On r-Gatherings on the Line

Toshihiro Akagi and Shin-ichi Nakano(✉)

Gunma University, Kiryu 376-8515, Japan
nakano@cs.gunma-u.ac.jp

Abstract. In this paper we study a recently proposed variant of the facility location problem, called the r-gathering problem. Given an integer r, a set C of customers, a set F of facilities, and a connecting cost $co(c, f)$ for each pair of $c \in C$ and $f \in F$, an *r-gathering* of customers C to facilities F is an assignment A of C to *open facilities* $F^{'} \subset F$ such that r or more customers are assigned to each open facility. We give an algorithm to find an *r-gathering* with the minimum cost, where the cost is $\max_{c_i \in C}\{co(c_i, A(c_i))\}$, when all C and F are on the real line.

Keywords: Algorithm · Facility location · Gathering

1 Introduction

The facility location problem and many of its variants are studied [5,6]. In the basic facility location problem we are given (1) a set C of customers, (2) a set F of facilities, (3) an opening cost $op(f)$ for each $f \in F$, and (4) a connecting cost $co(c, f)$ for each pair of $c \in C$ and $f \in F$, then we open a subset $F' \subset F$ of facilities and find an assignment A from C to F' so that a designated cost is minimized.

In this paper we study a recently proposed variant of the problem, called the r-gathering problem [4]. An *r-gathering* of customers C to facilities F is an assignment A of C to *open facilities* $F^{'} \subset F$ such that r or more customers are assigned to each open facility. This means each open facility has enough number of customers. We assume $|C| \geq r$ holds. Then we define the cost of (the *max* version of) a gathering as $\max_{c_i \in C}\{co(c_i, A(c_i))\}$. (We assume $op(f_j) = 0$ for each $f_j \in F$ in this paper.) The min-max version of the *r-gathering problem* finds an r-gathering having the minimum cost. For the min-sum version see the brief survey in [4].

Assume that F is a set of locations for emergency shelters, and $co(c, f)$ is the time needed for a person $c \in C$ to reach a shelter $f \in F$. Then an r-gathering corresponds to an evacuation assignment such that each opened shelter serves r or more people, and the r-gathering problem finds an evacuation plan minimizing the evacuation time span.

Armon [4] gave a simple 3-approximation algorithm for the r-gathering problem and proves that with assumption $P \neq NP$ the problem cannot be approximated within a factor of less than 3 for any $r \geq 3$. In this paper we give an

J. Wang and C. Yap (Eds.): FAW 2015, LNCS 9130, pp. 25–32, 2015.
DOI: 10.1007/978-3-319-19647-3_3

$O((|C| + |F|) \log(|C| + |F|))$ time algorithm to solve the r-gathering problem when all C and F are on the real line.

The remainder of this paper is organized as follows. Section 2 gives an algorithm to solve a decision version of the r-gathering problem. Section 3 contains our main algorithm for the r-gathering problem. Sections 4 and 5 present two more algorithms to solve two similar problems. Finally Sect. 6 is a conclusion.

2 (k,r)-Gathering on the Line

In this section we give a linear time algorithm to solve a decision version of the r-gathering problem [3].

Given customers $C = \{c_1, c_2, \cdots, c_{|C|}\}$ and facilities $F = \{f_1, f_2, \cdots, f_{|F|}\}$ on the real line (we assume they are distinct points and appear in those order from left to right respectively) and two numbers k and r, then problem $P(C, F, j, i)$ finds an assignment A of customers $C_i = \{c_1, c_2, \cdots, c_i\}$ to open facilities $F_j^{'} \subset F_j = \{f_1, f_2, \cdots, f_j\}$ such that (1) r or more customers are assigned to each open facility, (2) $co(c_i, A(c_i)) \leq k$ for each $c_i \in C_i$ and (3) $f_j \in F_j^{'}$. (2) means each customer is assigned to a near facility, and (3) means the rightmost facility is forced to open. We assume that $co(c, f)$ is the distance between $c \in C$ and $f \in F$, and for each $f_j \in F$ interval $[f_j - k, f_j + k]$ contains r or more customers, otherwise we can remove such f_j from F since such f_j never open.

