Noname manuscript No.
(will be inserted by the editor)

GOAL: A Clustering Based Method for the Group Optimal Location Problem

Fangshu Chen · Jianzhong Qi · Huaizhong Lin✉ · Yunjun Gao · Dongming Lu

Received: 21 Mar 2017
Revised: 27 Jul 2018
Accepted: 15 Sep 2018

Abstract Optimal location problems are important problems and are particularly useful for strategic planning of resources. However, existing studies mainly focus on computing one or k optimal locations. We study the *Group OptimAl Location* (GOAL) problem, which computes a minimum set of locations such that establishing facilities at these locations guarantees that every facility user can access at least one facility within a given distance $r \in \mathcal{R}^+$. We propose two algorithms, GOAL-Greedy and GOAL-DP, to first solve the problem in the Euclidean space. These two algorithms are supported by a clustering based method to compute an initial solution of the problem, which yields an upper bound of the number of locations needed to solve the problem. We propose a grid partitioning based strategy to refine the initial solution and obtain the final solution. We further extend our algorithms to road networks. We perform extensive experiments on the proposed algorithms. The results show that the proposed algorithms can solve the problem effectively and efficiently in both Euclidean spaces and road networks.

Keywords Location selection · Clustering · Grid partition · Road network

Fangshu Chen
The College of Computer Science and Information Engineering, Shanghai Polytechnic University, China, 201209.
E-mail: youyou_chen@foxmail.com
Jianzhong Qi
School of Computing and Information Systems, University of Melbourne, Australia, 3000.
E-mail: jianzhong.qi@unimelb.edu.au
Huaizhong Lin, Yunjun Gao, Dongming Lu
The College of Computer Science and Technology, Zhejiang University, China, 310000.
E-mail: {linhz, gaoyj, ldm}@zju.edu.cn
This work is partially done when Fangshu is visiting the University of Melbourne.

1 Introduction

Optimal location problems have attracted on-going research efforts in the database community since Zhang et al. [35] introduced the *Optimal Location* (OL) query. An OL query considers two spatial point sets: a set P representing facilities (or service sites) and a set O representing clients. The query objective is to identify a location (or region) p, such that a new facility built at p can optimize a certain cost metric (e.g., the average) defined on the distances between the facilities and the clients. OL queries have various applications such as resource allocation, facility location selection, etc.

Existing studies mainly focus on computing a fixed number of optimal locations (e.g., 1 or k). Few have studied the problem of computing a set of optimal locations to provide a full coverage to the clients. Xu et al. [33] study the group location selection queries to compute a set of locations to cover the clients. They consider clients with uncertain locations, and their goal is to cover the clients with a certain probability.

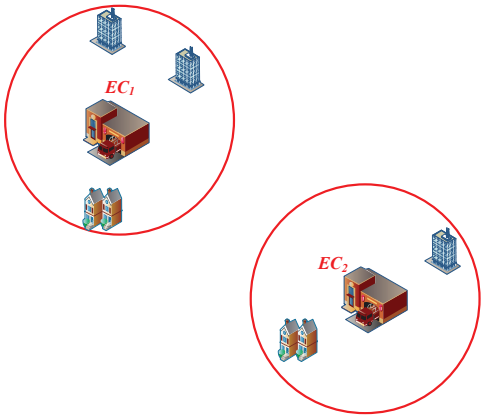
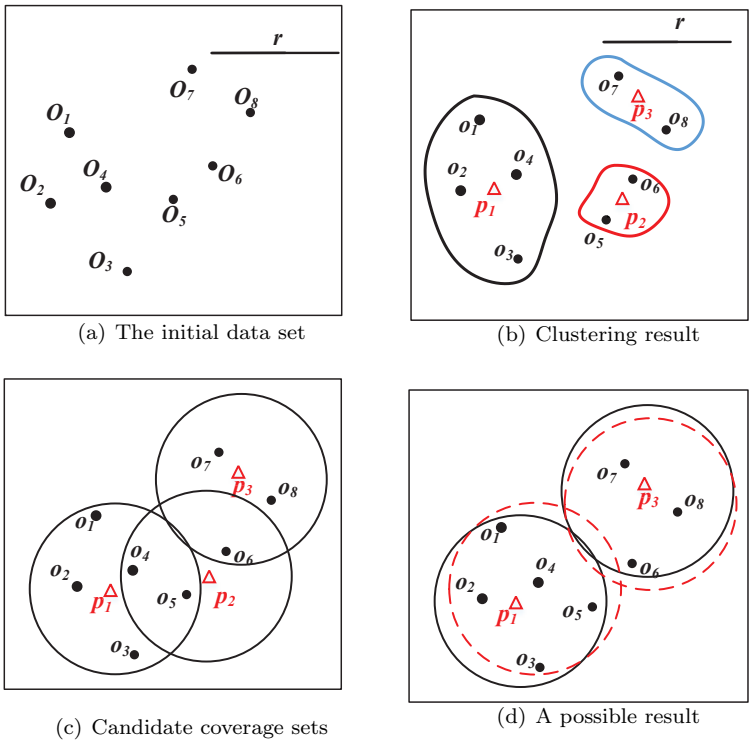


Fig. 1 Motivating Example: EC_1 and EC_2 represents two emergency centers,the other buildings represent residential buildings that need to be covered within a certain distance

We study a group location problem where client locations are certain: Given a constant $r \in \mathcal{R}^+$ representing the *coverage radius* of a facility and a client set O, we compute a minimum set of locations needed such that establishing facilities at these locations can *cover* all the clients in O. Here, we say that a facility covers a client if their distance does not exceed the coverage distance r. We call this problem the *Group OptimAl Location* (GOAL) problem.

The GOAL problem has various applications. For example, when setting up emergency centers in an area hit by a natural disaster, e.g., an earthquake, the emergency centers should be within a reachable distance to everyone. Our GOAL problem can help find out the minimum number of emergency centers needed and where to place them to cover the people in the area. This example is illustrated in Fig. 1, where two emergency centers EC_1 and EC_2 are needed



(a) The initial data set

(b) Clustering result

(c) Candidate coverage sets

(d) A possible result

Fig. 2 Example of the GOAL problem in Euclidean space

to cover all the residential areas nearby, and the circles represent the area covered. This can also be thought of as a scenario where a telecommunication service provider is to decide where to set up new base stations for the nearby users. Note that we assume an infinite facility capacity in this study. In reality, an emergency center or a base station can only serve a limited number of people. In such case, we can run our algorithms in the more dense regions with a smaller coverage radius in a divide-and-conquer manner, until each region has a sufficiently small population that can be served by a facility. The above problem can be formulated as a GOAL problem as shown in Fig. 2(a), where the clients are represented by a set $O = \{o_1, o_2, o_3, o_4, o_5, o_6, o_7, o_8\}$. A solution of the problem is shown in Fig. 2(d), where the clients are grouped into two coverage sets denoted by CS_1 and CS_2, each enclosed by a circle with the radius being the coverage radius r and the center being the geometric center of the set of clients, which are small triangles p_1, p_2. Establishing an emergency center (base station) at p_1, p_2 can provide a full coverage to all clients. As shown by the dashed circles, moving any circle slightly may still guarantee that all clients are covered. Thus, the problem solution may not be unique. However, it is clear that no single circle with radius r can cover all the clients, and 2 facilities are the minimum to provide a full coverage.

In this paper, we are interested in finding this minimum number, and one of the solutions with this minimum number. The overall procedure of our algorithms is: 1) First run a clustering algorithm to obtain an upper bound on the number of disjoint subsets needed to fully cover the data set where each cluster can be enclosed by an r-radius circle, as shown in Fig. 2(b), there are three clusters with centers p_1, p_2, p_3. Notice that, we must guarantee that the distance between a client point to its cluster center is smaller than r in this step. The clusters form our initial problem answer with their centers as coverage set centers and the radius is r. Figure 2(c) illustrate the initial coverage sets derived from clustering; 2) Second, we apply a two-stage merging procedure to further refine the coverage sets obtained. The first stage uses a grid to partition the data space and merges coverage sets in nearby grid cells. The second stage further merges the coverage sets through examining different pairs of coverage sets. Two strategies are used – greedy and dynamic programming – to reduce the number of pairs of coverage sets to be considered, resulting in two algorithms GOAL-Greedy and GOAL-DP, respectively, and the final result after merging is shown in Fig. 2(d), the answer set is $CS_1 = \{o_1, o_2, o_3, o_4, o_5\}$ and $CS_2 = \{o_6, o_7, o_8\}$.

The GOAL problem is NP-hard, as shown by Xu et al. [33]. Intuitively, the problem involves generating every subset of O that can be enclosed in a circle with radius r and selecting the combination of the minimum number of disjoint subsets that can cover O, which can be reduced to the set cover problem. In Fig. 2 (a), there are 30 candidate subsets: $\{o_1\}, \{o_2\}, ..., \{o_8\}$; $\{o_1, o_2\}, \{o_1, o_3\}, ..., \{o_7, o_8\}$; $\{o_1, o_2, o_3\}, \{o_1, o_2, o_4\}, \{o_1, o_3, o_4\}, ..., \{o_6, o_7, o_8\}$; $\{o_1, o_2, o_3, o_4\}, \{o_5, o_6, o_7, o_8\}$. Generating these subsets on n clients by enumeration takes $O(2^n)$ time, which is impractical.

Our contributions are summarized as follows.

- We are the first to study the group optimal location selection (GOAL) problem in both Euclidean spaces and road networks where the data object locations are certain.
- We estimate the upper bound of the number of locations needed to solve the problem which can reduce the searching space significantly.
- We propose a grid partitioning based method to refine the initial answer obtained by clustering. Two algorithms GOAL-Greedy and GOAL-DP are proposed to compute the final answer based on the grid partitioning in Euclidean space. The algorithms are further extended to solve the GOAL problem in road networks.
- We conduct extensive experiments with both real and synthetic data sets to demonstrate the efficiency and effectiveness of the proposed algorithms. The experimental results show that the proposed algorithms outperform the baseline algorithms significantly.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 formulates the GOAL problem. Section 4 describes the algorithms for GOAL in Euclidean space and road networks. Section 5 analyzes

the algorithm correctness and complexity. Section 6 presents the experimental results, and Section 7 concludes the paper.

2 Related Work

2.1 Optimal Location Queries

Zhang et al. [9] first introduced the optimal location (OL) query to the database community. Xiao et al. [32] classify optimal location queries into three categories based on the cost metrics: i) competitive location queries, which compute a candidate location $p \in P$ (where P is a given candidate facility location data set) that maximizes the total weight of the clients attracted by a new facility built at p [4] [5] [15] [19] [20] [29] [31] [36]; ii) $MinSum$ location queries, which compute a candidate location $p \in P$ on which a new facility can be set up to minimize the total *weighted attract distance* (WAD) of the clients [4]; iii) $MinMax$ location queries, which compute a candidate location $p \in P$ on which a new facility can be set up to minimize the maximum WAD of the clients [9] [18] [23] [24] [35]. Beyond these three categories, another category named the $MinCost$ query computes a location (a group of locations) for setting up a new facility (new facilities) such that the cost of setting up the facilities and the overall transportation cost is minimized [6] [7].

These studies differ from ours in that they are interested in computing just one location, while ours computes a (non-predetermined) number of locations, which is more challenging and cannot be solved by the existing approaches.

