Facility location problems in the read-only memory with constant work-space

Subhas C. Nandy

Indian Statistical Institute, Kolkata, India

with
Binay K. Bhattacharya, Minati De and Sasanka Roy
Table of Contents

1 Euclidean 1-center
   • Introduction
   • Megiddo’s Algorithm for Constrained-MEC
   • Constrained-MEC using constant space
   • Megiddo’s Algorithm for MEC
   • MEC using constant space

2 Facility location in tree network
   • Centroid of a Tree
   • Weighted 1-center of a Tree

3 Open Problems
Euclidean 1-center

Input
A set of \( n \) points \( P \) in \( \mathbb{R}^2 \)

Objective
- Find \( c^* \in \mathbb{R}^2 \) such that
  \[
  \max_{x \in P} d(c^*, x) = \min_{c \in \mathbb{R}^2} \max_{x \in P} d(c, x)
  \]
**Euclidean 1-center**

**Input**

A set of $n$ points $P$ in $\mathbb{R}^2$

**Objective**

- Find $c^* \in \mathbb{R}^2$ such that
  \[
  \max_{x \in P} d(c^*, x) = \min_{c \in \mathbb{R}^2} \max_{x \in P} d(c, x)
  \]
- Compute the center $c^*$ of the Minimum Enclosing Circle (MEC)
Literature

- Proposed by Sylvester, 1857
- Using FVD takes $O(n \log n)$ time
- $O(n)$ time and $O(n)$ space, Megiddo, 1983
- $O(n^2)$ time $O(1)$ space in Read-only model, by Asano, Mulzer, Rote and Wang, 2011

Open Problem

Any sub-quadratic time algorithm in read-only model using constant space?
Introduction
Megiddo’s Algorithm for Constrained-MEC
Constrained-MEC using constant space
Megiddo’s Algorithm for MEC
MEC using constant space

Literature

Open Problem
Any sub-quadratic time algorithm in read-only model using constant space?

Our Contribution

- In-place algorithm: $O(n)$ time and $O(1)$ extra space
- Read-only Model: $O(n^{1+\epsilon})$ time and $O(\frac{1}{\epsilon} \log n)$ space, where $\epsilon$ is a positive constant less than 1

*Appeared in FSTTCS 2012*
Open Problem

Any sub-quadratic time algorithm in read-only model using constant space?

Here we present

- **Read-only Model:** $O((n + M) \log^4 n)$ time using $O(1)$ space, where $M$ is the time to compute median using $O(1)$ space$^a$.

$^a$Known lower bound of $M$ by T.M. Chan (SODA, 2009): $\Omega(n \log \log S \ n)$ time using $O(S)$ bits
Read-only Model

- **Input**: Read-only
- **Constraints during Execution**:
  - Limited space for storing temporary results
- **After Execution**:
  - Desired result will be reported
Constant-work-space Read-only Model

**Advantages**

- Less prone to failure
- Secured
- Multiple process can access same data simultaneously
- Can handle big data
Example

**Sorting: An $O(n^2)$ time**

- Find the maximum and report
- Find the 2nd maximum and report
- ....
- Find the smallest and report
## Fundamental Algorithms

### Sorting

- $O\left(\frac{n^2}{s} + n \log s\right)$ time and $O(s)$ extra-space (by Frederickson, 1987)

- **Time-space product lower bound:** $\Omega(n^2)$ (by Borodin, 1981)

### Selection

- $O(n \log^3 n)$ time algorithm using $O(1)$ space (by Munro and Paterson, 1978)

- $O(n^{1+\epsilon})$ time and $O\left(\frac{1}{\epsilon}\right)$ extra-space (by Munro and Raman, 1996)

- **Lower bound:** $\Omega(n \log \log S \cdot n)$ time using $O(S)$ bits (by T.M. Chan, 2009)
Constrained MEC

Input
- A set of $n$ points $P$ in $\mathbb{R}^2$
- A vertical line $L$
Constrained MEC