An assignment A of C_i to F_j is called *monotone* if, for any pair $c_{i'}, c_i$ of customers with $i^{'} < i$, $A(c_{i'}) \leq A(c_i)$ holds. In a monotone assignment the interval induced by the assigned customers to a facility never intersects other interval induced by the assigned customers to another facility. We can observe that if $P(C, F, j, i)$ has a solution then $P(C, F, j, i)$ also has a monotone solution. Also we can observe that if $P(C, F, j, i)$ has a solution and $co(c_{i+1}, f_j) \leq k$ then $P(C, F, j, i + 1)$ also has a solution.

If $P(C, F, j, i)$ has a solution for some i then let $s(f_j)$ be the minimum i such that $P(C, F, j, i)$ has a solution. Note that (3) $f_j \in F_j^{'}$ means $c_{s(f_j)}$ is located in interval $[f_j - k, f_j + k]$. We define $P(C, F, j)$ to be the problem to find such $s(f_j)$ and a corresponding assignment. If $P(C, F, j, i)$ has no solution for every i then we say $P(C, F, j)$ has no solution, otherwise we say $P(C, F, j)$ has a solution.

Lemma 1. *For any pair $f_{j'}$ and f_j in F with $j^{'} < j$, $s(f_{j'}) \leq s(f_j)$ holds.*

Proof. Assume otherwise. Then $s(f_{j'}) > s(f_j)$ holds. Modify the assignment corresponding to $s(f_j)$ as follows. Reassign the customers assigned to f_j to $f_{j'}$ then close f_j. The resulting assignment is an r-gathering of $C_{s(f_j)}$ to $F_{j'}$ and now $s(f_{j'}) = s(f_j)$. A contradiction. □

Assume that $P(C, F, j)$ has a solution and $c_1 < f_j - k$. Then the corresponding solution has one or more open facilities except for f_j. Choose the solution of $P(C, F, j)$ having the minimum second rightmost open facility, say $f_{j'}$. We say $f_{j'}$ is the *mate* of f_j and write $mate(f_j) = f_{j'}$. We have the following three cases based on the condition of the mate $f_{j'}$ of f_j.

Case 1: $P(C, F, j^{'})$ has a solution, $f_{j'} + k < f_j - k$, the interval $(f_{j'} + k, f_j - k)$ has no customer and the interval $[f_j - k, f_j + k]$ has r or more customers.

Case 2: $P(C, F, j^{'})$ has a solution, $c_{s(f_{j'})} \geq f_j - k$ and interval $(c_{s(f_{j'})}, f_j + k]$ has r or more customers.

Case 3: $P(C, F, j^{'})$ has a solution, $c_{s(f_{j'})} < f_j - k$ and interval $[f_j - k, f_j + k]$ has r or more customers.

For each f_j by checking the three conditions above for every possible mate $f_{j'}$ one can design $O(|F|^2 + |C|)$ time algorithm based on a dynamic programming approach. However we can omit the most part of the checks by the following lemma.

Lemma 2. *(a) Assume $P(C, F, j)$ has a solution. If $P(C, F, j+1)$ also has a solution then $mate(f_j) \leq mate(f_{j+1})$ holds.*
(b) For $f_j \in F$, let f_{min} be the minimum $f_{j'}$ such that (i)$P(C, F, j^{'})$ has a solution and (ii)$f_{j'} + k \geq f_j - k$, if such f_{min} exists. If $P(C, F, j)$ has no solution with the second rightmost open facility f_{min}, then (b1) any $f_{j''}$ satisfying $f_{min} < f_{j''} < f_j$ is not the mate of f_j, and $P(C, F, j)$ has no solution, and (b2) $f_{min} \leq mate(f_{j+1})$ holds if $mate(f_{j+1})$ exists.