2.2 Facility Location and Set Cover Problems

Facility location problems (FLP) have been studied in the field of Operation Research [3] [22] [26] under application context such as sensor network deployment [2] [17] [25] [28], resource allocation [21] [25], etc. Common aims of FLP are: i) minimizing the cost of setting up new facilities [21] [22] or maximizing the number of clients served by a given number of facilities [2] [3] [25]; ii) selecting k of the candidate facility locations, and then assigning each of the clients to its closest facility so as to minimize their average distance [2] [17] [21] [26] [34]. FLP problems are similar to our problem but with two major differences. First, many FLP problems compute a fixed number of facility locations to maximize their impact rather than requiring a full coverage of the clients. Second, FLP often depends on specific applications and have application dependent cost metrics (e.g., sensor network cost model), and hence the solutions are difficult to extend to general distance based metrics.

Set cover [1] [8] [10] [14] is another branch of studies similar to ours. However, set cover algorithms are not directly applicable because they assume that the candidate subsets already exist while we aim to avoid generating all the candidate subsets.

2.3 Group Optimal Location Selection

More relevant studies are discussed in [12] [14] [25]. Sakai et al. [25] study the *k-optimal location query* (MkOLS) that selects the k locations that cover the maximum number of clients. They propose the *Estimation based Algorithm* (EB) which consists of a suite of pruning rules to reduce query cost. Although the algorithm is effective, its aim is to assign the maximum number of cilents to k facilities. Adapting the algorithm to our problem is inefficient because it requires running the algorithm iteratively with increasing value of k until all clients are covered. The other studies [12] [14] [33] share similar problem definition and techniques with [25] and hence their algorithms are not suitable to our problem either.

There are two algorithms that can be adapted to solve our problem – the GCA algorithm [33] and the CREST algorithm [30]. GCA answers the GOAL problem over a set of clients with uncertain locations in polynomial time while ensuring that all the client points can be covered with a bounded probability. GCA consists of two phases: i) selecting candidate coverage sets using a heuristic *Closure polygon* algorithm, ii) refining the coverage sets by dynamic programming. CREST is proposed to solve the reverse nearest neighbor (RNN) heat map problem which shows the number of RNNs of every point in the space with a grey scale heat map. It first draws nearest neighbor circles (NN-circles) to divide the space into separated regions and then adopts the classic *sweep line* algorithm to compute the number of RNNs of each region. By letting the NN-circle radius be our coverage radius r, each region produced by CREST corresponds to a candidate subset. We adapt the algorithm so that it produces the regions that together cover all the clients. We compare our algorithms with these two algorithms in the experimental study.

Notice that the competitive location queries, also called maximizing bichromatic reverse nearest neighbor search [20] [31] [36], can also be adapted to our problem. However, since the goal of the MaxOverlap algorithm introduced in paper "Efficient Method for Maximizing Bichromatic Reverse Nearest Neighbor (BRNN)" [20] [31] is finding out one optimal region or k regions to maximize the BRNN number, it cannot guarantee that all the client points can be served. The coverage sets derived by the algorithm may overlap with each other dramatically. That is to say, in the worst case, there may be some outlet point, and the MaxOverlap algorithm has to find out a huge number of optimal regions to cover it.

According to the analysis above, we modified the MaxOverlap algorithm [20] [31] to make it better adapt to our problem. The changes are as follows: i) For the NLC construction step in MaxOverlap [31] algorithm, we set the distance between a client point and its nearest service site be the coverage radius r, and in this way the nearest neighbor query in this step can be omitted; ii) we find out top k optimal regions instead of one optimal region such that the top k optimal regions can cover all the client points collectively; iii) we record the client point that has been covered, such that the top k MaxOverlap algorithm terminates only when all the points have been covered.

Table 1 Frequently Used Symbols

Symbol	Meaning
O	the set of data points (clients)
r	the coverage radius of a facility
$dist(o_i, o_j)$	the distance between two points
$C_{i,j}$	the grid cell on ith row and jth column
n	the size of O
k	the parameter of k-means clustering
Δ	the increment step size of k
p	the center of a coverage set
P	a set of facilities
CS	a coverage set
$\mathbf{\Theta}$	a set of coverage sets

3 Problem Statement

We first present a few basic concepts and a problem definition. Frequently used symbols are listed in Table 1.

We consider a point set O in a 2-dimensional Euclidean space representing the clients, and use the Euclidean distance to measure the distance between two points p and o, denoted by $d(p, o)$.

Given a constant $r \in \mathcal{R}^+$ and two points p and o, where p represents a facility location and o represents a client, we say that p *covers* o if their distance $d(p, o)$ is less than or equal to r. We call r the *coverage radius* of p. The set of clients covered by p is called a *coverage set*, denoted by CS.

Definition 1 (**Coverage Set**) Given a coverage radius $r \in \mathcal{R}^+$ and a point set O, a *coverage set* CS is a subset of O, such that every point in CS can be covered by the same point p.

The *Group OptimAl Location* (GOAL) problem computes a minimum number of coverage sets, such that the union of them is the set O.

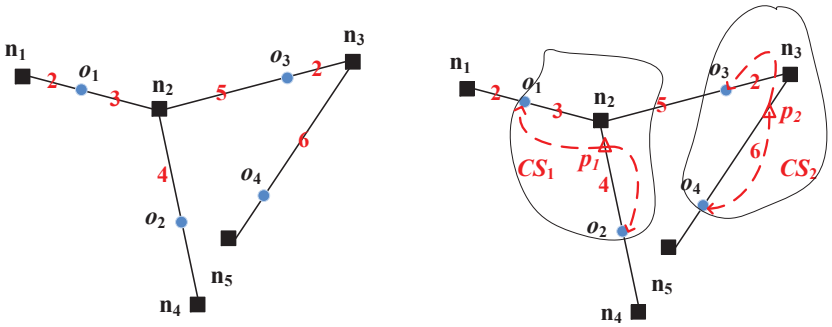
Definition 2 (**GOAL**) Given a coverage distance $r \in \mathcal{R}^+$ and a point set O, the *Group OptimAl Location* problem finds the minimum number of disjoint coverage sets such that the union of these coverage sets is O, and returns this number as well as a corresponding set of coverage sets.

Formally, let $\mathbf{\Theta} = \{CS_1, CS_2, ..., CS_m\}$ be the set of coverage sets returned. This set satisfies that

(i) $\forall i, j \in [1, m], i \neq j : CS_i \cap CS_j = \emptyset$;
(ii) $\bigcup_{i=1}^{m} CS_i = O$;
(iii) For any other set $\mathbf{\Theta'}$ satisfying the two conditions above, $|\mathbf{\Theta}| <= |\mathbf{\Theta'}|$.

As discussed in Section 1, the GOAL problem is NP-hard [14] [33], and the answer coverage sets are not unique. Our focus is to find an approximate solution with a small number of coverage sets. For example, as shown in Fig. 2

(a), given a client data set O=$\{o_1, o_2, o_3, o_4, o_5, o_6, o_7, o_8\}$, a possible problem answer is $\mathbf{\Theta} = \{CS_1, CS_2\}$ as shown in Fig. 2 (b), and $|\mathbf{\Theta}|$=2.



(a) The road network (b) A possible result

Fig. 3 Example of the GOAL problem in a road network

We also consider the GOAL problem in road networks, where a road network is modeled by a graph $G = \langle V, E \rangle$ embedded in a Euclidean space. Here, V denotes a set of vertices (intersections in the road network) and E denotes a set of edges (roads). The edge length is computed as the Euclidean distance between the two vertices of the edge. The set of objects O are on the edges (or at the vertices) of the graph. The GOAL problem in road networks has a similar definition to that in Euclidean space, except that now the distance between a facility and a client is computed as their graph shortest path distance.

For example, as shown in Fig. 3 (a), given a road network where n_1, n_2, n_3, n_4, n_5 are the vertices and the length of each edge are shown on the edges, a coverage radius $r = 4$, and a client data set $O = \{o_1, o_2, o_3, o_4\}$, a possible set of coverage sets is shown in Fig. 3 (b), where $\mathbf{\Theta} = \{CS_1, CS_2\}$, and $|\mathbf{\Theta}| = 2$.

4 Algorithms

The overall procedure of our algorithms is as follows. We run a clustering algorithm to obtain an upper bound on the number of disjoint subsets needed to fully cover the data set where each cluster can be enclosed by an r-radius circle. The clusters obtained form our initial problem answer. Then we apply a two-stage merging procedure to further refine the coverage sets obtained. The first stage uses a grid to partition the data space and merges coverage sets in nearby grid cells. The second stage further merges the coverage sets through examining different pairs of coverages sets. Two strategies are used – greedy and dynamic programming – to reduce the number of pairs of coverage sets to be considered, resulting in two algorithms GOAL-Greedy and GOAL-DP, respectively. Figure 4 summarizes the overall algorithm procedure.

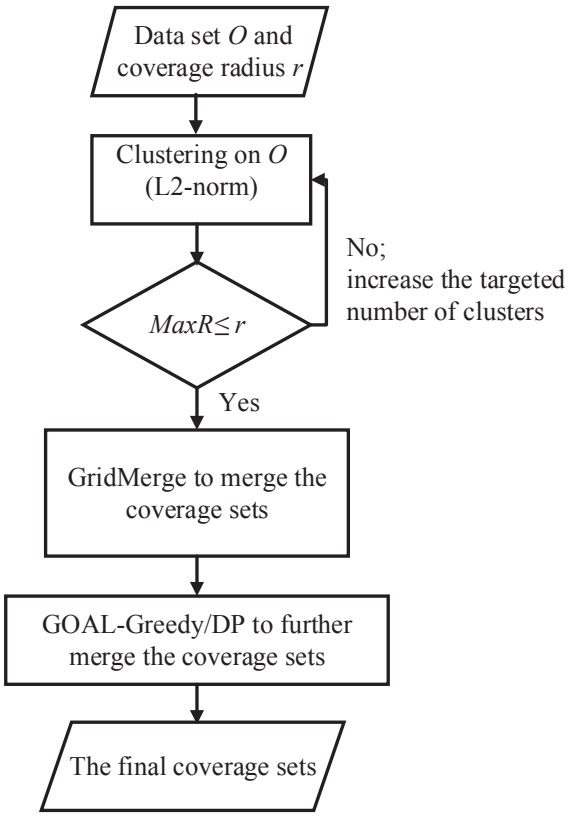


Fig. 4 Algorithm overview ($maxR$ is the $maxDist$ between two points in a cluster)

4.1 Clustering

We compute an upper bound of the number of coverage sets needed by running a clustering algorithm over the client data set O. We use the k-means clustering algorithm for simplicity, although other clustering algorithms such as DBSCAN may be used as well. Specifically, we run k-means with a predefined initial value of k over O. We compute the *inner maximum distance* for each cluster obtained, i.e., the maximum distance between any two objects in a cluster, denoted by $maxR$. If $maxR \leq r$ holds for every cluster, the clusters obtained satisfy the GOAL problem, and each forms a coverage set. We draw a circle centered at the geometric center of the clients in a coverage set with radius r (cf. the solid circles in Fig. 2(b)). This circle encloses all the clients in the coverage set, and a facility set up at its center can cover all the clients in the coverage set. We call this circle a *coverage set circle* and use it to represent a coverage set in the figures in the rest of the paper. If $maxR > r$ for some cluster, we increase k by Δ which is a empirically learned parameter. We repeat the process above until $maxR \leq r$ for every cluster obtained.

When the clustering algorithm terminates, the clusters obtained serve as our initial answer coverage sets. The value of k at termination is an upper bound of the number of coverage sets required to fully cover all the clients.

