A Linear Time Algorithm for the r-Gathering Problem on the Line

Anik Sarker
Department of CSE, BUET
Dhaka, Bangladesh

Wing-kin Sung
School of Computing
NUS, Singapore

M. Sohel Rahman
Department of CSE, BUET
Dhaka, Bangladesh
Facility Location Problem

Input

- A set of customers (C)
- A set of facilities (F)
- Opening cost for each facility
- Connecting cost for each customer to each facility
Facility Location Problem

Output

- Open a subset of facilities
- Find an assignment of each customer to some open facility
- Minimize a designated cost function
r-Gathering

New Constraints

- Each open facility must be assigned at least $r$ customers
- Cost is defined as the maximum over distance between each customer and its assigned facility

$r = 6$
r-Gathering

Practical Application

- Models emergency resident evacuation minimizing the evacuation time span
- For each customer, the time needed to reach each facility is known
- A shelter opens only when at least $r$ people come to it
Previous Results

- **NP-hard** when customers and facilities are on any points in 2D plane.
  - 3-approximation algorithm
- **Cannot be** approximated within a factor less than 3 for \( r \geq 3 \), unless P = NP.
- A related problem is \( r \)-gather-clustering problem
  - Partition points into clusters of size \( \geq r \)

References:
- Akagi & Nakano *FAW’15*
- Nakano; *WALCOM’18*
r-Gathering on the Line

Restriction:

- All customers and facilities are located on the same line.

Customer
Facility

$\bullet$ Customer
$\bigcirc$ Facility

$n = 8$, $m = 2$, $r = 2$

Position on the line
r-Gathering on the Line: Prior Works

- $O((|C| + |F|) \log(|C| + |F|))$ time algorithm
  
  \[ Akagi \& Nakano \]
  
  \[ FAW’15 \]

- $O(|C| + |F| \log^2 r + |F| \log |F|)$ time algorithm
  
  \[ Han \text{ and Nakano} \]
  
  \[ FCS’16 \]

- $O(|C| + r^2 |F|)$
  
  \[ Nakano; \]
  
  \[ WALCOM’18 \]
Our Algorithms

- Basic DP Solution: $O(|F| + |C|^2 \log |F|)$
- Monotone Q:
  - $O(|F| + |C| \log |C| \log |F|)$
  - $O(|F| + |C| \log |F|)$
- Sliding Window:
  - $O(|F| + |C| \log |F|)$
  - $O(|F| + |C|)$

28-Feb-19

WALCOM 2019, IIT Guwahati
Problem Analysis

Lemma 1. There exists an optimal solution in which each facility is assigned to a cluster of customers which are consecutive in C.

Nakano;
WALCOM’18
Problem Analysis

Lemma 2. The optimal facility to assign a cluster of consecutive points $c_a, c_{a+1}, \ldots, c_b$ is the nearest facility from $(c_a + c_b) / 2$.

We want to assign the cluster $a, b, c, d$
Problem Analysis

Lemma 2. The optimal facility to assign a cluster of consecutive points $c_a, c_{a+1}, ..., c_b$ is the nearest facility from $(c_a + c_b) / 2$.

We want to assign the cluster $a, b, c, d$

Corollary 1

Optimal facility for any cluster can be calculated in $O(\log |F|)$.

- Binary search over the sorted list of facilities
- Any $\text{Cost}(a, b) = \text{cost of assigning the cluster } c_a, c_{a+1}, ..., c_b$. query can be answered in $O(\log |F|)$.  

28-Feb-19  
WALCOM 2019, IIT Guwahati  
12
Dynamic Programming Solution

- PrefixCost(i) = minimum cost to assign \{c_1, c_2, c_3, ..., c_i\} such that each facility either is assigned 0 customer or at least r customers.

