist some other intersection points (not in $p_1$, $p_2$, ..., $p_n$) with the same influence list as $p$. The region $s_0$, which is determined by $p$ and the intersection points with the same influence list as $p$, must be a better region than $s$ for the C*k*OLS problem ($s_0$ attracts more customer points than $s$). This means that the region determined by intersection points with different influence list cannot be the optimal region of C*k*OLS. Therefore, it can be deduced that the optimal region of C*k*OLS problem must be determined by the intersection points with the same influence list.

According to Theorem 1, we can prune a large number of regions which cannot be the optimal regions of C*k*OLS. We only need to deal with the regions determined by intersection points with the same influence set. Since any point inside $s$ has the same influence set, we use just one intersection point $p_i$ in $p_1, p_2, ..., p_n$ to represent $s$. Adopting this region-to-point transformation, the C*k*OLS problem becomes finding $k$ intersection points, with the objective that the influence value of the $k$ intersection points is maximum.

### 3.3 NP-hardness of C$k$OLS

Next, we show that the C$k$OLS problem is NP-hard by reducing an existing NP-complete problem called the *Maximum Coverage Problem* to C$k$OLS in polynomial time.

**Maximum Coverage Problem:** Given a number $t$ and a collection of sets $\mathcal{S} = \{S_1, S_2, S_3, ..., S_m\}$, the objective is to find a subset $\mathcal{S}' \subset \mathcal{S}$ such that $|\mathcal{S}'| \leq t$ and the number of covered elements $|\cup_{S_i \in \mathcal{S}'} S_i|$ is maximum. In the weighted version, every element has a weight, and the objective is to find a collection of coverage sets which has the maximum sum of weight.

We reduce the Maximum Coverage Problem to C$k$OLS as follows. Sets $S_1, S_2, S_3, ..., S_m$ correspond to the influence sets of the intersection points. The collection of sets $\mathcal{S}$ is equal to the collection of the influence sets of all intersection points, and $t$ is equal to $k$. It is easy to see that this transformation can be constructed in polynomial time, and it can be easily verified that when the problem is solved in the transformed C$k$OLS problem, the original Maximum Coverage Problem is also solved. Since the Maximum Coverage Problem is an NP-complete problem, C$k$OLS is NP-hard.

**Theorem 2.** *The C$k$OLS problem is NP-hard.*

## 4 Precise Algorithm

In this section, we propose a precise algorithm which is a brute-force enumeration based algorithm. We call the algorithm C$k$OLS-Enumerate. Using the intersection points to represent regions, we compute all the intersection points and check the total combinations of $k$ intersection points to obtain the optimal result of C$k$OLS. We divide the precise algorithm into two phases.

**Phase 1** We compute all intersection points of NLCs, and obtain the influence set of each intersection point. In order to obtain the NLC of each customer point, we use kd-tree [19] to perform nearest neighbor query. We build a kd-tree of service sites and use the algorithm ANN [1] to find the nearest service sites over the kd-tree. The influence set of each intersection point is the set of customer points whose NLCs cover the intersection point.

**Phase 2** We sort the intersection points in descending order of their influence values, and the point with the largest influence value is processed first. We enumerate to check all combinations of $k$ intersection points to get the optimal result of C$k$OLS. The algorithm is summarized in Algorithm 1.

### 4.1 Enumeration Algorithm

We adopt a recursive algorithm in Algorithm 1. The main idea is to choose one point from all intersection points and choose $m - 1$ ($m$ is the number of points need to be added into one combination, and initially $m = k$) points from the remaining intersection points. During the process of enumeration, we add the intersection point one by one into a combination of $k$ points. Lines

7-9 check whether we have added $k$ points into one combination. Lines 12-14 prune the combinations which are unnecessary to check. We will elaborate this pruning strategy in the next subsection. Keeping the intersection points in a sorted list, lines 15-17 choose the next point to add into one combination, and iteratively choose $m-1$ points from the remaining $n-i-1$ ($n$ is the number of intersection points) intersection points. After having added $k$ points into one combination, lines 18-23 verify whether the influence value of this combination is larger than the maximum influence value ever found, and update the maximum influence value if necessary. Finally, we obtain the optimal result after checking all combinations of $k$ intersection points.