**Input**
- A set of $n$ points $P$ in $\mathbb{R}^2$
- A vertical line $L$

**Objective**
- Compute the Minimum Enclosing Circle whose center $m^*$ lies on $L$
Constrained MEC
Constrained MEC

Arbitrarily pair-up the points
Constrained MEC

Draw the perpendicular bisectors
Constrained MEC

Observe the intersection points on the line $L$
Constrained MEC

Find the median($m$) of the intersection points
Constrained MEC

Find farthest point(s) from $m$
Constrained MEC

Farthest points are both above and below of $m$

$\rightarrow m$ is the result
Constrained MEC

All farthest points are in same side of \( m \)
Constrained MEC

Prune $\frac{1}{4}$th points
Constrained MEC

Repeat same for the rest of the points
Constrained MEC

begin
while $|P| \geq 4$ do

Arbitrarily pair up the points in $P$. Let

$PAIR = \{(P[2i - 1], P[2i]), i = 1, 2, \ldots, \lfloor \frac{|P|}{2} \rfloor\}$ be the set of aforesaid disjoint pairs;

INTERMEDIATE-COMPUTATION-for-Constrained-MEC; (*
It returns a point $m$ on $L$ and a side $D$ *)

(* Pruning step *)
forall the pair of points $(P[2i], P[2i + 1]) \in PAIR$ do

if The bisector line $L_i$ defined by the pair
$(P[2i], P[2i + 1])$ intersect to the opposite side of $D$
from $m$

then

Discard one of $P[2i]$ and $P[2i + 1]$ from $P$ which lies
on the side of $m$ with respect to the bisector line $L_i$;

(* Finally, when $|P| < 4$ *) compute the constrained MEC in
brute force manner and decide on which side of $L$ the center of
unconstrained MEC lies.
Constrained MEC

How to keep track the pruned elements?

- pair and the feasible region
Dominance Relation

Definition
For a pair of points $p, q \in P$, $p$ is said to dominate $q$ with respect to a feasible region $U$, if their perpendicular bisector $b(p, q)$ does not intersect the feasible region $U$, and both $q$ and $U$ lie on the same side of $b(p, q)$.
Dominance Relation

**Property**

\( p \) dominates \( q \) with respect to a feasible region \( U \) if and only if from any point \( x \in U \),
\[ d(p, x) > d(q, x). \]
Dominance Relation

**Lemma**

If $p$ dominates $q$ and $q$ dominates $r$ with respect to a feasible region $U$, then $p$ dominates $r$ with respect to the feasible region $U$.

**Proof.**

$x \in U$, $d(p, x) > d(q, x)$ and $d(q, x) > d(r, x) \Rightarrow d(p, x) > d(r, x)$
Pairing Scheme

Input Array: $P[]$

The feasible region $U = [a, b]$ on $L$
Pairing Scheme

Input Array: $P[]$

The feasible region $U = [a, b]$ on $L$
The feasible region
\[ U = [a, b] \text{ on } L \]
Pairing Scheme

Input Array: $P[]$

The feasible region $U = [a, b]$ on $L$
Pairing Scheme

At the beginning of $k$-th phase

- $U = [a, b]$
- only one element is valid (i.e dominant) from each block of consecutive $2^{k-1}$ elements, namely
  
  $$B^k_i = P[i.2^{k-1} + 1, i.2^{k-1} + 2, \ldots, (i + 1).2^{k-1}],$$
  
  $i = 1, 2, \ldots, \left\lceil \frac{n}{2^{k-1}} \right\rceil$.
- We refer the only valid element of $B^k_i$ as $\text{valid}(B^k_i)$
Pairing Scheme

At the end of $k$-th phase

- $U = [a, b]$ (modified)
- only one element is valid (i.e dominant) from each block of consecutive $2^k$ elements, namely
  $$B_{i}^{k+1} = P[i.2^k + 1, i.2^k + 2, \ldots, (i + 1).2^k], \ i = 1, 2, \ldots, \left\lceil \frac{n}{2^k} \right\rceil.$$
Pairing Scheme

**Formed-Pair in the $k$-th phase**

$\text{valid}(B_{2i-1}^k)$ and $\text{valid}(B_{2i}^k)$ form a pair $Pair_i^k$ for $i = 1, 2, \ldots, \left\lceil \frac{n}{2^{k-1}} \right\rceil$