Proof. (a) Assume otherwise. If $mate(f_{j+1}) + k < f_j - k$ holds then $mate(f_{j+1})$ is also the mate of f_j, a contradiction. If $mate(f_{j+1}) + k \geq f_j - k$ holds then by Lemma 1 $mate(f_{j+1})$ is also the mate of f_j, a contradiction. (b1) Immediate from Lemma 1. (b2) Assume otherwise. If $mate(f_{j+1}) + k < f_j - k$ holds then $mate(f_{j+1})$ is also the mate of f_j, a contradiction. If $mate(f_{j+1}) + k \geq f_j - k$ holds then f_{min} is $mate(f_{j+1})$ not $mate(f_j)$, a contradiction. □

Lemma 2 means after searching for the mate of f_j upto some $f_{j'}$ the next search for the mate of f_{j+1} can start at the $f_{j'}$. Based on the lemma above we can design algorithm **find(k, r)-gathering**.

In the preprocessing we compute, for each $f_j \in F$, (1) the index of the first customer in interval $(f_j + k, c_{|C|})$, (2) the index of the first customer in interval $[f_j - k, c_{|C|})$ and (3) the index of the r-th customer in interval $[f_j - k, c_{|C|})$. Also we store the index $s(f_j)$ for each $f_j \in F$. Those needs $O(|C| + |F|)$ time. After the preprocessing the algorithm runs in $O(|F|)$ time since $j' \leq j$ always holds the most inner part to compute $s(f_j)$ executes at most $2|F|$ times.

We have the following lemma.

Lemma 3. *One can solve the (k, r)-gathering problem in $O(|C| + |F|)$ time.*

3 r-Gathering on the Line

In this section we give an $O((|C| + |F|)\log(|C| + |F|))$ time algorithm to solve the r-gathering problem when all C and F are on the real line.

Algorithm 1. find(k,r)-gathering (C, F, k)

```
j = 1
// One open facility Case //
while interval [f_j − k, f_j + k] has both c_1 and c_r do
  set s(f_j) to be the r-th customer c_r
  j = j + 1
end while
// Two or more open facilities Case//
j' = 1
while j ≤ |F| do
  flag = off
  while flag =off and s(f_j) is not defined yet and j' < j do
    if P(C, F, f_j') has a solution and f_j' + k < f_j − k, interval (f_j' + k, f_j − k) has
    no customer then
      set s(f_j) to be the r-th customer in the interval [f_j − k, f_j + k]
    else if P(C, F, f_j') has a solution and f_j' + k ≥ f_j − k then
      flag = on
      if s(f_j') ≥ f_j − k and interval (s(f_j'), f_j + k] has r or more customers then
        set s(f_j) to be the r-th customer in the interval (s(f_j'), f_j + k]
      else if P(C, F, f_j') has a solution, s(f_j') < f_j − k and interval [f_j − k, f_j + k]
      has r or more customers then
        set s(f_j) to be the r-th customer in the interval [f_j − k, f_j + k]
      end if
    end if
    j' = j' + 1
  end while
  j = j + 1
end while
if some f_j with defined s(f_j) has c_|C| within distance k then
  output YES
else
  output NO
end if
```

Our strategy is as follows. First we can observe that the minimum cost k^* of a solution of an r-gathering problem is some $co(c, f)$ with $c \in C$ and $f \in F$. Since the number of distinct $co(c, f)$ is at most $|C||F|$, sorting them needs $O(|C||F| \log(|C||F|))$ time. Then find the smallest k such that the (k, r)-gathering problem has a solution by binary search, using the linear-time algorithm in the preceding section $\log(|C||F|)$ times. Those part needs $O((|C| + |F|) \log |C||F|)$ time. Thus the total running time is $O(|C||F| \log(|C||F|))$.

However by using the sorted matrix searching method [7] (See the good survey in [2, Section 3.3]) we can improve the running time to $O((|C|+|F|) \log(|C|+|F|))$. Similar technique is also used in [8,9] for a fitting problem. Now we explain the detail in our simplified version.