**Algorithm 1** C$k$OLS-Enumerate algorithm

```
Input: $k$: number of optimal regions, $IP$: set of sorted intersection points
Output: $k$ optimal points
 1: $n$: number of intersection points; $i = 0$
 2: $m := k$ /*number of points to be added into one combination */
 3: $maxValue := 0$ /* the largest combination influence value already found */
 4: $optCombination := \emptyset$ /* the optimal combination */
 5: $cValue := 0$ /* combination influence value of added points in one combination */
 6: procedure COMBINATION($n$, $IP$, $m$)
 7:     if $m == 0$ then
 8:         return $optCombination$
 9:     end if
10:     while $i < n - m$ do
11:         $value :=$ influence value of the $i$th point
12:         if $m \cdot value + cValue < maxValue$ then
13:             break
14:         end if
15:         Add $i$ into this combination
16:         $m := m - 1$, update $cValue$
17:         Combination($n - i - 1$,$IP$,$m$)
18:         if $m == 0$ then
19:             if $cValue > maxValue$ then
20:                 $maxValue := cValue$
21:                 update $optCombination$
22:             end if
23:         end if
24:         $i := i + 1$
25:     end while
26: end procedure
```

## 4.2 Pruning Strategy

In this subsection, we introduce our pruning strategy exploited by the enumeration algorithm (Lines 12-14 in Algorithm 1). Using the sorted intersection point list $IP$, we choose the point with larger influence value earlier to add into one combination. During the process of checking any combination $\mathcal{C}$, we assume that there are $m$ points to be added into $\mathcal{C}$. The influence value of the $k - m$ added points of $\mathcal{C}$ is $cValue$. The point with the largest influence value left in $IP$ is $maxP$ (which is the very first point left in $IP$), and the influence value of $maxP$ is $maxPValue$. The influence value of set $\mathcal{C}$ cannot be larger than the sum of the influence value of each intersection point. Therefore, after adding the remaining $m$ points into $\mathcal{C}$, the influence value of $\mathcal{C}$ cannot be larger than $m \cdot maxPValue + cValue$. Let $maxValue$ be the maximal influence value of combinations already found. As long as $m \cdot maxPValue + cValue < maxValue$, combination $\mathcal{C}$ can be pruned.

### 4.3 Time Complexity of C*k*OLS-Enumerate

First, we discuss the time complexity to compute all intersection points. In order to compute the NLC of each customer point, we use the All Nearest Neighbor algorithm [1], which requires $O(\log |P|)$ time. The time complexity of the construction of a kd-tree for $P$ is $O(|P| \log |P|)$. Then, it takes $O(|P| \log |P| + |O| \log |P|)$ time to get NLCs of all customer points in $O$. After obtaining NLCs, we use MBR of NLCs to compute the intersection points of NLCs. Suppose that, for each NLC, its MBR intersects with at most $d$ MBRs. Then it takes $O(d)$ time to process each NLC. Therefore, the total time complexity to compute the intersection points is $O(|P| \log |P| + |O| (\log |P| + d))$. In addition, we need to sort the intersection points $IP$, which requires $O(|IP| \log |IP|)$ time.

Second, as we can see that the enumeration algorithm is a brute-force algorithm. In the worst case, let $N$ be the number of intersection points, the enumeration algorithm will check $\binom{k}{N}$ combinations to get the optimal result. Let $m$ be the maximum size of influence set of points in the intersection points list. The time complexity of checking each combination is $O(k^2 \cdot m)$. Thus, the worst-case time complexity of the enumeration algorithm is $O(k^2 \cdot m \cdot \binom{k}{N})$. With the increase of the data volumes, the enumeration algorithm degrades rapidly.

## 5 Approximate Algorithm

In the enumeration algorithm, with the growth of the cardinality of intersection points, the combinatorial number of $k$ intersection points would be very large. As a result, it would take a long running time. In order to reduce the cost of enumeration, we aim to reduce the number of intersection points to be checked. Towards this aim, we propose a clustering based approach.

As we can see, there are many intersection points whose influence sets are similar to each other. For two intersection points with largely overlapping influence sets, the influence value of the combination of the two intersection points may not be much larger than the influence value of either point. Putting the two points in a combination is less desired. Hence, we partition the intersection points into clusters according to the similarity of their influence sets. We choose a *representative point* of each cluster, and use the representative point to represent all the other points inside the cluster. Afterwards, we only need to check the combinations of the representative points to obtain the approximate optimal result. Since the number of representative points is much smaller than that of the entire set of intersection points, the enumeration cost can be greatly reduced. We call the clustering-based approximate algorithm C$k$OLS-Approximate. The approximate algorithm also runs in two phases. The first phase clusters the intersection points and compute the representative points. The second phase checks the combination of the representative points in the same way as the precise algorithm. We omit the full pseudo-code of C$k$OLS-Approximate, and only describe *Clustering algorithm* here.

## 5.1 Clustering Algorithm

In order to cluster the intersection points according to the similarity between their influence sets, we propose the concept of *Discrepancy*. For ease of discussion, we first define a new operation in Definition 3.