**Valid Pair**

A pair $Pair_i^k$ is valid pair with respect to $U = [a, b]$ if the corresponding perpendicular bisector intersects $[a, b]$ on $L$. 
In the $k$-th phase

Recognize the only valid element from a block $B^k_i$

Lemma

Let $p, q \in B^k_i$ having perpendicular bisector $b(p, q)$, then any one of the following happens

- *If $b(p, q)$ intersects outside $[a, b]$ on $L$, then one of $p, q$ is not valid($B^k_i$).*
- *If $b(p, q)$ intersects $[a, b]$ on $L$, then none of $p$ and $q$ are valid($B^k_i$).*
In the $k$-th phase

Recognize the valid element from a block $B^k_i$
In the $k$-th phase

Recognize the valid element from a block $B_i^k$
In the $k$-th phase

Recognize the valid element from a block $B_i^k$
In the $k$-th phase

Recognize the valid element from a block $B_i^k$
In the $k$-th phase

Recognize the valid element from a block $B_i^k$
In the $k$-th phase

**Recognize the valid element from a block $B_i^k$**

The $\text{valid}(B_i^k)$ can be identified in $O(|B_i^k|)$ time using $O(1)$ extra-space.

**Lemma**

*We can enumerate all the valid elements of $k$-th phase in $O(n)$ time using $O(1)$ extra-space.*
In the $k$-th phase

In an iteration

- Consider the intersection points on $L$ of the perpendicular bisectors of the valid pairs
- Find the median $m$ among these intersection points
- Decide on which side of $m$ the $m^*$ lies and update $U$
In the $k$-th phase

**In an iteration**

- Consider the intersection points on $L$ of the perpendicular bisectors of the valid pairs
- Find the median $m$ among these intersection points
- Decide on which side of $m$ the $m^*$ lies and update $U$

**After this iteration**

Number of valid pairs decreases by $\frac{1}{4}$
In the $k$-th phase

In an iteration

- Consider the intersection points on $L$ of the perpendicular bisectors of the valid pairs
- Find the median $m$ among these intersection points
- Decide on which side of $m$ the $m^*$ lies and update $U$

After at most $\log n$ iterations, none of $Pair_i^k$ are valid pairs, where $i = 1, 2, \ldots, \left\lceil \frac{n}{2^{k-1}} \right\rceil$
Lemma

The Constrained-MEC can be computed in $O((n + M) \log^2 n)$ time using $O(1)$ extra-space, where $M$ is the time needed to compute the median of $n$ elements in read-only memory when $O(1)$ space is provided.
Megiddo’s Algorithm for MEC
Megiddo’s Algorithm for MEC

Arbitrarily pair up the points
Megiddo’s Algorithm for MEC

Consider their perpendicular bisectors and find the median slope $S_m$
Megiddo’s Algorithm for MEC

Pair up the bisector $L_i, L_j$ such that $\text{slope}(L_i) < S_m < \text{slope}(L_j)$
Megiddo’s Algorithm for MEC

Consider the intersection points of the paired bisectors (tuple)
Megiddo’s Algorithm for MEC

Find their median $x$-coordinate and evoke Decide-on-a-Line
Megiddo’s Algorithm for MEC