- Cost(a,b) = minimum cost to assign c_a, c_{a+1}, ..., c_b to a single facility

\[
PrefixCost(i) = \begin{cases} 
0 & \text{if } i = 0 \\
\infty & \text{if } 1 \leq i < r \\
\min_{j=0}^{i-r} \max\{PrefixCost(j), Cost(j+1, i)\} & \text{otherwise}
\end{cases}
\]

- PrefixCost(n) is the solution to our problem
$$PrefixCost(i) = \begin{cases} 
0 & \text{if } i = 0 \\
\infty & \text{if } 1 \leq i < r \\
\min_{j=0}^{i-r} \max\{PrefixCost(j), Cost(j+1, i)\} & \text{otherwise}
\end{cases}$$

Calculation of PrefixCost (6)
Dynamic Programming Solution

**Algorithm 1. min-cost-r-gathering(C,F,r)**

```plaintext
1:  PrefixCost(0) ← 0
2:  for each \(i \in [1, r - 1]\) do
3:      PrefixCost(i) ← ∞
4:  for each \(i \in [r, n]\) do
5:      PrefixCost(i) ← ∞
6:  for each \(j \in [0, i - r]\) do
7:      Let \(F_{\text{mid}}\) be the nearest facility to the midpoint of \([j+1,i]\) segment.
8:      Cost\((j+1,i)\) ← max(\(|C_i - F_{\text{mid}}|, |F_{\text{mid}} - C_{j+1}|\)).
9:      CurrentCost ← max(PrefixCost(j), Cost\((j+1,i)\)).
10:     PrefixCost(i) ← min(PrefixCost(i), CurrentCost).
11: return PrefixCost(n)
```

**Running Time**

\(O(|F| + |C|^2 \log |F|)\)
Improvement 1

Monotone Queue Trick

\[ \text{PrefixCost}(i) = \begin{cases} 
0 & \text{if } i = 0 \\
\infty & \text{if } 1 \leq i < r \\
\min_{j=0}^{i-r} \max \{ \text{PrefixCost}(j), \text{Cost}(j+1, i) \} & \text{otherwise}
\end{cases} \]

For fixed \( i \), sample \( \text{PrefixCost}(j) \) and \( \text{Cost}(j+1,i) \) values for all valid \( j \)

PrefixCost: \[\begin{array}{cccccccc}
0 & 2 & 4 & 3 & 6 & 9 & 7 & 8
\end{array}\]

Cost: \[\begin{array}{cccccccc}
9 & 7 & 6 & 5 & 5 & 3 & 3 & 1
\end{array}\]
Improvement 1

Monotone Queue Trick

For fixed $i$, sample $\text{PrefixCost}(j)$ and $\text{Cost}(j+1,i)$ values for all valid $j$

- $\text{PrefixCost}$ sequence do not follow any pattern.
- $\text{Cost}$ sequence is strictly non-increasing from left to right.

<table>
<thead>
<tr>
<th>PrefixCost</th>
<th>0</th>
<th>2</th>
<th>4</th>
<th>3</th>
<th>6</th>
<th>9</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost</td>
<td>9</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>
Improvement 1

Monotone Queue Trick

It is sufficient to maintain only a strictly increasing sequence in PrefixCost

PrefixCost: 0 2 4 3 6 9 7 8

Cost: 9 7 6 5 5 3 3 1

Turning Point Sequence: 0 1 - 3 4 - 6 7
Improvement 1

Monotone Queue Trick

- We can maintain “Turning Point Sequence” using a stack
- Amortized Complexity: $O(1)$

PrefixCost: [0, 2, 3, 6, 7, 8]

Cost: [9, 7, 5, 5, 3, 1]

Turning Point Sequence: [0, 1, 3, 4, 6, 7]
Improvement 1

Monotone Queue Trick

- Both PrefixCost and Cost sequences are sorted
- We can binary search to find minimum $\max(\text{PrefixCost}(j), \text{Cost}(j+1, i))$

The two candidates

PrefixCost: 0 2 3 6 7 8

Cost: 9 7 5 5 3 1

Turning Point Sequence: 0 1 3 6 7
Improvement 1

Monotone Queue Trick

PrefixCost (i) = \min( \text{Cost}(X_i + 1, i), \text{PrefixCost}(Y_i) )

- Overall Complexity: $O(|F| + |C| \log |C| \log |F|)$
Improvement 1

Algorithm 2. min-cost-$r$-gathering1($C, F, r$)