**Definition 3.** *[Operation $\langle\rangle$] Given an influence set $N$, $\langle N\rangle = \sum_{o\in N} w(o)$. Point $o$ is a customer point in $N$, and $w(o)$ is the weight of $o$.*

**Definition 4.** *[Discrepancy] Given two service sites $s_1$ and $s_2$, let $N_1$ and $N_2$ denote the influence sets of $s_1$ and $s_2$. We define the discrepancy of $s_2$ to $s_1$ as $d(s_1, s_2)$, $d(s_1, s_2) = \frac{\langle N_2 - N_1\rangle}{\langle N_1\rangle}$. Notice that $d(s_2, s_1)$, which denotes the discrepancy of $s_1$ to $s_2$, is not equal to $d(s_1, s_2)$.*

We cluster the intersection points as follows. First, we sort the intersection point list $L$ in decreasing order of influence value. Second, we pick the point $p$ with the largest influence value in $L$ as the representative point of a new cluster $cl$, and remove $p$ from $L$. Third, after building a new cluster, we need to add the points with discrepancy to $p$ less than $\alpha$ into $cl$, and remove these points from $L$ as well. We say that the points in one cluster are represented by the representative point of this cluster. We repeat the second and third steps until $L$ becomes empty. When finishing clustering, the influence value of any representative point is larger than that of any other point in the same cluster.

As described in Algorithm 2, lines 3-5 get the point $p$ with the largest influence value in $L$ to build a new cluster $cl$. Point $p$ is the representative point of $cl$. Since $L$ is sorted by decreasing order of influence value, the top point of $L$ is just the one with the largest influence value. We say that, an intersection point $cp$ is represented by $p$ if $d(p, cp) < \alpha$. Lines 6-12 get the candidate points that could probably be represented by $p$, check all these points, and add these points into $cl$. The procedure stops when every intersection point has been partitioned into a cluster.

Table 1 shows a part of intersection points and clustering result of Fig. 1. Points $q_1, ..., q_{10}$ are intersection points of NLCs of $o_1, ..., o_4$. The clustering algorithm partitions $q_1, ..., q_{10}$ into clusters $cl_1$ and $cl_2$, and $cl_1 = \{q_1, q_2, q_3, q_7, q_8\}$, $cl_2 = \{q_4, q_5, q_6, q_9, q_{10}\}$. Point $q_1$ ($q_4$) is the representative point of $cl_1$ ($cl_2$). The

**Algorithm 2** Clustering algorithm

**Input:** $L$: sorted intersection point list
**Output:** $\mathcal{C}$: set of clusters of intersection points

```
 1: procedure Clustering($L$)
 2:     while $L$ is not empty do
 3:         $p :=$ top point of $L$
 4:         remove $p$ from $L$
 5:         build a new cluster $cl$
 6:         $cl.clusterCenter := p$
 7:         $Cand :=$ GetCandidate($p$)
 8:         for $cp \in Cand$ do
 9:             if $d(p, cp) < \alpha$ then
10:                 add $cp$ into $cl$
11:                 remove $cp$ from $L$
12:             end if
13:         end for
14:     end while
15: end procedure
16:
17: procedure GetCandidate($p$)
18:     $set :=$ influence set of $p$
19:     $maxRadius :=$ get maximum NLC radius of point in $set$
20:     $Cand :=$ the points with distance to $p$ smaller than $2 \cdot maxRadius$
21:     return $Cand$
22: end procedure
```

discrepancies of the other points in $cl_1$ ($cl_2$) to $q_1$ ($q_4$) are all 0. The discrepancy of $q_4$ to $q_1$ is 0.33. As a result, $q_4$ cannot be partitioned into the cluster of $q_1$ if $\alpha$ equals 0.2. We list some examples of discrepancy in Table 2.

### 5.2 Getting Candidate Represented Points

In the process of getting represented points of any representative point, it will take much time if we check all points in the intersection point list. Considering the principle of locality, given an intersection point $p$, the intersection points that are closer to $p$ are more likely to be represented by $p$. Next, we will describe how to get the candidate represented points in Lines 17-22 of Algorithm 2 by the explanation of Theorem 3.

**Theorem 3.** *Given two intersection points p and q, let maxRadius be the maximum NLC radius of points in p's influence set; let $N_1$ and $N_2$ be the influence set of p and q. If $dist(p, q)$ (distance from p to q) is larger than $2 \cdot maxRadius$, and $\langle N_2 \rangle > \langle N_1 \rangle \cdot \alpha$, then the discrepancy of q to p must be larger than $\alpha$.*

*Proof.* Since $dist(p, q) > 2 \cdot maxRadius$, $p$ and $q$ cannot be covered by any NLC simultaneously. If there is an NLC $c$ that covers $p$ and $q$ at the same time, let $r$ be the radius of $c$, then $dist(p, q)$ must be smaller than $2r$. As we have