4.2 Grid Partition Based Coverage Set Merging

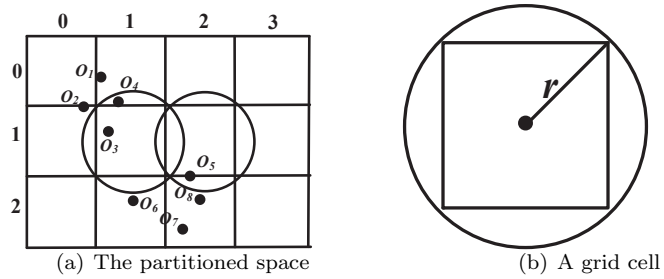
Since $maxR \leq r$ is a relaxed bound, we may not need all the initial coverage sets but can merge some of them to obtain fewer coverage sets.

We use a grid partition to refine the coverage sets as follows. We partition the space using a grid where each cell is a square with an edge length of $\sqrt{2}r$, as shown in Fig. 5 (a). For ease of discussion we assume that the data space can be partitioned into integer numbers of rows and columns. The last row and column of the grid may contain partially empty space but this does not affect the correctness of the algorithm. This partition guarantees that all points in the same cell can be covered in a coverage set, because a cell is fully enclosed in a circle sharing the same center with the cell with a radius of r, as shown in Fig. 5(b).

The advantages of the grid partition are: i) only the data points in the same or adjacent cells can form a coverage set which can help prune the search space for coverage set merging; ii) for a given coverage set, we only need to check its adjacent cells for coverage set refinement by merging with a neighboring coverage set or exchanging some clients with a neighboring coverage set.

We present two lemmas to support coverage set merging based on the grid partition.

4.2.1 Coverage Sets Not to Be Merged



(a) The partitioned space (b) A grid cell

Fig. 5 Grid partition

First, we discuss the case where two coverage sets are not to be merged. Let $C_{i,j}$ be the grid cell at row i, column j. The cells within $2\sqrt{2}r$ distance of $C_{i,j}$ (including $C_{i,j}$ itself) form the *adjacent region* of $C_{i,j}$, i.e., the adjacent

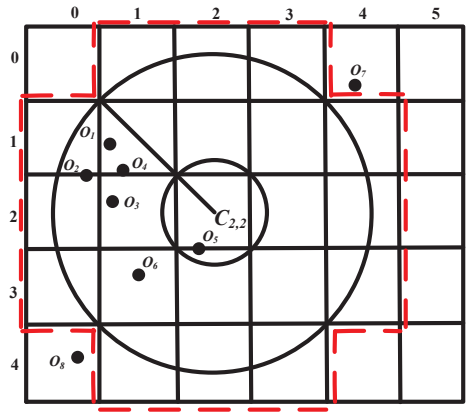


Fig. 6 Adjacent Region

region is a set $\{C_{x,y}|i-2 \leq x \leq i+2, j-2 \leq y \leq j+2\}$. The region enclosed by the dashed line boundary in Fig. 6 exemplifies the adjacent region of $C_{2,2}$.

The adjacent region of $C_{i,j}$ defines a region where a client o' must be in if o' and some client $o \in C_{i,j}$ belongs to the same coverage set. This is formulated as the following lemma.

Lemma 1 Given two client points $o \in C_{i1,j1}, o' \in C_{i2,j2}$, if $C_{i2,j2}$ is not in the adjacent region of $C_{i1,j1}$, then o, o' cannot be in the same coverage set.

Proof Suppose that the centers of $C_{i1,j1}, C_{i2,j2}$ are $c_1(x_1, y_1)$ and $c_2(x_2, y_2)$. Since $C_{i2,j2}$ is not in the adjacent region of $C_{i1,j1}$, we know that $i2 < i1 - 2 or i2 > i1 + 2 or j2 < j1 - 2 or j2 > j1 + 2$. Suppose that $i2 < i1 - 2$ holds (same argument holds for the other cases). Then, according to the grid cell size,

$$\frac{x_2}{\sqrt{2}r} < \frac{x_1}{\sqrt{2}r} - 2 \Rightarrow |x_1 - x_2| > 2\sqrt{2}r \tag{1}$$

Since $c_1(x_1, y_1)$ and $c_2(x_2, y_2)$ are the centers of two cells, we also have

$$|y_1 - y_2| > 2\sqrt{2}r \tag{2}$$

Thus,

$$dist(c_1, c_2) > 4r \tag{3}$$

For any $o \in C_{i1,j1}, o' \in C_{i2,j2}$, we have

$$dist(o, c_1) < r \wedge dist(o', c_2) < r \tag{4}$$

Also,

$$dist(c_1, c_2) \leq dist(o, c_1) + dist(o', c_2) + dist(o, o') \tag{5}$$

Based on Inequalities (3), (4), and (5), we have

$$dist(o, o') > dist(c_1, c_2) - 2r > 2r \tag{6}$$

Thus, o, o' cannot be in the same coverage set. ■

According to Lemma 1, if two coverage sets are not in the adjacent region of each other, they should not be merged into a coverage set.

4.2.2 Coverage Sets to Be Merged

Next, we discuss two cases where two coverage sets can be merged together to form a single coverage set, and hence reduce the number of coverage sets.

The first case states that two coverage sets can be merged if their centers are sufficiently close.

Lemma 2 Given two coverage sets CS_1 and CS_2 centered at c_1 and c_2, if $dist(c_1, c_2) \leq r - max\{maxR(CS_1), maxR(CS_2)\}$, then CS_1, CS_2 can be merged into one coverage set.

Proof Without loss of generality we assume that

$$max\{maxR(CS_1), maxR(CS_1)\} = maxR(CS_1)$$

It is sufficient to show that for any client point o in CS_1, its distance to the center of CS_2 does not exceed r, meaning that o can be merged into CS_2:

$$\forall o \in CS_1, dist(o, c_2) \leq r \tag{7}$$

Based on triangular inequality,

$$dist(o, c_2) \leq dist(o, c_1) + dist(c_1, c_2) \tag{8}$$

Meanwhile,

$$dist(o, c_1) \leq maxR(CS_1) \tag{9}$$

If

$$\begin{aligned} dist(c_1, c_2) \leq r - max\{maxR(CS_1), maxR(CS_2)\}, \\ then, dist(c_1, c_2) \leq r - maxR(CS_1) \end{aligned} \tag{10}$$

From Inequalities (8), (9), and (10), we derive

$$\forall o \in CS_1, dist(o, c_2) \leq maxR(CS_1) + r - maxR(CS_1) = r \tag{11}$$

Thus, all the points in CS_1 can be merged into CS_2 safely. ■

The next case for merging is based on a concept named the *reachable region*. The reachable region of a grid cell $C_{i,j}$ is a square that has the same center as $C_{i,j}$, but with an edge length of $2\sqrt{2}r$. The dashed line square in Fig. 7 exemplifies the reachable region of $C_{1,1}$. The four vertices of the reachable region of $C_{i,j}$ are the centers of the four cells $C_{i-1,j-1}$, $C_{i-1,j+1}$, $C_{i+1,j-1}$, and $C_{i+1,j+1}$, i.e., $C_{0,0}$, $C_{0,2}$, $C_{2,0}$, and $C_{2,2}$.

Lemma 3 Let $\mathcal{CS}_{i,j}$ be the set of coverage sets whose centers are all in a cell $C_{i,j}$. Let $\mathcal{CS}_R$ be the set of coverage sets whose centers are in the reachable region of cell $C_{i,j}$ (excluding the coverage sets in $\mathcal{CS}_{i,j}$). Define CS_0 as a coverage set that consists of the points in a circle centered at the center

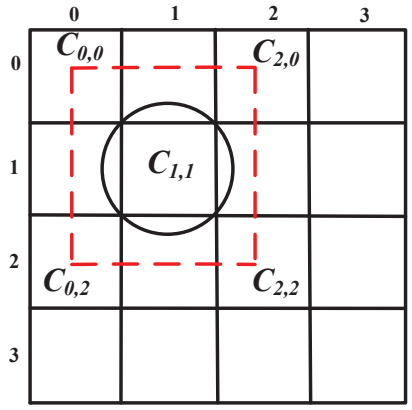


Fig. 7 Reachable Region

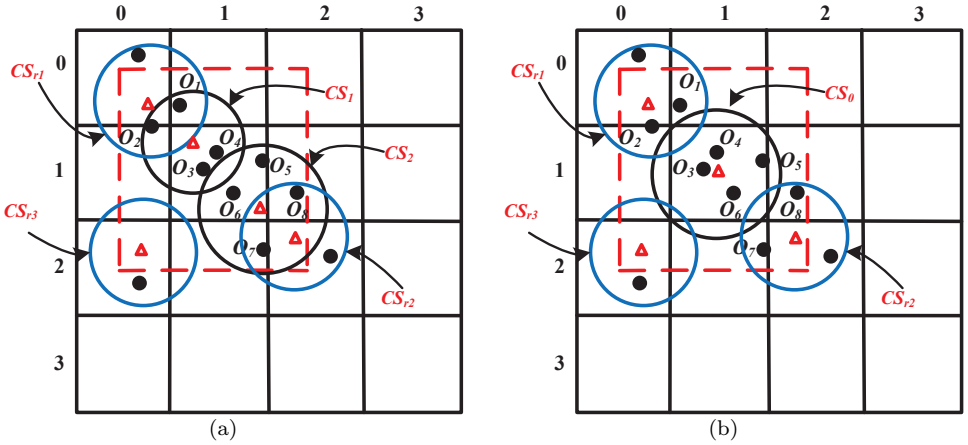


Fig. 8 Examples of reachable region based merging

of $C_{i,j}$ with a radius of r. If each of the points in $\mathcal{CS}_{i,j} \setminus CS_0$ can be covered in some coverage set in $\mathcal{CS}_R$, we can safely move these points to the coverage sets in $\mathcal{CS}_R$, keep CS_0, and remove all the coverage sets in $\mathcal{CS}_{i,j}$.

Proof Figure 8 illustrates the lemma. In Fig. 8(a), there are two coverage sets $CS_1 = \{o_1, o_2, o_3, o_4\}$ and $CS_2 = \{o_5, o_6, o_7, o_8\}$ in cell $C_{1,1}$. There are coverage sets CS_{r1}, CS_{r2} and CS_{r3} in the reachable region of $C_{1,1}$. The clients o_3, o_4, o_5, o_6 are all in $C_{1,1}$, while o_1, o_2 are enclosed by the coverage set circle of CS_{r1}, and o_7, o_8 are enclosed by the coverage set circle of CS_{r2}. We can safely merge o_3, o_4, o_5, o_6 as a coverage set CS_0, while adding o_1, o_2 to CS_{r1} and o_7, o_8 to CS_{r2}, as shown in Fig. 8(b). This way we reduce the number of coverage sets.

A formal proof of the correctness of the lemma is straightforward based on the edge length of the cells and the coverage radius of the circles. We omit it for conciseness.

Algorithm 1: GridMerge

```
   Input: initial coverage set collection Θ
   Output: refined coverage sets Θ*
 1 count = 0
 2 while count < IterationNum do
 3     Sort the grid cells, get the sorted queue QC
 4     while QC is not null do
 5         Get a Cell C from QC
 6         for every coverage set CS_i in C do
 7             for every point o in CS_i do
 8                 if o is only covered by CS_i then
 9                     Put CS_i into Θ*
10                 flag=true;
11                 break;
12             if flag==true then
13                 continue; //goto the next coverage set
14             for every coverage set CS_j (j > i) in Ω, Ω is the adjacent region of C)
                do
15                 if CS_i and CS_j satisfy Lemmas 2 or 3 then
16                     CS_j ← CS_i ⋃ CS_j
17                     delete CS_i from Θ
18                     flag=true;
19                     break;
20         if flag==true then
21             continue;
22     count ++;
23 Add the remaining sets in Θ to Θ*
24 return Θ*
```

4.3 All Coverage Set Pair Based Merging

We summarize the grid based merging process as the GridMerge algorithm. The pseudo-code of the algorithm is shown in Algorithm 1. The algorithm first sorts the grid cells based on the number of coverage sets centering at them in the descending order. Let QC be the sorted queue of grid cells (line 3). Merge processes the cells in QC one at a time (lines 4 to 19). For each coverage set in cell $C_{i,j}$ being processed, we check if the clients in the set is also enclosed by the coverage set circle of some other coverage sets. If not then we put this coverage set into the final problem answer $\mathbf{\Theta}^*$ (lines 7 to 13). Otherwise, we merge the coverage sets based on Lemmas 2 and 3 (lines 14 to 19). Note that after processing all cells in the grid, new coverage sets may be formed which

may need further merging. We thus run the algorithm iteratively until the coverage sets become stable or a preset number of iterations has been reached (line 2). Our experiments show that 5 iterations are sufficient for the data sets tested.