First let M_C be the matrix in which each element is $m_{i,j} = c_i - f_j$. Then $m_{i,j} \geq m_{i,j+1}$ and $m_{i,j} \leq m_{i+1,j}$ always holds, so the elements in the rows and columns are sorted respectively. Similarly let M_F be the matrix in which each element is $m'_{i,j} = f_j - c_i$. The minimum cost k^* of an optimal solution of an r-gathering problem is some positive element in those two matrices. We can find the smallest k in M_C for which the (k,r)-gathering problem has a solution, as follows.

Let n be the smallest integer which is (1) a power of 2 and (2) larger than or equal to $\max\{|C|, |F|\}$. Then we append the largest element $m_{|C|,1}$ to M_C as the elements in the lowest rows and the leftmost columns so that the resulting matrix has exactly n rows and n columns. Note that the elements in the rows and columns are still sorted respectively. Let M_C be the resulting matrix. Our algorithm consists of stages $s = 1, 2, \cdots, \log n$, and maintains a set L_s of submatrices of M_C possibly containing k^*. Hypothetically first we set L_0=$\{M_C\}$. Assume we are now starting stage s.

For each submatrix M in L_{s-1} we partite M into the four submatrices with $n/2^s$ rows and $n/2^s$ columns and put them into L_s.

Let k_{min} be the median of the upper right corner elements of the submatrices in L_s. Then for the $k = k_{min}$ we solve the (k,r)-gathering problem. We have two cases.

If the (k,r)-gathering problem has a solution then we remove from L_s each submatrix with the upper right corner element (the smallest element) greater than k_{min}. Since $k_{min} \geq k^*$ holds each removed submatrix has no chance to contain k^*. Also if L_s has several submatrices with the upper right corner element equal to k_{min} then we remove them except one from L_s. Thus we can remove $|L_s|/2$ submatrices from L_s.

Otherwise if the (k,r)-gathering problem has no solution then we remove from L_s each submatrix with the lower left corner element (the largest element) smaller than k_{min}. Since $k_{min} < k^*$ holds each removed submatrix has no chance to contain k^*. Now we can observe that, for each "chain" of submatrices, which is the sequence of submatrices in L_s with lower-left to upper-right diagonal on the same line, the number of submatrices (1) having the upper right corner element smaller than k_{min} (2) but remaining in L_i is at most one (since the elements on "the common diagonal line" are sorted). Thus, if $|L_s|/2 > D_s$, where $D_s = 2^{s+1}$ is the number of chains plus one, then we can remove at least $|L_s|/2 - D_s$ submatrices from L_s.

Similarly let k_{max} be the median of the lower left corner elements of the submatrices in L_s, and for the $k = k_{max}$ we solve the (k,r)-gathering problem and similarly remove some submatrices from L_s. This ends stage s.

Now after stage $\log n$ each matrix in $L_{\log n}$ has just one element, then we can find the k^* using a binary search with the linear-time decision algorithm.

We can prove that at the end of stage s the number of submatrices in L_s is at most $2D_s$, as follows.

First L_0 has 1 submatrix and $1 \leq 2D_0 = 2 \cdot 2^{0+1}$ submatrix. By induction assume L_{s-1} has $2D_{s-1} = 2 \cdot 2^s$ submatrices.

At stage s we first partite each submatrix in L_{s-1} into four submatrices then put them into L_s. Now the number of submatrices in L_s is $4 \cdot 2D_{s-1} = 4D_s$. We have four cases.

If the (k, r)-gathering problem has a solution for $k = k_{min}$ then we can remove at least a half of the submatrices from L_s, and so the number of the remaining submatrices in L_s is at most $2D_s$, as desired.

If the (k, r)-gathering problem has no solution for $k = k_{max}$ then we can remove at least a half of the submatices from L_s, and so the number of the remaining submatices in L_s is at most $2D_s$, as desired.

Otherwise if $|L_s|/2 \leq D_s$ then the number of the submatices in L_s (even before the removal) is at most $2D_s$, as desired.