Table 1: An example of intersection points

| intersection point | influence set | cluster |
|---|---|---|
| $q_1$ | $\{o_1, o_2, o_4\}$ | $cl_1$ |
| $q_2$ | $\{o_1, o_2, o_4\}$ | $cl_1$ |
| $q_3$ | $\{o_1, o_2, o_4\}$ | $cl_1$ |
| $q_4$ | $\{o_2, o_3, o_4\}$ | $cl_2$ |
| $q_5$ | $\{o_2, o_3, o_4\}$ | $cl_2$ |
| $q_6$ | $\{o_2, o_3, o_4\}$ | $cl_2$ |
| $q_7$ | $\{o_1, o_2\}$ | $cl_1$ |
| $q_8$ | $\{o_1, o_4\}$ | $cl_1$ |
| $q_9$ | $\{o_2, o_3\}$ | $cl_2$ |
| $q_{10}$ | $\{o_3, o_4\}$ | $cl_2$ |

Table 2: Discrepancy value

| discrepancy | value |
|---|---|
| $(q_1, q_2)$ | 0 |
| $(q_1, q_3)$ | 0 |
| $(q_1, q_4)$ | 0.33 |
| $(q_1, q_7)$ | 0 |
| $(q_1, q_9)$ | 0.33 |

discussed in Section 3, the influence set of $p$ equals the set of points whose NLC covers $s$. Thus $r$ must be smaller than $maxRadius$, and we get the contradiction that $dist(p, q) < 2 \cdot maxRadius$. Therefore, there is no intersection of $p$ and $q$'s influence sets. This means that $N_2 - N_1 = N_2$. Then we can get that $d(p, q) = \frac{\langle N_2 \rangle}{\langle N_1 \rangle}$. Under the condition that $\langle N_2 \rangle > \langle N_1 \rangle \cdot \alpha$, we can prove that $d(p, q) > \alpha$.

Let $c$ be the circle centered at $p$ with radius equals $2 \cdot maxRadius$, then the candidate represented points of $p$ are all the intersection points inside $c$. According to Theorem 3, we can get limited candidate represented points of $p$ instead of checking every point in $L$, and this helps saving lots of time while clustering the intersection points. In this paper, we traverse the influence set of the representative points to get $maxRadius$, and construct a kd-tree [19] of intersection points to help us get the candidate represented points. Given an representative point $p$, we do a range query on kd-tree to obtain the other intersection points that are within the distance of $2 \cdot maxRadius$ to $p$.

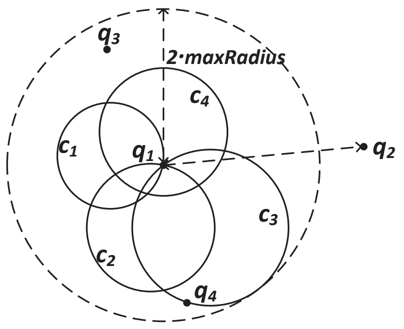


Fig. 3: An example of Theorem 3

As shown in Fig. 3, given four intersection points $q_1, ..., q_4$, $c_1, ..., c_4$ are the NLCs of points in $q_1$'s influence set. $MaxRadius$ is the maximum NLC radius of $q_1$'s influence set, which is the radius of $c_3$ in this example. Since $dist(q_1, q_2) > 2 \cdot maxRadius$, there cannot be any NLC that covers $q_1$ and $q_2$ simultaneously. According to Theorem 3, we can easily get the candidate represented points set of $q_1$, which is $\{q_3, q_4\}$. During the second phase of the approximate algorithm, we adopt the same enumeration process as the C$k$OLS-Enumerate algorithm. We simply check the overall combinations of representative points, and finally obtain an approximate optimal $k$ points.

## 5.3 Accuracy of C$k$OLS-Approximate

Next, we will prove the accuracy of algorithm C$k$OLS-Approximate.

**Theorem 4.** *The accuracy of CkOLS-Approximate is at least* $\frac{1+\alpha}{1+k\cdot\alpha}$.

*Proof.* We suppose that the optimal result of C$k$OLS is $\{a_1, a_2, a_3, ..., a_k\}$, each $a_i$ indicates the intersection point. The corresponding representative point of $a_i$ is $b_i$. For convenience, we use $A_i$ and $B_i$ to indicate the influence set of $a_i$ and $b_i$. Next, we will prove that $\frac{\langle B_1 \cup B_2 \cup B_3 \cup, ..., \cup B_k\rangle}{\langle A_1 \cup A_2 \cup A_3 \cup, ..., \cup A_k\rangle} > \frac{1+\alpha}{1+k\cdot\alpha}$.