The grid based merging is simple and effective, but it is also conservative. We further refine the coverage sets via checking them pair by pair. We use two strategies to avoid checking all pairs, resulting in two algorithms, GOAL-Greedy and GOAL-DP, respectively.

4.3.1 GOAL-Greedy

Algorithm 2: GOAL-Greedy

```
   Input: set O, cover radius r
   Output: coverage sets collection Θ and |Θ|
1  while maxR > r do
2      k = k + Δ;
3      Run k-means on data set O, get the clustering sets Θ*;
4  Build a grid over the coverage sets Θ* obtained by clustering
5  Call GridMerge over the initial coverage sets and get the merged coverage sets Π;
6  Sort Π in descending order;
7  for every coverage set CS_i in Π do
8      for every coverage set CS_j in the reachable region of CS_i do
9          for every point o in CS_j do
10             if o ∈ CS_i then
11                 Delete o from CS_j;
12         if CS_j is NULL then
13             delete CS_j from Π;
14 Θ = Π;
15 return Θ and |Θ|;
```

GOAL-Greedy examines the coverage sets in the descending order of their numbers of client points. For a coverage set CS_i being examined, we fetch all the client points from the coverage sets that have not been examined, and add them to CS_i if they are within the coverage set circle of CS_i. The added client points are removed from their original coverage sets. If a coverage set becomes empty after the client point removal, we remove this coverage set. Note that we can pre-label the clients that are covered by multiple coverage set circles with a sequential scan to reduce the computational costs.

We present the full procedure of GOAL-Greedy in Algorithm 2. First, we run the k-means algorithm iteratively to get the initial coverage sets (lines 1∼3). At each iteration we increase parameter k by Δ, which is a predefined increment step size. We will study the choice of initial value of k and the value of Δ empirically in Section 6.1. When $maxR \leq r$ is reached, we terminate the clustering process, and use the clusters obtained as the initial coverage sets.

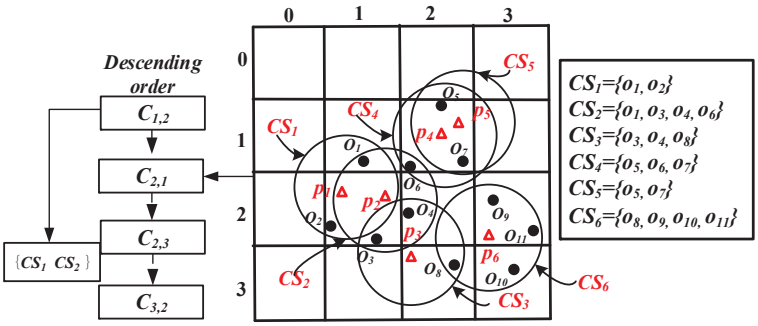
Then we build the grid over the centers of the coverage sets (line 4), and call GridMerge for an initial merging of the coverage sets. From line 6 to line 13, we use a greedy strategy to further remove the redundant coverage sets as described above. In what follows, we use an example to illustrate the whole procedure of GOAL-Greedy.

Example 1 As shown in Fig. 9(a), the client data set $O = \{o_1, o_2, o_3, o_4, o_5$ $o_6, o_7\}$. Suppose that the initial coverage sets derived from running k-means is $\mathbf{\Theta}_{initial} = \{CS_1, CS_2, CS_3, CS_4, CS_5, CS_6\}$, and the centers of these coverage sets are $p_1, p_2, p_3, p_4, p_5, p_6$, as denoted by the triangles. First, the gird cells are sorted by the number of clusters centers. In Fig. 9(a), the descending order of the cells is $C_{1,2}, C_{2,1}, C_{2,3}, C_{3,2}$, because $C_{1,2}$ and $C_{2,1}$ both contains two centers, while $C_{2,3}$ and $C_{3,2}$ only contains one center. Based on Lemma 3, CS_1, CS_2 can be merged to one coverage set CS_7, as shown in Fig. 9(b), where o_1, o_2, o_3, o_4 are covered by the new coverage set CS_7 centered at the center of cell $C_{2,1}$. Client o_6 is merged to the adjacent coverage set CS_4. We get an intermediate result $\mathbf{\Theta}_{middle} = \{CS_3, CS_4, CS_5, CS_6, CS_7\}$. Then $C_{2,1}$ is processed. Based on Lemma 2, we can merge CS_4, CS_5 to coverage set CS_4. The result is shown in 9(c). And next, we run a greedy algorithm to further merge the coverage sets. The descending order of coverage sets are $\{CS_6, CS_7, CS_3, CS_4\}$. The algorithm checks CS_6 at first, and o_8 is moved to CS_6 and deleted from CS_3. Next o_3 and o_4 are deleted from CS_3, and since CS_3 is empty now, we can remove it from the answer set. The final coverage sets are $\mathbf{\Theta} = \{CS_4, CS_6, CS_7\}$ as shown in Fig. 9(d), and $|\mathbf{\Theta}| = 3$.

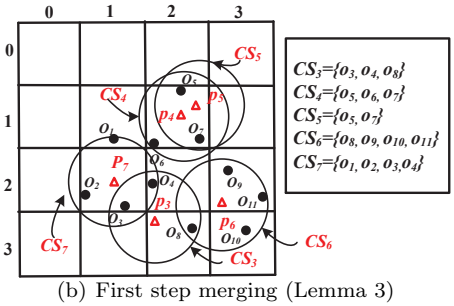
GOAL-Greedy uses a greedy approach to merge the coverage sets which is efficient, but may fail to find an optimal merging order of the coverage sets. For example, in Fig. 10(a), there are seven client points $\{o_1, o_2, ..., o_7\}$ forming three coverage sets after running GridMerge. Based on GOAL-Greedy, coverage set CS_1 will be processed at first, and points $\{o_1, o_2, o_3, o_4\}$ will be deleted from CS_2, CS_3. However,$\{o_5, o_6\}$ can only be covered in CS_2, so CS_2 has to be put into the final result, and this is same for CS_3. Thus, the answer set found by GOAL-Greedy is $\{CS_1, CS_2, CS_3\}$. However, if we process coverage sets CS_2 and CS_3 first, then CS_1 can be removed from the result which derives a more concise result $\{CS_2, CS_3\}$, as shown in Fig. 10(b). Next, we present a Dynamic Programming based algorithm to obtain more concise coverage sets.

4.3.2 GOAL-DP

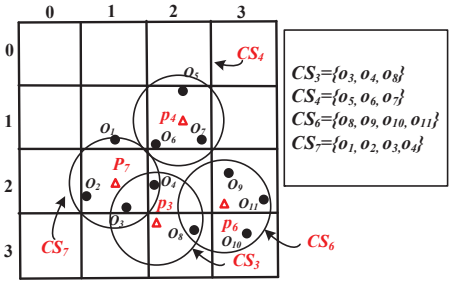
To find a better merge of the coverage sets, we consider all the possible merging of coverage sets. Suppose that the number of client points is n, and the minimum number of facilities needed to provide a full coverage is $minNum$. We use $\mathbf{\Theta}$ to represent the optimal coverage set collection, and $\mathit{\Pi} = \{CS_1, CS_2...CS_m\}$ are the coverage sets derived from merging. The GOAL-DP algorithm decomposes the GOAL problem into two sub-problems: evaluating a sub- problem $GOAL(O', i)(O' \subseteq O)$ which returns the number of coverage sets needed to solve the sub-problem, where $i = 1, 2...m$, and selecting min coverage sets



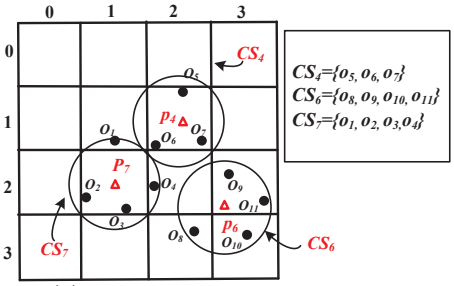
(a) Initial distribution



(b) First step merging (Lemma 3)



(c) First step merging (Lemma 2)



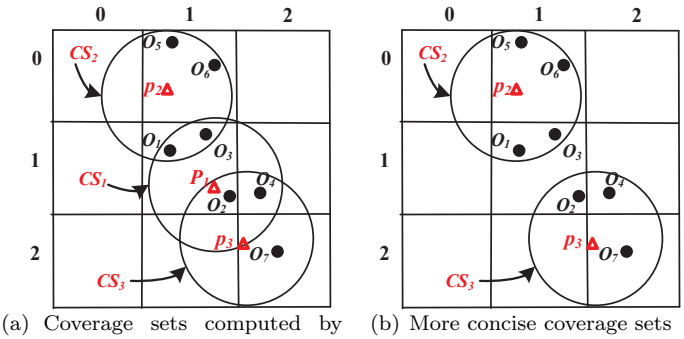
(d) Second step merging by Greedy

Fig. 9 Procedure of GOAL-Greedy

which contain objects whose number is not less than the number of objects in *min* other coverage sets (i.e., at least a coverage set is different). When *min* coverage sets contain all the objects, which means $O' = O$, then *min* is minimal. Thus the Dynamic Programming equitation can be described as,

$$GOAL(\emptyset, 0) = 0; \tag{12}$$

$$GOAL(O' \neq \emptyset, 0) = \infty; \tag{13}$$



(a) Coverage sets computed by Merge (b) More concise coverage sets

Fig. 10 Limitation of greedy merging

$$GOAL(O', i) = min(GOAL(O' - CS_i, i-1) + 1, GOAL(O', i-1)); \quad (14)$$

The reasoning is quite straightforward: in the latest equation we can either use or not use set CS_i. In case we decide to use it, we still have to cover set O',using $CS_1, CS_2, ...$ But we have already used CS_i, so we have to add 1 to the overall result. On the other hand we may decide not to use CS_i. In this case we have to cover the entire set O' with remaining sets. Clearly we take minimum out these two values. Algorithm 3 summarizes the above procedure.

Algorithm 3: GOAL-DP

Input: set O, cover radius r
Output: coverage sets collection $\mathbf{\Theta}$ and $minNum$
1 **while** $maxR > r$ **do**
2 $k = k + \Delta$;
3 Run k-means on data set O, get the clustering sets $\mathbf{\Theta}^*$;
4 Build a grid over the coverage sets $\mathbf{\Theta}^*$ obtained by clustering
5 Call *GridMerge* over the initial coverage sets and get the merged coverage sets Π;
6 initial the status table;
7 **for** *every coverage set* CS_i *in* Π **do**
8 take CS_i into O' (use O' to record the temporal client points set been covered);
9 $GOAL(O', i) = min(GOAL(O' - CS_i, i-1) + 1, GOAL(O', i-1))$;
10 $minNum = GOAL(O', m); (O' = O)$
11 $\mathbf{\Theta} = O'$;
12 return $minNum$, $\mathbf{\Theta}$;

The overall procedure of GOAL-DP is similar to GOAL-Greedy except the DP procedure described as line 6 to line 10 in Algorithm 3. Next, we use an example to illustrate the whole procedure of GOAL-DP.