Otherwise (1) after the check for $k = k_{min}$ we can remove at least $|L_s|/2 - D_s$ submatices (consisting of too small elements) from L_s, and (2) after the check for $k = k_{max}$ we can remove at least $|L_s|/2 - D_s$ submatices (consisting of too large elements) from L_s, so the number of the remaining submatices in L_s is at most $|L_s| - 2(|L_s|/2 - D_s) = 2D_s$, as desired.

Thus at the end of stage s the number of submatrices in L_s is always at most $2D_s$.

Now we consider the running time. We implicitly treat each submatrix as the index of the upper right element in M_C and the number of lows. Except for the calls of the linear-time decision algorithm for the (k, r)-gathering problem, we need $O(|L_{s-1}|) = O(D_{s-1})$ time for each stage $s = 1, 2, \cdots, \log n$, and $D_0 + D_1 + \cdots + D_{\log n - 1} = 2 + 4 + \cdots + 2^{\log n} < 2 \cdot 2^{\log n} = 2n$ holds, so this part needs $O(n)$ time in total. (Here we use the linear time algorithm to find the median.)

Since each stage calls the linear-time decision algorithm twice this part needs $O(n \log n)$ time in total.

After stage $s = \log n$ each matrix has just one element, then we can find the k^* among the $|L_{\log n}| \leq 2D_{\log n} = 4n$ elements using a binary search with the linear-time decision algorithm at most $\log 4n$ times. This part needs $O(n \log n)$ time.

Then we similarly find the smallest k in M_F for which the (k, r)-gathering problem has a solution. Finally we output the smaller one among the two.

Thus the total running time is $O((|C| + |F|) \log(|C| + |F|))$.

Theorem 1. *One can solve the r-gathering problem in $O((|C| + |F|) \log(|C| + |F|))$ time when all C and F are on the real line.*

4 r-Gather Clustering

In this section we give an algorithm to solve a similar problem by modifying the algorithm in Sect. 3.

Given a set C of n points on the plane an *r-gather-clustering* is a partition of the points into clusters such that each cluster has at least r points. The r-gather-clustering problem [1] finds an r-gather-clustering minimizing the maximum radius among the clusters, where the radius of a cluster is the minimum radius of the disk which can cover the points in the cluster. A polynomial time 2-approximation algorithm for the problem is known [1].

When all C are on the real line, in any solution of any r-gather-clustering problem, we can assume that the center of each disk is at the midpoint of some pair of points, and the radius of an optimal r-gather-clustering is the half of the distance between some pair of points in C.

Given C and two numbers k and r the decision version of the r-gather-clustering problem find an r-gather-clustering with the maximum radius k. We can assume that in any solution of the problem the center of each disk is at $c-k$ for some $c \in C$. Thus, by introducing the set of all such points as F, we can solve the decision version of the r-gather-clustering problem as the (k,r)-gathering problem. Using the algorithm in Sect. 2 we can solve the problem in $O(|C|)$ time.

Now we explain our algorithm to solve the r-gather-clustering problem. First sort C in $O(|C|\log|C|)$ time. Let $c_1, c_2, \cdots, c_{|C|}$ be the resulting non decreasing sequences and let M be the matrix in which each element is $m_{i,j} = (c_i - c_j)/2$. Note that the optimal radius is in M and this time M has $|C|$ rows and columns. Now $m_{i,j} \geq m_{i,j+1}$ and $m_{i,j} \geq m_{i+1,j}$ holds, so the elements in the rows and columns are sorted respectively. Then as in Sect. 3 we can find the optimal radius by the sorted matrix searching method. The algorithm calls the decision algorithm $O(\log|C|)$ times and the decision algorithm runs in $O(|C|)$ time, and in the stages the algorithm needs $O(|C|)$ time in total except for the calls. Finally we needs $O(|C|\log|C|)$ time for the last binary search. Thus the total running time is $O(|C|\log|C|)$.

Theorem 2. *One can solve the r-gather-clustering problem in $O(|C|\log|C|)$ time when all points in C are on the real line.*

5 Outlier

In this section we consider a generalization of the r-gathering problem where at most h customers are allowed to be not assigned.