Let $\Delta B_i = A_i - B_i$, we can easily see that $A_i \subset B_i \cup \Delta B_i$, then

$$\begin{aligned} & \frac{\langle B_1 \cup B_2 \cup B_3 \cup, ..., \cup B_k\rangle}{\langle A_1 \cup A_2 \cup A_3 \cup, ..., \cup A_k\rangle} \\ \geq & \frac{\langle B_1 \cup B_2 \cup B_3 \cup, ..., \cup B_k\rangle}{\langle (B_1 \cup \Delta B_1) \cup (B_2 \cup \Delta B_2) \cup (B_3 \cup \Delta B_3) \cup, ..., \cup (B_k \cup \Delta B_k)\rangle} \\ \geq & \frac{\langle B_1 \cup B_2 \cup B_3 \cup, ..., \cup B_k\rangle}{\langle B_1 \cup B_2 \cup B_3 \cup, ..., \cup B_k\rangle + \langle \Delta B_1 \cup \Delta B_2 \cup \Delta B_3 \cup, ..., \cup \Delta B_k\rangle} \\ = & \frac{1}{1 + \frac{\langle \Delta B_1 \cup \Delta B_2 \cup \Delta B_3 \cup, ..., \cup \Delta B_k\rangle}{\langle B_1 \cup B_2 \cup B_3 \cup, ..., \cup B_k\rangle}} \end{aligned} \tag{1}$$

With the precondition that the discrepancy of $a_i$ to $b_i$ is less than $1 - \alpha$, $\Delta B_i$ must be less than $\langle B_i\rangle \cdot \alpha$. Supposing that $b_j$ is the point with the largest influence value in $\{b_1, b_2, b_3, ..., b_k\}$, then

$$\begin{aligned} & \langle \Delta B_1 \cup \Delta B_2 \cup \Delta B_3 \cup, ..., \cup \Delta B_k\rangle \\ \leq & \langle \Delta B_1\rangle + \langle \Delta B_2\rangle + \langle \Delta B_3\rangle + ... + \langle \Delta B_k\rangle \\ \leq & \langle B_1\rangle \cdot \alpha + \langle B_2\rangle \cdot \alpha + \langle B_3\rangle \cdot \alpha + ... + \langle B_k\rangle \cdot \alpha \\ \leq & \langle B_j\rangle \cdot \alpha \cdot k \end{aligned} \tag{2}$$

Knowing that $b_j$ has the largest influence value, there must exist a certain point $b_m$ in $\{b_1, b_2, b_3, ..., b_k\}$, that satisfies $d(b_m, b_j) > \alpha(b_i \neq b_m)$. If not, it means that all of $\{b_1, b_2, b_3, ..., b_k\}$ can be partitioned into the same cluster as

$b_j$, which means that $\{b_1, b_2, b_3, ..., b_k\}$ cannot be the result of the approximate algorithm. As a result, $\langle B_j \cup B_m \rangle = \langle B_j \rangle + \langle B_m - B_j \rangle > \langle B_j \rangle \cdot (1+\alpha)$. Therefore, $\langle B_1 \cup B_2 \cup B_3 \cup, ..., \cup B_k \rangle$ must be larger than $\langle B_j \rangle \cdot (1+\alpha)$, then

$$\frac{\langle \Delta B_1 \cup \Delta B_2 \cup \Delta B_3 \cup, ..., \cup \Delta B_k \rangle}{\langle B_1 \cup B_2 \cup B_3 \cup, ..., \cup B_k \rangle} \leq \frac{\langle B_j \rangle \cdot \alpha \cdot k}{\langle B_j \rangle \cdot (1+\alpha)} = \frac{k\alpha}{1+\alpha} \tag{3}$$

$$\frac{\langle B_1 \cup B_2 \cup B_3 \cup, ..., \cup B_k \rangle}{\langle A_1 \cup A_2 \cup A_3 \cup, ..., \cup A_k \rangle} \geq \frac{1}{1+\frac{k\alpha}{1+\alpha}} \geq \frac{1+\alpha}{1+k\cdot\alpha} \tag{4}$$

Since the approximate algorithm makes combination of the representative points, the output result of the approximate algorithm must be larger than (or at least equal to) $\langle B_1 \cup B_2 \cup B_3 \cup, ..., \cup B_k \rangle$. Through the derivation above, we can conclude that the accuracy of the approximate algorithm is at least $\frac{1+\alpha}{1+k\cdot\alpha}$.

### 5.4 Time Complexity of C*k*OLS-Approximate

We partition the time complexity into three parts. First, we calculate all intersection points and the time complexity is the same as we have discussed in Section 4.3. We omit it here. Second, we analyze the time complexity of the clustering algorithm. We use heap sort to sort the intersection points $IP$, which requires $O(|IP| \log |IP|)$ time. Then we build a kd-tree of $IP$, which requires $O(|IP| \log |IP|)$ time. Let $m$ be the maximum size of influence sets of intersection points in $IP$, $n$ be the maximum size of candidate represented points of the representative points. Assuming that there are $t$ clusters after partition of the intersection points, it would take $O(t(\log |P| + m))$ time to get the candidate represented points. For the convenience of discrepancy computation, we sort the influence set of each point in $IP$, which requires $O(|IP| \cdot m \log m)$ time. The time complexity of discrepancy computation in one cluster is $O(m \cdot n)$. Totally, the time complexity of clustering algorithm is $O(|IP|\,(m \log m + \log |IP|) + t(\log |IP| + m \cdot n))$. Third, in the worst case, we have to check $\binom{k}{0.1\cdot|IP|}$ combinations to get the approximate optimal result, which is far fewer than the precise algorithm. The worst query time complexity is $O(k^2 \cdot m \cdot \binom{k}{0.1\cdot|IP|})$.