1: $\text{PrefixCost}(0) \leftarrow 0$
2: for $i = 1$ to $r - 1$ do
3: \hspace{1em} $\text{PrefixCost}(i) \leftarrow \infty$
4: Let $S$ be an empty stack
5: for $i = r$ to $n$ do
6: \hspace{1em} while $S$ is not empty and $\text{PrefixCost}(\text{Top}(S)) \geq \text{PrefixCost}(i - r)$ do
7: \hspace{2.5em} pop the element on top of $S$
8: \hspace{1em} push $(i - r)$ at the top of $S$
9: \hspace{1em} Find maximum index $p$ such that $\text{PrefixCost}(S[p]) \leq \text{Cost}(S[p] + 1, i)$
10: \hspace{1em} $\text{PrefixCost}(i) \leftarrow \text{Cost}(S[p] + 1, i)$
11: \hspace{1em} if $p + 1 \leq \text{Size}(S)$ then
12: \hspace{2.5em} $\text{PrefixCost}(i) \leftarrow \min(\text{PrefixCost}(i), \text{PrefixCost}(S[p + 1]))$
13: return $\text{PrefixCost}(n)$

- Overall Complexity: $O(|F| + |C| \log |C| \log |F|)$
Improvement 2

Sliding Window Technique

We define $\text{Opt}_i$ as:
- $\text{Opt}_i = X_i$ when $\text{Cost}(X_i + 1, i) < \text{PrefixCost}(Y_i)$
- Otherwise $\text{Opt}_i = Y_i$
- Partition Point (Rightmost customer of the previous cluster)
Improvement 2

Sliding Window Technique

Changes in \((i+1)\)th customer than \(i\)th customer:
- Cost \((j+1, i+1) \geq \text{Cost} \((j+1, i)\)
- \((i - r + 1)\)th element appended in the sequences

PrefixCost: 0 2 3 6 7 8 10

Cost: 11 10 8 7 5 2 1

Turning Point Sequence: 0 1 3 4 6 7 8
Improvement 2

Sliding Window Technique

- We can show that, $\text{Opt}_{i+1} \geq \text{Opt}_i$
- Replace binary search part with sliding window technique
- Complexity: $O(|F| + |C| \log |F|)$

PrefixCost:

Cost:

Turning Point Sequence:
Improvement 2

Algorithm 3. min-cost-r-gathering2(C,F,r)

1: PrefixCost(0) ← 0
2: for i = 1 to r - 1 do PrefixCost(i) ← ∞
3: OptIndx ← 1, NextOptIndx ← 2
4: Let S be an empty stack
5: for i = r to n do
6: while S not empty and PrefixCost(Top(S)) ≥ PrefixCost(i - r) do
7: pop the element on top of S
8: push (i - r) at the top of S
9: if OptIndx > Size(S) then
10: OptIndx ← Size(S)
11: NextOptIndx ← OptIndx + 1
12: while true do
13: Opt ← S[OptIndx]
14: F(Opt) ← max(PrefixCost(Opt), Cost(Opt + 1, i)
15: if OptIndx = Size(S) then break
16: NextOpt ← S[NextOptIndx]
17: F(NextOpt) ← max(PrefixCost(NextOpt), Cost(NextOpt + 1, i)
18: if F(Opt) < F(NextOpt) then break
19: OptIndx ← NextOptIndx
20: NextOptIndx ← NextOptIndx + 1
21: PrefixCost(i) ← F(Opt)
22: return PrefixCost(n)

Each Cost(a,b) query runs in $O(\log |F|)$ time

Complexity: $O(|F| + |C| \log |F|)$
Improvement 3

Again Sliding Window Technique

- Current Complexity: $O(|F| + |C| \log |F|)$
- We still need a log $|F|$ factor for answering $\text{Cost}(a,b)$ queries
- Can we make this better?
Improvement 3

Again Sliding Window Technique

- We can notice - in all subsequent Cost(a,b) queries in our algorithm, neither a nor b ever decreases.

- Hence optimal facility in subsequent queries always shifts only to the right.

- We can replace binary search with sliding window technique here once again.
Improvement 3

Again Sliding Window Technique

- We can answer all Cost \((a,b)\) queries in total \(O(|C| + |F|)\) running time

- Overall Time Complexity : \(O(|F| + |C|)\)
Future Directions

- Other variants of r-Gathering
  - r-Gathering on a Star (Already handled here in WALCOM 2019 by Shareef et al.)
    - Can we do better?
  - r-Gathering on trees
- ...
THANK YOU