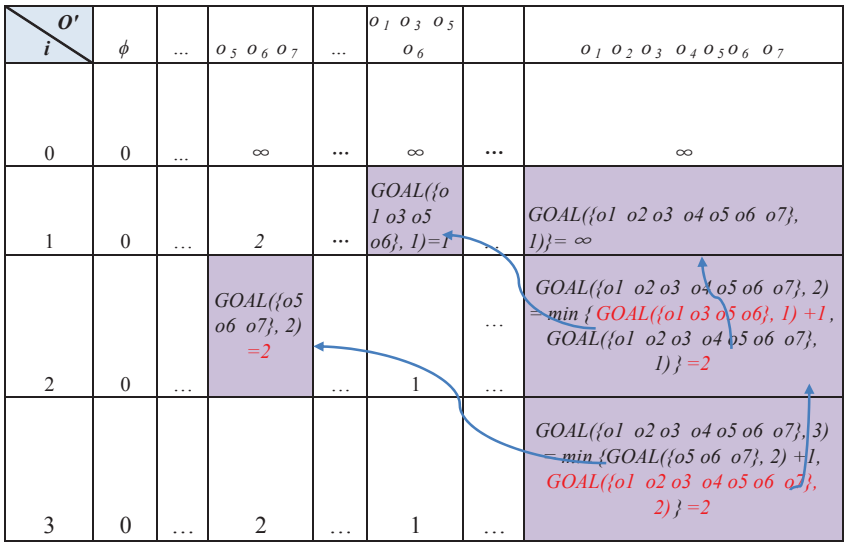


Fig. 11 Status table of DP

Example 2 We still use the example shown in Fig. 10, suppose that Fig. 10(a) is the result derived from GridMerge procedure. We conclude the status table of DP algorithm in Fig. 11. Each column describes a possible subset of O, i.e., set $O^{'}$. Each row describes a status, status i means that coverage set CS_i is now discussed, for example, status 0 means that no coverage set is considered, and status 1 means consider coverage set CS_1. It's obvious that the result found by GOAL-DP is the minimum number in the last column, which is $\{ CS_2, CS_3 \}$ with minimum coverage set number 2, as shown in Fig. 11.

4.4 GOAL Algorithm for Road Networks

We extend our solution to GOAL in road networks, where we use the lemma below to prune coverage sets from being merged.

Lemma 4 Given a point o and a coverage set CS in a road network, suppose that the center of the coverage set is c. If the Euclidean distance $dist(o, c) > r$, then o cannot be in CS.

Proof Since for any two points their Euclidean distance does not exceed their road network distance [14], we know $dist_{rn}(o, c) > dist(o, c)$, and hence $dist_{rn}(o, c) > r$, where $dist_{rn}()$ denotes the the network distance. Therefore, o must not be in CS.

Based on Lemma 4, we can use the Euclidean Distance to derive the initial coverage sets (as shown in Fig. 12) as what has been described in Section 4.1. This step can help us avoid computing road network distance between

the clients for clustering. Then, for each coverage set, we compute the road network shortest path distance between a client point and the coverage set center (the vertex on the road network that is closest to the geometric center of the clients in the coverage set), and remove the clients that are not covered by the center according to the network shortest path distance (as shown in Fig. 12, we call this procedure *Refinement*). At last, we call DP algorithm for final coverage set adjustment. We omit the pseudo-code of GOAL-RN, since it is similar to GOAL-DP described above. We only illustrate the Refinement procedure of GOAL-RN in Example 3, since the other steps are similar to GOAL-DP.

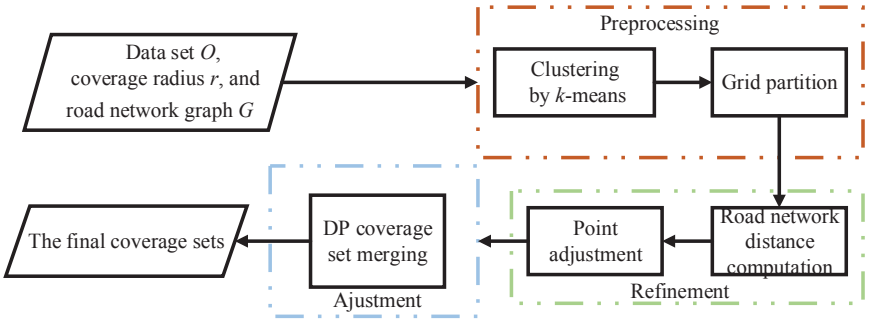


Fig. 12 Procedure of GOAL-RN

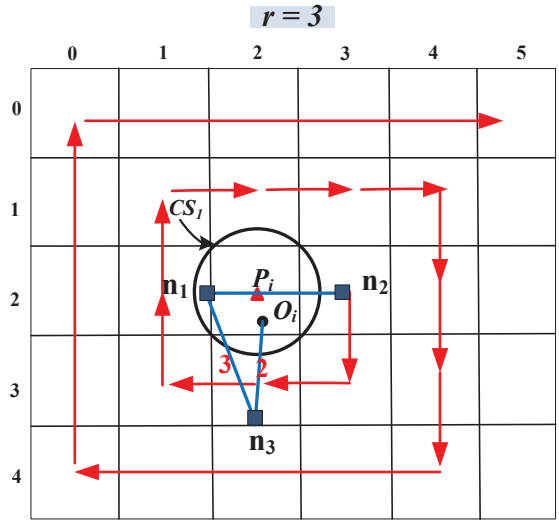


Fig. 13 Example of GOAL-RN refinement

Example 3 Suppose a coverage set CS_1 locates at grid cell $C_{1,1}$, where its center p_i is shown in Fig. 13; client o_i is covered by p_i; and $dist(o_i, p_i) < r$. We then compute $dist_{rn}(o_i, p_i)$. Note that, it is possible that p_i is not on the road network G, since p_i is a cluster center. In that case, we replace p_i by the vertex on G that is nearest to it, which is n_1 in Fig. 13. We have $dist_{rn}(o_i, p_i) = 5 > r$, and o_i is removed from CS_1. We will search for a new nearest facility location p_j (center of another coverage set center) for o_i in the order of the arrow line as shown in Fig. 13 which is from the closest cells of $C_{1,1}$ to the cells farther away. We compute $dist_{rn}(o_i, p_j)$ until it is not larger than r, and then put o_i in the coverage set of p_j. If all coverage sets have been explored and none of them can cover o_i, then o_i will form a new coverage set with itself.

5 Algorithm Analysis

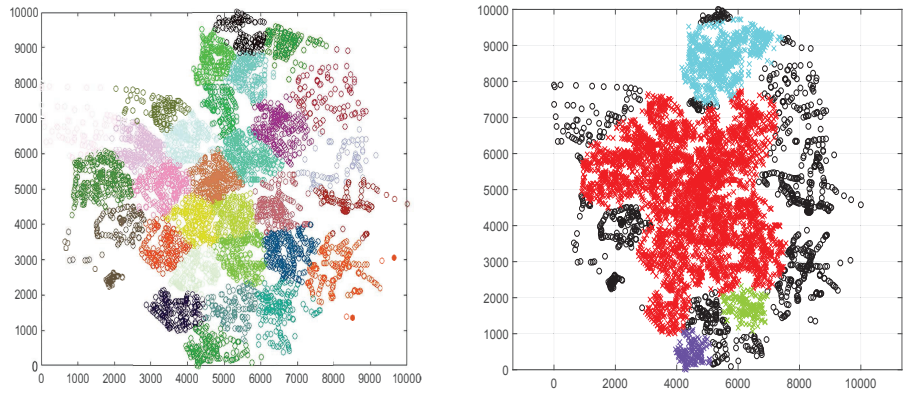
5.1 Choice of Clustering Algorithm

The proposed algorithms do not rely on any particular clustering algorithm. We use k-means for its simplicity. Additionally, compared with other popular clustering algorithms such as DBSCAN [11], Hierarchical Clustering [27], and Gaussian Mixture Model (GMM) [16] [13], k-means suits our problem setting better in the following aspects. First, it naturally creates spherical clusters, which form coverage sets nicely. Second, it is more stable on different coverage radius values. In comparison, DBSCAN is more sensitive to the clustering radius, and it is difficult to form coverage sets with a particular coverage radius. To show this we run k-means and DBSCAN on the OL data set (described in Section 6) and plot the result in Fig. 14. In the figure, each cluster formed is represented by a different color. We see that DBSCAN creates clusters with much more different sizes, while k-means creates clusters with similar sizes.

5.2 Optimality of Algorithm Output

As described in Section 4, each step of the three proposed algorithms can ensure that all the points in one coverage set can be covered by a facility at the coverage set center. Therefore, all the algorithms guarantee that all client points are covered.

Next, we analyze how close the answers produced by GOAL-Greedy and GOAL-DP are to the *optimal* problem answer with the minimum answer set size. Let OPT be the minimum number of coverage sets required to cover the whole client set, and k be the number of initial coverage sets produced after clustering the clients. Let G_{greedy} be the number of coverage sets that are reduced by GOAL-Greedy. Then the approximation ratio of the GOAL-Greedy algorithm is $\frac{OPT}{k-G_{greedy}}$. Similarly the approximation ratio of the GOAL-DP algorithm can be represented by $\frac{OPT}{k-G_{dp}}$ where G_{dp} represents the number of coverage sets that are reduced by GOAL-DP.



(a) k-means

(b) DBSCAN

Fig. 14 Clusters produced by k-means and DBSCAN on the OL data set

Ideally, if the initial coverage sets at different grid cells can be processed independently, the grid cell based coverage set merging may achieve a good coverage set reduction. However, this case is unlikely since there may be coverage sets overlapping with more than one grid cells. In the worst case, the greedy algorithm can be $|O|$-approximation algorithm.

There is no closed form equation to model the exact approximation ratio of GOAL-Greedy or GOAL-DP, but GOAL-DP is expected to produce a smaller number of coverage sets because it considers more combinations in merging the coverage sets.

Based on the clustering result, we can use a enumerate method to derive the optimal coverage sets. As we can see the enumeration algorithm is a brute-force algorithm. In the worst case, let K be the number of initial coverage sets produced after clustering, the enumeration algorithm will check $\binom{n}{K}$ combinations to get the optimal result, where n is an integer satisfying $n \in [1, K)$. And let m be the maximum size of influence set of a coverage set. The time complexity of checking each combination is $O(n^2 \cdot m)$. In the worst case, the cost of the proposed algorithm is $O(n^2 \cdot m \cdot \binom{n}{K})$. Thus, we only do experiments on small size data sets to demonstrate the two proposed algorithms' approximation ratio. For large data sets, we only use the answer set size to illustrate the algorithms' accuracy, since the algorithms accuracy increases with the decrease of the answer set size.

5.3 Time Complexity

Both GOAL-Greedy and GOAL-DP contain the following three steps.

Step 1 (Clustering) We adopt the k-means clustering algorithm to compute initial coverage sets. The time complexity of this step is $O(|O| \cdot (k + k \cdot \Delta + k \cdot 2\Delta + ... + k(t-1)\Delta) \cdot t) = O(|O| \cdot k \cdot (t^2/2) \cdot \Delta)$, where $|O|$ denotes the number of client, k is the initial clustering parameter, Δ is the increment step size and t is the number of iterations that k-means is run.

Step 2 (Grid partition and coverage set merging) We use a grid to index the center of the initial coverage sets, such that when merging the coverage sets, we only need to consider those in the adjacent or reachable regions of each other. Suppose that the number of adjacent or reachable regions of all the grid cells is w. Then the time complexity of this step is $O(|O| + \alpha \cdot w \cdot |O|)$, where α denotes the percentage of clients that are covered in more than one coverage set circles.