The total time complexity of the approximate algorithm is the sum of the above three parts.

## 6 Performance Study

We have conducted extensive experiments to evaluate the proposed algorithms using both synthetic and real datasets. The algorithms are implemented in C++. All experiments are carried out on a Linux Machine with an Intel(R) Core i5-4590 3.30 GHz CPU and 8.00 GB memory. The synthetic datasets follow Gaussian and Uniform distribution, and the data volume ranges from 1K to 100K. The customer dataset and service dataset follow the same distribution. As for real datasets, we use LB and CA, which contain 2D points representing geometric locations in Long Beach Country and California respectively[3]. To

[3] http://www.rtreeportal.org/spatial.html

keep consistency, we partition the real datasets into two parts to ensure that the customer points and service sites share the same distribution. Considering that the number of service sites is usually much fewer than that of customer points, we set the cardinality of $P$ to be half of the cardinality of $O$ for all datasets.

Section 6.1 tests effect of clustering parameter $\alpha$. Section 6.2 compares the running time and accuracy of the proposed algorithms. Section 6.3 tests the algorithms' scalability. Section 6.4 compares the effectiveness of our C$k$OLS query with Top-k MaxBRNN query.

## 6.1 Effect of Clustering Parameter

The clustering algorithm outputs the representative points of the intersection points. Instead of making combination of the total intersection points, we only use the representative points to get the optimal combination. As a result, the ratio of the number of representative points to the number of entire intersection points is essential for the efficiency of the approximate algorithm. For simplicity, we call the ratio *clustering efficiency.* A small *clustering efficiency* means a small number of representative points.

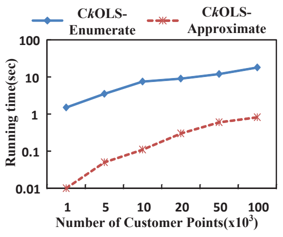
Table 3: Effect of $\alpha$

| $\alpha$ | *clustering efficiency* | accuracy | running time |
|---|---|---|---|
| 0.05 | 0.22 | 0.98 | 3$s$ |
| 0.1 | 0.22 | 0.95 | 2.9$s$ |
| 0.2 | 0.1 | 0.9 | 1.2$s$ |
| 0.3 | 0.04 | 0.83 | 0.8$s$ |
| 0.4 | 0.02 | 0.8 | 0.6$s$ |

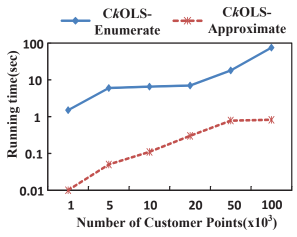
The experiments are conducted on synthetic datasets with 1K to 100K customer points, and we get the average clustering efficiency when $\alpha$ varies from 0.05 to 0.4. As we can see from Table 3, the *clustering efficiency* decreases with the increase of $\alpha$. Besides impacting the clustering efficiency, $\alpha$ also decides the effectiveness of the approximate algorithm. As shown in Table 3, the larger $\alpha$ is, the worse approximate result we would get with shorter running time. For the balance of effectiveness and clustering efficiency, we conduct experiments with $\alpha = 0.2$ in the following experiments.

## 6.2 Comparison of Algorithms

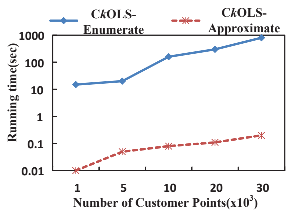
The results are given in Fig. 4 and Fig. 5 over datasets with different cardinality and distribution. We compare the running time of the precise algorithm with approximate algorithm with $k = 10$ and $\alpha = 0.2$, and set the weight of all customer points to 1. We only consider the running time without the time of calculating the intersection points of NLCs.