Find median y-coordinate and evoke Decide-on-a-Line
Megiddo’s Algorithm for MEC

Prune $\frac{n}{16}$ elements
Megiddo’s Algorithm for MEC

**Algorithm 1: MEC(P)**

**Input:** An array \( P[1, \ldots, n] \) containing a set \( P \) of \( n \) points in \( \mathbb{R}^2 \).

**Output:** The center \( \pi^* \) of the minimum enclosing circle of the points in \( P \).

```
begin
while \(|P| \geq 16\) do
  Arbitrarily pair up the points in \( P \). Let \( PAIR = \{(P[2i-1], P[2i]), i = 1, 2, \ldots, \lfloor |P|/2 \rfloor\} \) be the set of aforesaid disjoint pairs;
  Intermediate-Computation-for-MEC; (It returns a quadrant \( Quad \) defined by two orthogonal lines \( L_H \) & \( L_V \))
  (* Pruning step *)
  forall the pair of points \((P[2i], P[2i+1])\) \(\in PAIR\) do
    if The bisector line \( L_i \) defined by the pair \((P[2i], P[2i+1])\) does not intersect the quadrant \( Quad \) then
      Discard one of \( P[2i] \) and \( P[2i+1] \) from \( P \) which lies on the side of the quadrant \( Quad \) with respect to the bisector line \( L_i \);
  (* Finally, when \(|P| < 16\) *) compute by brute force manner.
end
```
Decide-on-a-Line using constant space

Find the Constrained-MEC center $m^*$
Decide-on-a-Line using constant space

Find all the farthest points $F$
Decide-on-a-Line using constant space

Convex hull of $F$ contains $m^*$
Decide-on-a-Line using constant space

Convex hull of $F$ does not contain $m^*$
Decide-on-a-Line using constant space

Lemma

Decide-on-a-Line(L) can be computed in $O((n + M) \log^2 n)$ time using $O(1)$ extra-space, where $M$ is the time needed to compute the median of $n$ elements given in a read-only memory using $O(1)$ extra-space.
MEC using constant space

How to keep track of the pruned elements?

Intersection of Feasible Region $U$
How to keep track of the pruned elements?

Intersection of Feasible Region $U$
How to keep track of the pruned elements?

Intersection of Feasible Region $U$
Intersection of Feasible Region $U$ of constant complexity

After each iteration we will keep a feasible triangle
Intersection of Feasible Region $U$ of constant complexity

After each iteration we will keep a feasible triangle
Intersection of Feasible Region $U$ of constant complexity

After each iteration we will keep a feasible triangle
Intersection of Feasible Region $U$ of constant complexity

After each iteration we will keep a feasible triangle
Intersection of Feasible Region $U$ of constant complexity

After each iteration we will keep a feasible triangle
Intersection of Feasible Region $U$ of constant complexity

After each iteration we will keep a feasible triangle
Time and Space Complexity

**Single iteration**

Evokes at most four Constrained MEC

**Theorem**

*The Euclidean 1-center of a set of points in \( \mathbb{R}^2 \) given in a read-only array can be found in \( O((n + M) \log^4 n) \) time using \( O(1) \) extra-space, where \( M \) is the time needed to compute the median of \( n \) elements in read-only memory when \( O(1) \) space is provided.*
Similar prune-and-search algorithm

Linear programming in $\mathbb{R}^3$

\[
\begin{align*}
\min_{x_1, x_2, x_3} & \quad d_1 x_1 + d_2 x_2 + d_3 x_3 \\
\text{subject to:} & \quad a'_i x_1 + b'_i x_2 + c'_i x_3 \geq \beta_i, \ i \in I = \{1, 2, \ldots n\}.
\end{align*}
\]

can be computed in $O((n + M) \log^4 n)$ time using $O(1)$ extra-space, where $M$ is the time needed to compute the median of $n$ elements in read-only memory when $O(1)$ space is provided.
Table of Contents

1 Euclidean 1-center
   • Introduction
   • Megiddo’s Algorithm for Constrained-MEC
   • Constrained-MEC using constant space
   • Megiddo’s Algorithm for MEC
   • MEC using constant space

2 Facility location in tree network
   • Centroid of a Tree
   • Weighted 1-center of a Tree

3 Open Problems
Centroid of a Tree

**Centroid of a tree**

A vertex \( v \) of the tree \( T \) such that the size of the largest subtree of \( v \) is minimum among all vertices of \( T \).

**Available Results**

- Can be computed in \( O(n) \) time using \( O(n) \) space.
- In read-only environment, it is easy to compute in \( O(n^2) \) time using \( O(1) \) extra space.

**Our Result**

- A linear time algorithm using \( O(1) \) extra space in read-only environment.
Centroid of a Tree

Data Structure

The tree $T$ is stored as a DCEL in read-only memory, such that for each vertex $v$ the following operations can be performed in $O(1)$ time.