Step 3 (Merging of clusters) We have introduced two kinds of merging, the Greedy approach and Dynamic Programming approach. Suppose that the coverage sets number is C after Step 2, the time complexity of Greedy and Dynamic Programming are $O(C \log C)$ and $O(C^2)$ respectively.

The overall time complexity of GOAL-Greedy and GOAL-DP are $O((k \cdot t^2 \cdot \Delta + \alpha \cdot w) * |O| + C \log C)$ and $O((k \cdot t^2 \cdot \Delta + \alpha \cdot w) * |O| + C^2)$ respectively.

For the GOAL problem in road networks, our GOAL-RN algorithm contains the following three three steps:

Step 1 (Clustering) The time complexity of this step is the same as that in the Euclidean space since we use the Euclidean distance in this step.

Step 2 (Refinement and adjustment) The time complexity for road network distance computation by *Dijkstra's* algorithm is $O(|O| \cdot (|E| + |V| \cdot \log |V|))$ with an implementation based on a min-heap (where $|V|$ is the number of nodes and $|E|$ is the number of edges in G). Refinement and adjustment can be done by one pass of the coverage centers for each data points, the time complexity is $O(|O| \cdot (|E| + |V| \cdot \log |V|) + |O| \cdot k)$.

Step 3 (Merging of coverage sets) This step is the same as GOAL-DP, and the time complexity is $O(C^2)$.

The overall time complexity of GOAL-RN is $O(|O|^2 + k \cdot t^2 \cdot \Delta \cdot |O| + |E| + |V| \cdot \log |V|) + C^2)$.

6 Performance Evaluation

This section presents the experimental results. The algorithms were implemented in C++ and all experiments were performed on a computer with a 3.6GHz Intel Core i7 CPU and 32GB memory running Ubuntu 14.04.

The experiments are performed using both synthetic and real data sets. The synthetic data sets follow uniform, Gaussian, and Zipfian distributions, and the number of clients ranges from 2K to 500K. Four real world point sets LB (Long Beach), CA (California), LA (Los Angeles), and NE (Northeast

Table 2 Summary of real data sets

Dataset	Cardinality
LB	53145
CA	62556
LA	116596
NE	123593
OL	6104 nodes,7034 edges
SJ	18263 nodes,23873 edges
SF	174956 nodes,223001edges

Table 3 Experiment parameters

Parameters	Default Values
$\|O\|$	50K
Initial k	$\|O\|/300$
Δ	$\|O\|/300$

US)[1] and three real world road network data sets OL (the road network data set for City of Oldenburg), SJ (City of San Joaquin County) and SF (San Francisco)[2] are used in the experiments. Table 2 summarizes the real data sets and Fig. 6 illustrates four of them.

We compare the efficiency and the answer set size (the number of coverage sets produced) of the proposed algorithms GOAL-Greedy and GOAL-DP with two baseline algorithms GCA [33] and CREST [30] for the GOAL problem in Euclidean space . The two baseline algorithms have been described at the end of Section 2. We vary the number of clients, the data point distribution, and the value of coverage radius r in the experiments. Table 3 summarizes the default values of the parameters used.

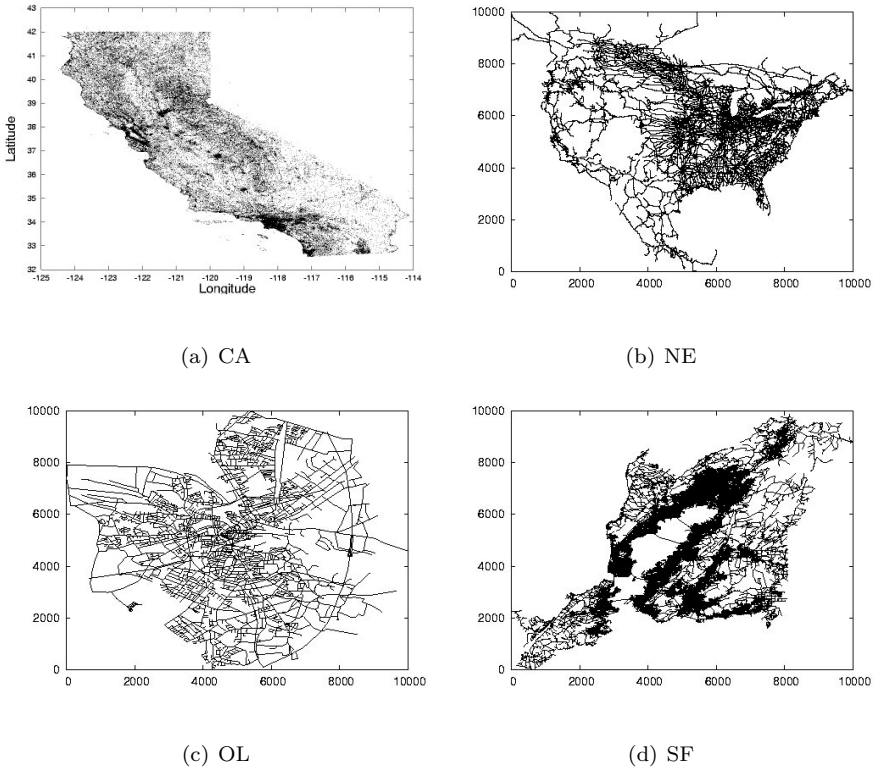
We also study the performance of GOAL-RN for the road network setting. Since GCA and CREST are incompatible with road networks, they are not used in the road network experiments.

6.1 Effect of Parameter k

We first evaluate the effect of the initial value of k in k-means. We use the synthetic data set Uniform (50K) and set the other parameters at their default values. The result shows that k has a non-trivial influence on the algorithm efficiency and answer set size. We can see from Fig. 16(a) and Fig. 16(b) that, when $n/300 < k < n/200$ (where n is the size of the client set), we can get the best performance overall.

Regarding efficiency, too large or too small k values both have negative impact on the efficiency and effectiveness of algorithms. If the k value is too

[1] http://www.dis.uniroma1.it/challenge9/download.shtml
[2] http://www.cs.utah.edu/∽lifeifei/SpatialDataset.htm

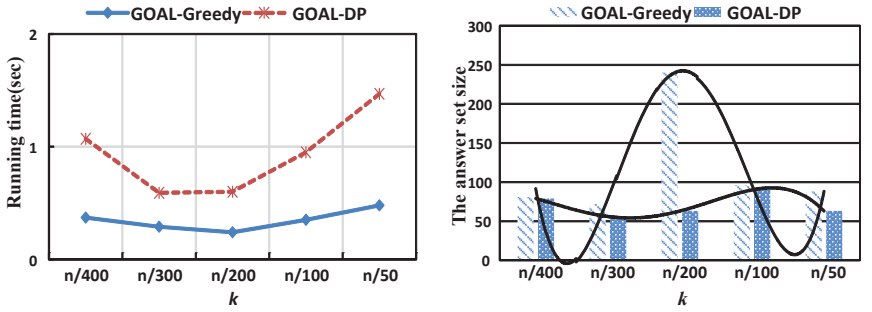


(a) CA (b) NE

(c) OL (d) SF

Fig. 15 Real world data sets

small, the clusters produced may be skewed and the clusters may have large radius. We may need to run k-means for many times to reach a satisfying radius which is time consuming. On the other hand, if k is too large, there may be lots of initial coverage sets after produced by clustering, leading to a more expensive merging step. Regarding answer set size, the impact of k is less significant. This is expected as regardless of the initial value of k, the clusters obtained as initial coverage sets are all bounded by the coverage radius. The variation among different initial value of k is expected to be small. We observe an exception for GOAL-Greedy at $k = n/200$. This is due to the greedy order of merging used by the algorithm which happened to be a less preferable order in this case. We repeat the same experiments and find a suitable of initial k value for the other data sets. For parameter Δ, it has no directly impact on the efficiency and effectiveness of algorithms, and it only has impact on k value. Thus, we tune parameter Δ and k at the same time.



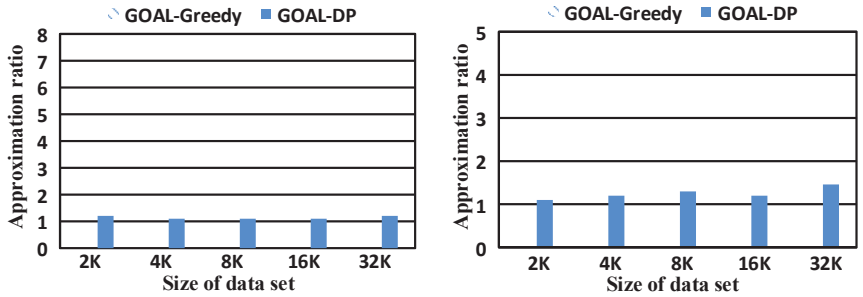
(a) Effect of initial k on efficiency (b) Effect of initial k on the answer set size

Fig. 16 Effect of k

6.2 Comparison of algorithms

6.2.1 Approximation ratio comparison

According to the analysis in Section 5.2, we can use a enumerate method to derive the optimal coverage sets. However, in this way, the cost is very high. Thus, we only do experiments on small size data sets to demonstrate the two proposed algorithms' approximation ratio. For large data sets, we only use the answer set size to illustrate the algorithms' accuracy, since the algorithms accuracy increases with the decrease of the answer set size. As shown in Figure 17, the approximation ratio of GOAL-DP algorithm is near 1, and is only a little higher than 1. And the approximation ratio of GOAL-Greedy algorithm changes dramatically with the data set size, but still keeps a constant approximation ratio with respect to the enumerate algorithm.



(a) Uniform (b) Zipfian ($\alpha = 0.8$)

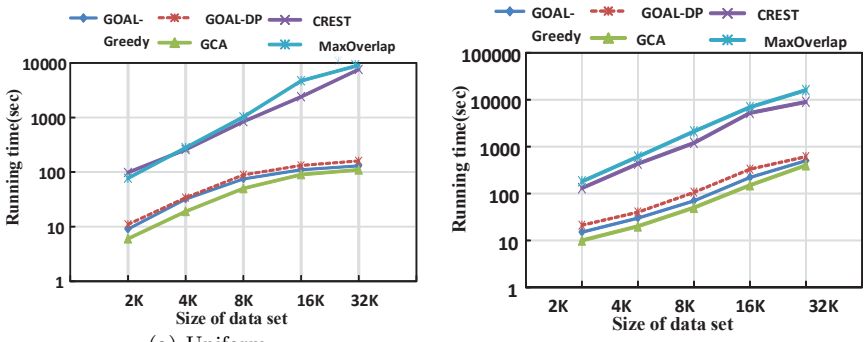
Fig. 17 Approximation ratio comparison

6.2.2 Effect of |O|

We then compare GOAL-Greedy and GOAL-DP with GCA [33], CREST [30] and MaxOverlap [20] [31](which has been modified as described in Section 2 to adapt to our problem). We use both Uniform and Zipfian data sets and vary the data set cardinality from 2k to 32k. We set the probability of a client point being covered by a facility to 1 in GCA so that it can satisfy our problem requirement. Figure 18 shows the overall algorithm running time. The result shows that both GOAL-Greedy and GOAL-DP outperform GCA, CREST and MaxOverlap, by up to 2 orders of magnitude. CREST and MaxOverlap perform the worst.