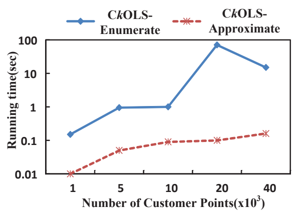
(a) Gaussian



(b) Uniform



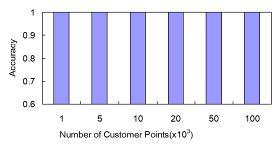
(c) lb



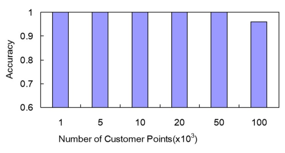
(d) ca

Fig. 4: Running time of proposed algorithms

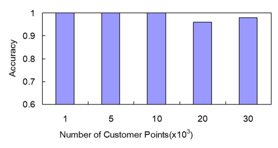
From Fig. 4 we can see, as the cardinality of customer points increases, the query time of C$k$OLS-Approximate is at least one order of magnitude faster than C$k$OLS-Enumerate. The efficiency of C$k$OLS-Enumerate is unstable on different datasets, and the query time of dataset with larger cardinality is possibly less than the same dataset with smaller cardinality. For example, in the experiments on CA dataset, the query time of C$k$OLS-Enumerate over 40K customer points is less than that over 20K customer points. The reason behind is that the efficiency of enumeration algorithm is decided by how fast the optimal combination is found during the enumeration of combination. For the convenience of comparison, we set $k$ to 5 in the experiments of CA dataset for the reason that it takes too long for the enumeration algorithm to get the optimal result with $k = 10$. Fig. 5 shows the accuracy of C$k$OLS-Approximate algorithm. For the synthetic datasets, the accuracies of different datasets are almost 1. While for the real datasets, the average accuracy is about 0.9 to 0.95.



(a) Gaussian

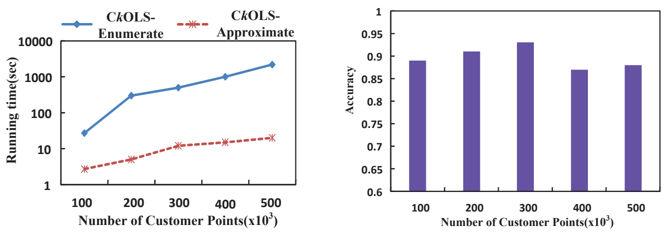


(b) Uniform



(c) lb

Fig. 5: Accuracy of C$k$OLS-Approximate algorithm

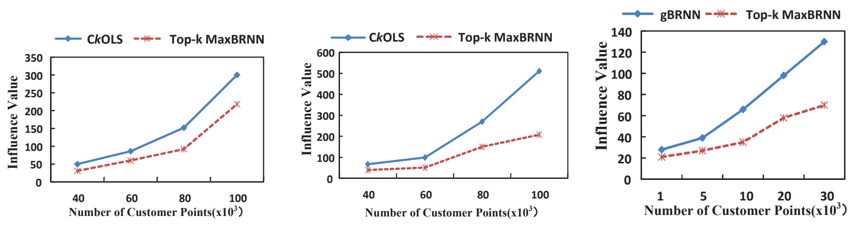


(a) Running Time (b) Accuracy

Fig. 6: Scalability of proposed algorithms

## 6.3 Scalability

Figure 6 shows the scalability of proposed algorithms. We set $\alpha = 0.2$ and $k = 10$ as their default values. We vary the number of customer points from 100K to 500K, and use Uniform dataset here, other distribution can get similar results. From Fig. 6(a), we can see that, as the increase of the number of customer points, the running time of approximate algorithm keeps almost stable, while the precise algorithm degrades a lot, especially when the dataset is 500K, the precise algorithm is very slow. As shown in Fig. 6(b), compare to the accuracy result on small scale datasets, the accuracy of approximate algorithm decreases, however, the worst case is still beyond 0.8.

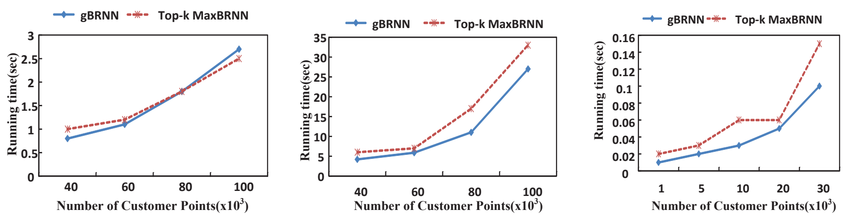


(a) Uniform (b) Gaussian (c) lb

Fig. 7: Comparison to Top-k MaxBRNN query (accuracy)

## 6.4 Comparison to Top-k MaxBRNN

We know that the Top-k MaxBRNN query can also return a group of $k$ regions to attract large number of customers. And it can be regarded as an approximate method. We compare the proposed algorithm C$k$OLS-Approximate with Top-k MaxBRNN query to find out which can collectively attract the maximum number of customers by proximity. Fig. 7 shows that, C$k$OLS query can



(a) Uniform (b) Gaussian (c) lb

Fig. 8: Comparison to Top-k MaxBRNN query (running time)

always get the larger number of attracted customer points. And it is almost three times the number of Top-k MaxBRNN query when using Gaussian data set with $|O| = 100K$. From Fig. 7 (c), we can see that the result on real data set is similar to the synthetic data set. We also compare the running time of the two methods. As shown in Fig. 8, the two methods almost have the same running time no matter on which data distribution. However, as we have discussed, the Top-k MaxBRNN query has much worse accuracy.