- Parent($v$)
- FirstChild($v$)
- NextChild($u, v$) - Child of $u$ which is next to $v$

We use $T_m(t)$: the subtree of the node $t$ rooted at one of its neighbors $m$. 
Centroid of a Tree

Algorithm

1. Start from an arbitrary vertex \( t \in V \)
2. Compute a neighbor \( m \) of \( t \) having maximum number of nodes in the subtree \( T_m(t) \).
3. If \( |T_m(t)| \leq \lceil \frac{n}{2} \rceil \), then report \( t \) as the centroid. Otherwise, the centroid lies in \( T_m(t) \).
Centroid of a Tree

We now concentrate of Step 2 of the algorithm.

**Invariant**

Maintain three variables $t$, $t'$ and Size satisfying

- Initially $t = \text{root}$, $t' = \emptyset$ and $\text{Size} = 0$
- If $t' \neq \emptyset$ then $t$ and $t'$ are adjacent vertices and $\text{Size} = |T_{t'}(t)|$.

**Note:** During the search $t'$ is the parent of $t$
Centroid of a Tree

Compute $m$ such that $|T_m(t)| = \max_{v \in N(t)} |T_v(t)|$ as follows:

- Initially, $t'$ is the predecessor of $t$, $\text{Size} = |T_{t'}(t)|$, and $\text{SIZE} = \text{MaxSize} = |T_t(t')|$ computed in the earlier step.
- Use two pointers $\phi_1$ and $\phi_2$ to count the size of two subtrees of $T$ in a parallel manner.
- Use the procedures $\text{Parent}(u)$, $\text{FirstChild}(u)$ and $\text{NextChild}(u, v)$ to compute the size of the subtrees pointed by $\phi_1$ and $\phi_2$. 
Centroid of a Tree

- When the counting of a pointer in one subtree is finished, it starts with an unprocessed subtree immediately.
- Each time, when the counting of a subtree by one of the pointers is finished, the count is added in a counter $TOTAL$.
- When the task of one of the pointers is finished, and it does not find any unprocessed subtree of $T$, then the process stops without completing the task of the other pointer in which it was working.
- Its count is available by computing $SIZE - TOTAL$.

During this process
the node $m \in N(t)$ having maximum $|T_m(t)|$, and $MaxSize = |T_m(t)|$ is maintained.
Centroid of a Tree

- If \( \text{MaxSize} \leq \left\lfloor \frac{n}{2} \right\rfloor \), report \( m \) as centroid.
- Else recurse with \( t' = t \), \( t = m \), \( \text{Size} = n - \text{MaxSize} \) and \( \text{SIZE} = \text{MaxSize} \).

The recursion continues in the subtrees rooted at \( m \). One of its neighbors is \( t \), whose elements are already visited in the previous iteration, and the count is \( T_m(t) \).
Centroid of a Tree

Result

- In each iteration, at most half of the elements of $T$ are visited by each of $\phi_1$ and $\phi_2$.
- In an iteration, either answer is reported; or the process starts with the unfinished subtree, which is of largest size among the subtrees of $t$.
- The total time complexity of an iteration is $2 \times Size$.
- Since in each iteration, a unfinished subtree is processed, the charge goes to the subtrees which are completely processed.
- Thus, the total time complexity becomes $O(n)$. 
Given a tree $T$ having each vertex attached with positive weights, a point $\pi$ on an edge $e^*$ of $T$ is said to be the weighted 1-center if

$$\Delta_w(\pi) = \min_{p \in T} \Delta_w(p)$$

where $\Delta_w(p) = \max_{v \in V} w(v) \times d(p, v)$, and $d(p, v)$ is the distance of the vertex $v$ from the point $p$ along the edges of $T$. 
Weighted 1-center of a Tree

Overview of the Algorithm

- Identify an edge $e^*$ on which the 1-center lies
- Next, compute the point $\pi$ (1-center) on $e^*$.

An important result

- If $u$ is a fixed vertex in $T$, and
- $v'$ be a vertex in $T$ satisfying $w(v')d(v', u) = \max_{v \in V} w(v)d(v, u)$ that lies in $T_t(u)$, then
- the 1-center of $T$ lies in $T_t(u^+)$.