This is because CREST has to compute every disjoint region's number of RNNs and then computes the combination of the most influential regions, which is very time-consuming. And for MaxOverlap algorithm since the goal of the algorithm is finding out one optimal region or k regions to maximize the BRNN number, it cannot guarantee that all the client points can be served. The coverage sets derived by the algorithm may overlap with each other dramatically. That is to say, in the worst case, there may be some outlet point, and the MaxOverlap algorithm has to find out a huge number of optimal regions to cover it. As a result, in the rest of the experiments, we only compare with GCA, since CREST and MaxOverlap do not scale to larger data sets.

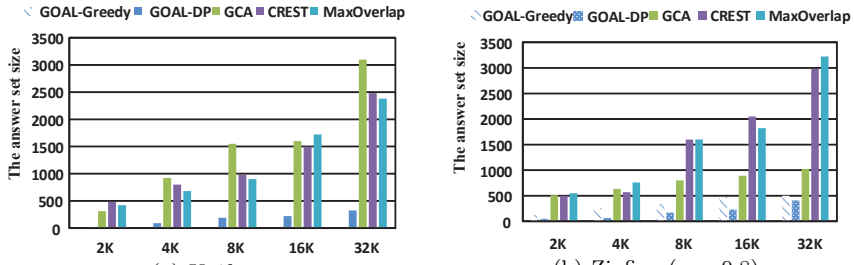
Figure 19 shows the effect of $|O|$ on answer set size. GOAL-Greedy and GOAL-DP again show significant advantage. Their answer sets are up to 80% smaller than those of GCA, CREST and MaxOverlap. This is because GOAL-Greedy and GOAL-DP have three effective merging strategies to reduce the redundant coverage sets.



(a) Uniform (b) Zipfian ($\alpha = 0.8$)

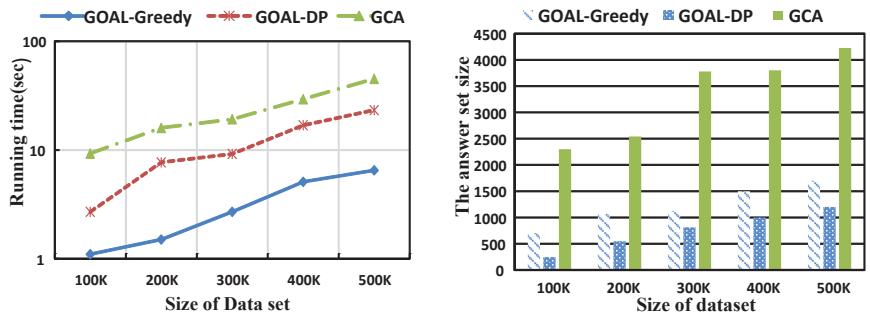
Fig. 18 Effect of $|O|$ on running time

In Fig. 20 we show the algorithm performance on even larger data sets. We only show the coverage merging time for GOAL-Greedy and GOAL-DP, since clustering is independent from merging and we may use any clustering



(a) Uniform (b) Zipfian ($\alpha = 0.8$)

Fig. 19 Effect of $|O|$ on answer set size



(a) Running time (b) Answer set size

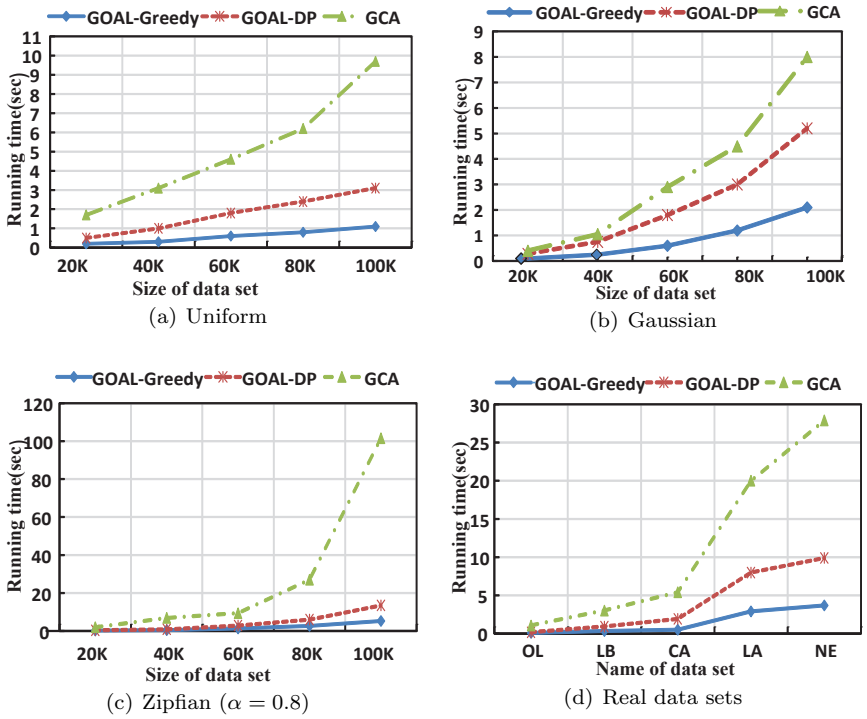
Fig. 20 Effect of data set size

algorithm to compute the initial coverage sets. For fairness of comparison we also omit the R-tree building time for GCA in the following figures.

From Fig. 20(a), we can see that, GOAL-Greedy and GOAL-DP outperform GCA significantly with the larger data sets. For GOAL-Greedy, it is up to an order of magnitude faster. Fig. 20(b) illustrates the answer set size of the three algorithms. With the increase of data set size, all the algorithms' answer set size increases. The performance of GCA degrades dramatically compared with the proposed algorithms.

6.2.3 Effect of data set distribution

Figure 21 shows the effect of data set distribution on GOAL-Greedy, GOAL-DP, and GCA. As shown in the figure, the proposed algorithms outperform GCA in running time for the various data distributions tested, and the advantage grows as the size of the data set increases. Both proposed algorithms can solve the problem within 10 seconds on all data sets. They are several times faster than GCA on Uniform and Gaussian data sets, and almost an order of magnitude faster on Zipfian and real world data sets.



(a) Uniform (b) Gaussian

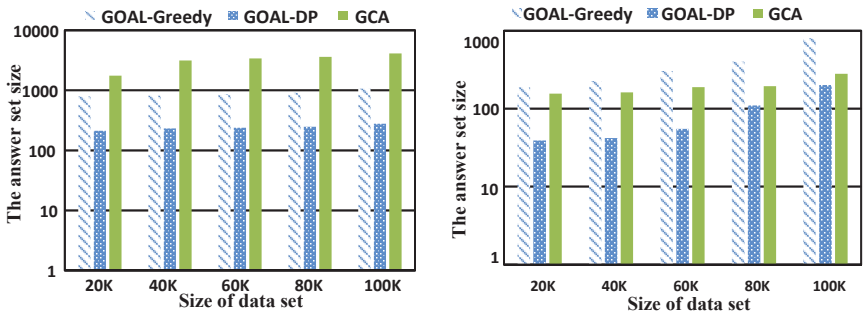
(c) Zipfian ($\alpha = 0.8$) (d) Real data sets

Fig. 21 Effect of data distribution on efficiency

We also compare the effect of data distribution on answer set size. As shown in Fig. 22, GOAL-DP gets much smaller answer set size than GOAL-Greedy in all data sets tested, and they both outperform GCA, especially on Zipfian and real world data sets In Fig. 22(c),(d), GOAL-DP shows almost two order of magnitude improvement compared with GCA. This is because these two data sets have non-uniform distributions, and in some dense area, the coverage set checking process (by the Closure Polygon Algorithm [33]) of GCA is very time consuming.

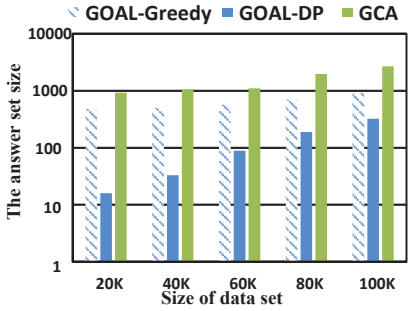
6.2.4 Effect of r

We further test the impact of the coverage radius r. Figure 23 shows the result. As expected, as r increases, both the algorithm time and answer set size decrease. This is because larger r results in more clients for each coverage set and fewer coverage sets for the refinement.

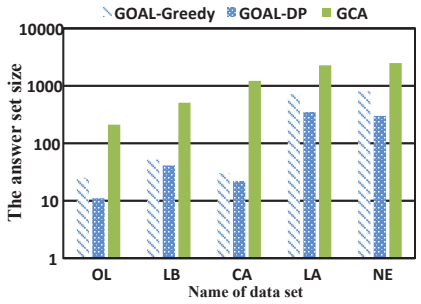


(a) Uniform

(b) Gaussian

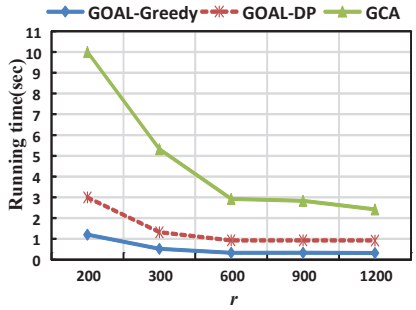


(c) Zipfian ($\alpha = 0.8$)

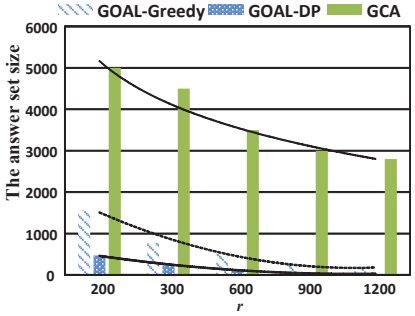


(d) Real data sets

Fig. 22 Effect of data distribution on the size of answer set



(a) Effect of r on efficiency

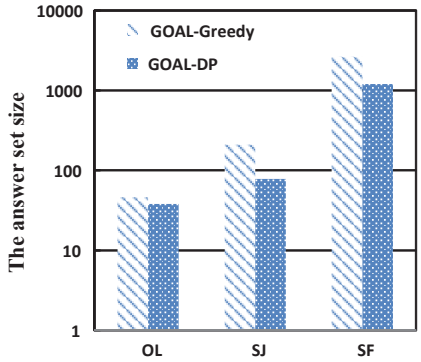


(b) Effect of r on the answer set size

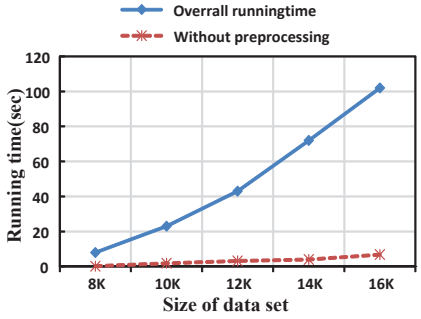
Fig. 23 Effect of r

6.3 Performance on Road Networks

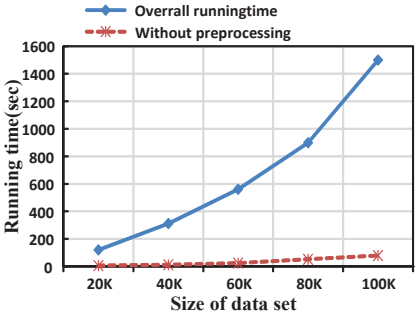
In this section, we demonstrate the effectiveness and efficiency of GOAL-RN, Figure 24(a) demonstrates the answer set size of GOAL-RN algorithm on three road network data sets with different size. We sample from two larger road



(a) Answer set size on real data sets



(b) Efficiency test on SJ



(c) Efficiency test on SF

Fig. 24 Performance on road network data sets

network data sets TG and SF to test the algorithm's efficiency as shown in Fig. 24(b)∼(c). The figure shows that GOAL-RN can solve the GOAL problem in road networks within acceptable running time. We denote the algorithm time excluding the time for running Dijkstra's algorithm to compute the network distance by "Without pre-processing" since this computation can be done offline. We see that our algorithm scales well to larger data sets without the network shortest path distance computation. It takes just a few seconds to process a data set with 100k clients.