## 7 Conclusion

We studied the C$k$OLS problem and proposed a precise and an approximate algorithm to solve the problem. The precise algorithm uses a brute-force search with a pruning strategy. The approximate algorithm adopts a clustering-based approach, which restricts the combinational search space within representative regions of clusters, and has a bounded approximation ratio. Experiment results show that the approximate algorithm is effective and efficient. Its output is very close to that of the precise algorithm, with an approximation ratio of up to 0.99 on all the datasets. The algorithm is also efficient. It outperforms the precise algorithm by up to three orders of magnitudes in terms of the running time.

For future work, we are interested in adapting the proposed algorithms to solve collective-$k$ location selection problems with other optimization goals such as min-dist [9][10][11].

**Acknowledgements** This work was partly supported by The University of Melbourne Early Career Researcher Grant (project number 603049), National Science and Technology Supporting plan (2015BAH45F00), the public key plan of Zhejiang Province (2014C23005), the cultural relic protection science and technology project of Zhejiang Province.

## References

1. Arya, S., Mount, D.-M., Netanyahu, N.-S., Silverman, R., Wu, A.-Y.: An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. JACM 45(6), 891-923 (1998)

2. Cabello, S., Daz-Bez, J.M., Langerman, S., Seara, C., Ventura, I.: Reverse facility location problems. In: CCCG, pp. 1-17 (2005)
3. Cabello, S., Daz-Bez, J.M., Langerman, S., Seara, C., Ventura, I.: Facility location problems in the plane based on reverse nearest neighbor queries. Eur. J. Oper. Res. 202(1), 99-106 (2010)
4. Chen,Z., Liu, Y., Wong, C., Xiong, J., Mai, G., Long, C.: Efficient algorithms for optimal location queries in road networks. In: SIGMOD, pp. 123-134 (2014)
5. Gao, Y., Zheng, B.H., Chen, G., Li, Q., Guo, X.F.: Continuous visible nearest neighbor query processing in spatial databases. PVLDB 20(3), 371-396 (2011)
6. Korn, F., Ukrishnan, S.-M.: Influence sets based on reverse nearest neighbor queries. In: SIGMOD, pp. 201-212 (2000)
7. Liu, Y.-B., Wong, R.-C., Wang, K., Li, Z.-J., Chen, C.: A New Approach for Maximizing Bichromatic Reverse Nearest Neighbor Search. In: KAIS, pp. 23-58 (2013)
8. Lin H, Chen F, Gao Y, Lu D.: Finding optimal region for bichromatic reverse nearest neighbor in two- and three-dimensional spaces. Geoinformatica 20(3), 351-384 (2016)
9. Qi, J., Zhang R. Wang Y., Xue Y., Yu G., and Kulik L.: The Min-dist Location Selection and Facility Replacement Queries. WWWJ 17(6), 1261-1293 (2014)
10. Qi, J., Zhang R., Kulik L., Lin D. and Xue Y.: The Min-dist Location Selection Query. In: ICDE, pp. 366-377 (2012)
11. Qi, J., Zhang R., Xue Y., and Wen Z.: A Branch and Bound Method for Min-dist Location Selection Queries. In: ADC, pp. 51-60 (2012)
12. Singh, A., Ferhatosmanoglu, H., Tosun, A.: High Dimensional Reverse Nearest Neighbor Queries. In: CIKM, pp. 91-98 (2003)
13. Stanoi, I., Agrawald, D.: Reverse nearest neighbor queries for dynamic databases. In: DMKD, pp. 44-53 (2000)
14. Sakai K., Sun M.-T., Ku W.-S., Lai T.H. and Vasilakos A.V.: A Framework for the Optimal-Coverage Deployment Patterns of Wireless Sensors. IEEE Sensors Journal 15(12), 7273-7283 (2015)
15. Sun Y., Huang J., Chen Y., Du X., Zhang R.: Top-k Most Incremental Location Selection with Capacity Constraint. In: WAIM, pp. 165-171 (2012)
16. Sun Y., Qi J., Zhang R., Chen Y., and Du X.: MapReduce based location selection algorithm for utility maximization with capacity constraints. Computing 97(4), 403-423 (2015)
17. Wong, R.-C., Tamer zsu, M., Fu, A.-W., Yu, P.-S., Liu, L., Liu, Y.: Maximizing bichromatic reverse nearest neighbor for Lp-norm in two- and three-dimensional space. In: VLDB, pp. 893-919 (2011)
18. Wong, R.-C., zsu, M.T., Yu, P.-S., Fu, A.-W., Liu, L.: Efficient method for maximizing bichromatic reverse nearest neighbor. In: VLDB, pp. 1126-1137 (2009)
19. Zhou, Z., Wu, W., Li, X., Lee, M.-L.: Wynne Hsu: MaxFirst for MaxBRkNN. In: ICDE, pp. 828-839 (2011)
20. Friedman J-H, Bentley J-L, Finkel R-A: An algorithm for finding best matches in logarithmic expected time. ACM TOMS 3:209-226 (1977)