An *r-gathering with h-outliers* of customers C to facilities F is an assignment A of $C \backslash C'$ to *open facilities* $F' \subset F$ such that r or more customers are assigned to each open facility and $|C'| \leq h$. The *r-gathering with h-outliers problem* finds an r-gathering with h-outliers having the minimum cost.

Given customers $C = \{c_1, c_2, \cdots, c_{|C|}\}$ and facilities $F = \{f_1, f_2, \cdots, f_{|F|}\}$ on the real line and three numbers k and r and h, problem $P(C,F,j,i,h)$ finds an *r-gathering with* h-outliers of $C_i = \{c_1, c_2, \cdots, c_i\} \backslash C'_i$ to $F'_j \subset F_j = \{f_1, f_2, \cdots, f_j\}$ such that (1) r or more customers are assigned to each open facility, (2) $co(c_i, A(c_i)) \leq k$ for each $c_i \in C_i \backslash C'_i$, (3) $f_j \in F'_j$ and (4) $|C'_i| \leq h$. For designated j and h' if $P(C,F,j,i,h')$ has a solution for some i then let $s(f_{j,h'})$ be the minimum i such that $P(C,F,j,i,h')$ has a solution. We define $P(C,F,j,h')$ to be the problem to find such $s(f_{j,h'})$ and a corresponding assignment.

By a dynamic programming approach one can compute $P(C,F,j,h')$ for each $j = 1, 2, \cdots, |F|$ and $h' = 1, 2, \cdots, h$ in $O(|C| + h^2|F|)$ time in total. Then one can decide whether an *r-gathering with* h-outliers problem has a solution with cost k.

Lemma 4. *One can decide whether an r-gathering with h-outliers problem has a solution with cost k in $O(|C| + h^2|F|)$ time.*

The minimum cost k^* of a solution of an r-gathering with h-outliers problem is again some $co(c, f)$ with $c \in C$ and $f \in F$. By the sorted matrix searching method using the $O(|C| + h^2|F|)$ time decision algorithm above one can solve the problem with outliers in $O((|C| + h^2|F|)\log(|C| + |F|))$ time.

Theorem 3. *One can solve the r-gathering with h-outliers problem in $O((|C| + h^2|F|)\log(|C| + |F|))$ time when all C and F are on the real line.*

6 Conclusion

In this paper we have presented an algorithm to solve the r-gathering problem when all C and F are on the real line. The running time of the algorithm is $O((|C| + |F|)\log(|C| + |F|))$. We also presented two more algorithm to solve two similar problems.

Can we design a linear time algorithm for the r-gathering problem when all C and F are on the real line?

References

1. Aggarwal, G., Feder, T., Kenthapadi, K., Khuller, S., Panigrahy, R., Thomas, D., Zhu, A.: Achieving anonymity via clustering, Tranaction on Algorithms, vol. 6, Article No.49, pp. 49:1-49:19 (2010)
2. Agarwal, P., Sharir, M.: Efficient algorithms for geometric optimization. Comput. Surv. **30**, 412–458 (1998)
3. Akagi, T., Nakano, S.: On (k, r)-gatherings on a road. In: Proceedings of Forum on Information Technology, FIT 2013, RA-001 (2013)
4. Armon, A.: On min-max r-gatherings. Theoret. Comput. Sci. **412**, 573–582 (2011)
5. Drezner, Z.: Facility Location: A Survey of Applications and Methods. Springer, New York (1995)
6. Drezner, Z., Hamacher, H.W.: Facility Location: Applications and Theory. Springer, Heidelberg (2004)
7. Frederickson, G., Johnson, D.: Generalized selection and ranking: sorted matrices. SIAM J. Comput. **13**, 14–30 (1984)
8. Fournier, H., Vigneron, A.: Fitting a step function to a point set. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 442–453. Springer, Heidelberg (2008)
9. Liu, J.-Y.: A randomized algorithm for weighted approximation of points by a step function. In: Wu, W., Daescu, O. (eds.) COCOA 2010, Part I. LNCS, vol. 6508, pp. 300–308. Springer, Heidelberg (2010)