Here $T_t(u^+)$ is the subtree of $T$ containing $T_t(u) \cup \{u\} \cup \{(u, t)\}$.
Finding $e^*$

1. Initialize $T' = T$.
2. Repeat the following steps until $T'$ is an edge.
   - Compute the centroid in $T'$. Let it be $u$.
   - Identify the vertex $v'$ traversing all the subtrees of $u$ in the tree $T$ completely.
   - Suppose $v' \in T_t(u)$. Set $T' = T'_t(u^+)$. 
Finding $e^*$

- Since, in each iteration, half of the vertices are pruned, to set $T'$, the number of iteration is $O(\log n)$.
- In order to compute $v'$, the entire tree $T$ is scanned. Thus, the time complexity of each iteration is $O(n)$.

**Result**

The overall time complexity of identifying $e^*$ is $O(n \log n)$. The space complexity is $O(1)$.

**An important property**

At most two internal vertices of $T$ become the leaves of $T'$. This enables us to encode $T'$ using only four variables.
Computing the 1-center on the edge \( e^* = [u^*, v^*] \)

**Query(\( m \))** — for a query point \( m \in e^* \)

Test whether \( \pi = m \) or \( \pi \in [u^*, m] \) or \( \pi \in [m, v^*] \).

**Process**

- For a point \( x \in [a, b] \), define
  \[
  f_1(x, v) = w(v) \times (d(u^*, v) + d(x, u^*)) \quad \text{for all } v \in T_u^*(v^*),
  \]
  \[
  f_2(x, v) = w(v) \times (d(v^*, v) + d(x, v^*)) \quad \text{for all } v \in T_v^*(u^*).
  \]

- Find \( f_1(k_1, m) = \max_{v \in T_u^*(v^*)} f_1(m, v) \), and \( f_2(k_2, m) = \max_{v \in T_v^*(u^*)} f_1(m, v) \).

- If \( f_1(k_1, m) = f_2(k_2, m) \), then \( \pi = m \)
  Else if \( f_1(k_1, m) > f_2(k_2, m) \) then \( \pi \in [a, m] \), else \( \pi \in [m, b] \).

**Time complexity of Query(\( m \)): \( O(n) \)**
Computing $m$

**Dominance**

For a pair of vertices $p, q \in T_{u^*}(v^*)$, $p$ dominates $q$ if $f_1(p, x)$ and $f_1(q, x)$ do not intersect in $[a, b]$, and $f_1(q, x) < f_1(p, x)$. 
Computing $m$

**Process**

- Pair up vertices of $V_1 = T_{u^*}(v^*)$.
- If one of them dominates the other, then prune the dominated one.
  Otherwise compute the point of intersection $x \in [a, b]$ of $f_1(p, x)$ and $f_1(q, x)$.
- Similarly, pair up vertices of $V_2 = T_{v^*}(u^*)$, and compute the point of intersection of $f_1(p, x)$ and $f_1(q, x)$ for the undominated pairs $(p, q) \in V_2$.
- Compute the median $m$ of these intersection points in read-only memory.
Complexity

Result

Time complexity: $O((n + M) \log^2 n)$, and
Space: $O(1)$, where $M$ is the time needed for computing the median of $n$ elements in read-only environment using $O(1)$ space.
Table of Contents

1 Euclidean 1-center
   • Introduction
   • Megiddo’s Algorithm for Constrained-MEC
   • Constrained-MEC using constant space
   • Megiddo’s Algorithm for MEC
   • MEC using constant space

2 Facility location in tree network
   • Centroid of a Tree
   • Weighted 1-center of a Tree

3 Open Problems
Open Problem

Farthest Pair of Points

Given a set $P$ of $n$ points in $\mathbb{R}^2$, find the farthest pair of points in sub-quadratic time in the constant-work-space read-only model.

Maximum Empty Circle

Given a set $P$ of $n$ points in $\mathbb{R}^2$, find the maximum empty circle (whose center is inside the convex hull of $P$) in sub-quadratic time in the constant-work-space read-only model.
Thank You!