7 Conclusion

We studied the group optimal location (GOAL) problem in Euclidean space and road networks. We used a clustering based method to compute an initial answer and to estimate an upper bound of the number of locations needed to solve the problem, which can prune the search space effectively. We proposed three efficient algorithms GOAL-Greedy, GOAL-DP, and GOAL-RN for Euclidean space and road networks that can efficiently (with near-linear time complexity) produce answer sets with small size. Our algorithms outperform

existing algorithms GCA and CREST significantly in Euclidean space with respect to both efficiency and answer set size. They are up to two and one order of magnitude faster than CREST and GCA, respectively.

For future work, it will be interesting to investigate how the problem can be solved in the general metric space, and how parallel frameworks such as Spark can be used to scale the proposed algorithms to even larger data sets. And, since running Dijkstra's algorithm costs a lot of time. On large networks, computing all-pair shortest paths consumes huge time and space. We consider to improve the efficiency of this part in our future work.

Acknowledgement. This work was supported in part by the Key Disciplines of Computer Science and Technology of Shanghai Polytechnic University (No. XXKZD1604), the Research Project of Shanghai Polytechnic University (project number EGD18XQD02), Australian Research Council (ARC) Discovery project (project number DP180103332), the Cultural Relic Protection Science and Technology project of Zhejiang Province, the Key Research and Development Program of Zhejiang Province, the NSFC under Grants (project number 61522208), and the ZJU-Hikvision Joint Project. Huaizhong Lin is the corresponding author.

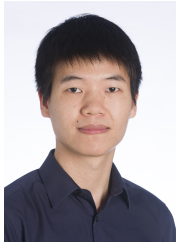
References

1. Helmut Alt, Esther M Arkin, Hervé Brönnimann, Jeff Erickson, Sándor P Fekete, Christian Knauer, Jonathan Lenchner, Joseph SB Mitchell, and Kim Whittlesey. Minimum-cost coverage of point sets by disks. In *Proceedings of the twenty-second annual symposium on Computational geometry*, pages 449–458. ACM, 2006.
2. Habib M Ammari. On the problem of k-coverage in mission-oriented mobile wireless sensor networks. *Computer Networks*, 56(7):1935–1950, 2012.
3. Bhaswar B Bhattacharya. Maximizing voronoi regions of a set of points enclosed in a circle with applications to facility location. *Journal of Mathematical Modelling and Algorithms*, 9(4):375–392, 2010.
4. Sergio Cabello, José Miguel Díaz-Báñez, Stefan Langerman, Carlos Seara, and Inmaculada Ventura. Facility location problems in the plane based on reverse nearest neighbor queries. *European Journal of Operational Research*, 202(1):99–106, 2010.
5. Fangshu Chen, Huaizhong Lin, Yunjun Gao, and Dongming Lu. Capacity constrained maximizing bichromatic reverse nearest neighbor search. *Expert Systems with Applications*, 43:93–108, 2016.
6. Zitong Chen, Yubao Liu, Raymond Chi-Wing Wong, Jiamin Xiong, Ganglin Mai, and Cheng Long. Efficient algorithms for optimal location queries in road networks. In *SIGMOD*, pages 123–134, 2014.
7. Zitong Chen, Yubao Liu, Raymond Chi-Wing Wong, Jiamin Xiong, Ganglin Mai, and Cheng Long. Optimal location queries in road networks. *ACM Transactions on Database Systems*, 40(3):17, 2015.
8. Kenneth L Clarkson and Kasturi Varadarajan. Improved approximation algorithms for geometric set cover. *Discrete & Computational Geometry*, 37(1):43–58, 2007.
9. Yang Du, Donghui Zhang, and Tian Xia. The optimal-location query. In *International Symposium on Spatial and Temporal Databases*, pages 163–180, 2005.
10. Thomas Erlebach, Klaus Jansen, and Eike Seidel. Polynomial-time approximation schemes for geometric graphs. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 671–679. Society for Industrial and Applied Mathematics, 2001.

11. Martin Ester, Hans-Peter Kriegel, Jrg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, pages 226–231. AAAI Press, 1996.
12. Yunjun Gao, Shuyao Qi, Lu Chen, Baihua Zheng, and Xinhan Li. On efficient k-optimal-location-selection query processing in metric spaces. *Information Sciences*, 298:98–117, 2015.
13. Xiaofei He, Deng Cai, Yuanlong Shao, Hujun Bao, and Jiawei Han. Laplacian regularized gaussian mixture model for data clustering. *IEEE Transactions on Knowledge and Data Engineering*, 23(9):1406–1418, 2011.
14. Dorit S Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and vlsi. *Journal of the ACM*, 32(1):130–136, 1985.
15. Jin Huang, Zeyi Wen, Jianzhong Qi, Rui Zhang, Jian Chen, and Zhen He. Top-k most influential locations selection. In *CIKM*, pages 2377–2380, 2011.
16. Mindaugas Kavaliauskas and Rimantas Rudzkis. Multivariate data clustering for the gaussian mixture model. *Informatica, Lith. Acad. Sci.*, 16(1):61–74, 2005.
17. Chun-Hee Lee, Chin-Wan Chung, and Seok-Ju Chun. Effective processing of continuous group-by aggregate queries in sensor networks. *Journal of Systems and Software*, 83(12):2627–2641, 2010.
18. Feifei Li, Bin Yao, and Piyush Kumar. Group enclosing queries. *IEEE Transactions on Knowledge and Data Engineering*, 23(10):1526–1540, 2011.
19. Huaizhong Lin, Fangshu Chen, Yunjun Gao, and Dongming Lu. Optregion: finding optimal region for bichromatic reverse nearest neighbors. In *International Conference on Database Systems for Advanced Applications*, pages 146–160. Springer, 2013.
20. Yubao Liu, Chi Wing Wong, Ke Wang, Zhijie Li, Cheng Chen, and Zhitong Chen. A new approach for maximizing bichromatic reverse nearest neighbor search. *Knowledge Information Systems*, 36(1):23–58, 2013.
21. Mehrdad Mohammadi, Fariborz Jolai, and Hamideh Rostami. An m/m/c queue model for hub covering location problem. *Mathematical and Computer Modelling*, 54(11):2623–2638, 2011.
22. Kyriakos Mouratidis, Dimitris Papadias, and Spiros Papadimitriou. Tree-based partition querying: a methodology for computing medoids in large spatial datasets. *The VLDB Journal*, 17(4):923–945, 2008.
23. Jianzhong Qi, Zhenghua Xu, Yuan Xue, and Zeyi Wen. A branch and bound method for min-dist location selection queries. In *Proceedings of the Twenty-Third Australasian Database Conference-Volume 124*, pages 51–60, 2012.
24. Jianzhong Qi, Rui Zhang, Lars Kulik, Dan Lin, and Yuan Xue. The min-dist location selection query. In *ICDE*, pages 366–377, 2012.
25. Kazuya Sakai, Min-Te Sun, Wei-Shinn Ku, Ten H Lai, and Athanasios V Vasilakos. A framework for the optimal-coverage deployment patterns of wireless sensors. *IEEE Sensors Journal*, 15(12):7273–7283, 2015.
26. David B Shmoys, Éva Tardos, and Karen Aardal. Approximation algorithms for facility location problems. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 265–274, 1997.
27. R. Sibson. SLINK: an optimally efficient algorithm for the single-link cluster method. *Comput. J.*, 16(1):30–34, 1973.
28. Rafael Suárez-Vega, José Luis Gutiérrez-Acuña, and Manuel Rodríguez-Díaz. Locating a supermarket using a locally calibrated huff model. *International Journal of Geographical Information Science*, 29(2):217–233, 2015.
29. Yu Sun, Jianzhong Qi, Rui Zhang, Yueguo Chen, and Xiaoyong Du. Mapreduce based location selection algorithm for utility maximization with capacity constraints. *Computing*, 97(4):403–423, 2015.
30. Yu Sun, Rui Zhang, Andy Yuan Xue, Jianzhong Qi, and Xiaoyong Du. Reverse nearest neighbor heat maps: A tool for influence exploration. In *ICDE*, pages 966–977, 2016.
31. Raymond Chi-Wing Wong, M Tamer Özsu, Philip S Yu, Ada Wai-Chee Fu, and Lian Liu. Efficient method for maximizing bichromatic reverse nearest neighbor. *Proceedings of the VLDB Endowment*, 2(1):1126–1137, 2009.

32. Xiaokui Xiao, Bin Yao, and Feifei Li. Optimal location queries in road network databases. In *ICDE*, pages 804–815, 2011.
33. Chuanfei Xu, Yu Gu, Roger Zimmermann, Shukuan Lin, and Ge Yu. Group location selection queries over uncertain objects. *IEEE Transactions on Knowledge and Data Engineering*, 25(12):2796–2808, 2013.
34. Chuanfei Xu, Yanqiu Wang, Yu Gu, Shukuan Lin, and Ge Yu. Optimal k-constraint coverage queries on spatial objects. In *Proceedings of the Twenty-Third Australasian Database Conference-Volume 124*, pages 41–50, 2012.
35. Donghui Zhang, Yang Du, Tian Xia, and Yufei Tao. Progressive computation of the min-dist optimal-location query. In *VLDB*, pages 643–654, 2006.
36. Zenan Zhou, Wei Wu, Xiaohui Li, Mong Li Lee, and Wynne Hsu. Maxfirst for maxbrknn. In *ICDE*, pages 828–839, 2011.

Fangshu Chen is currently a lecturer in the College of Computer Science and Information Engineering, Shanghai Polytechnic University. She received her Ph.D degree from Zhejiang University in 2017. Her research interests include spatio-temporal databases, location based social networks, and data mining.

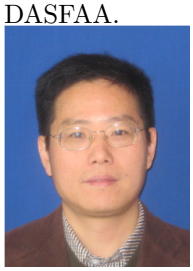


Jianzhong Qi is currently a lecturer in the Department of Computing and Information Systems at the University of Melbourne. He received his Ph.D degree from the University of Melbourne in 2014. His research interests include spatio-temporal databases, location based social networks, information extraction, and text mining. He has published more than 40 papers on several premium/leading journals including TODS, VLDBJ, TKDE, and various prestigious international conferences such as ICDE, VLDB, NIPS, ACL and DASFAA.

Huaizhong Lin is currently an associate professor of computer science in Zhejiang University. He received a Ph.D. degree in Computer Science from Zhejiang University in 2002. His research interests are database

and data mining, spatial database, information retrieval etc., he has published over 40 research papers in journals and conferences.

Yunjun Gao is now a full professor at the College of Computer Science, Zhejiang University. He received the Ph.D. degree in computer science from ZJU in 2008. His primary research areas are Database, Big Data Management and Analytics, and AI Interaction with DB Technology. He has published more than 100 papers on several premium/leading journals including TODS, VLDBJ, TKDE, TOIS, TFS, TITS, and DKE, and various prestigious international conferences such as SIGMOD, VLDB, ICDE, SIGIR, EDBT, and DASFAA.



Dongming Lu is currently a professor of computer science in Zhejiang University, the executive director of the Computer Society of network technical, director and members of the special committee of the China Digital Museum of Zhejiang Province. His research fields mainly focus on the digital media network technology and system, Heritage digital protection and culture passing technology, Wireless and next generation Internet technology and Network information security technology. He has published more than 60 articles and 11 invention patents in the past five years.