Geometric Spanners

Anil Maheshwari

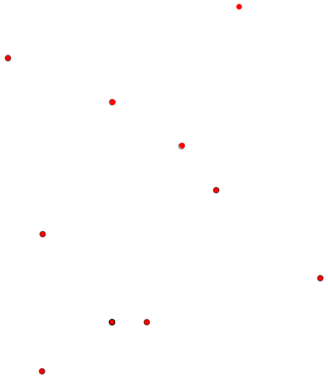
School of Computer Science
Carleton University
Canada

July 19, 2008

Outline

Designing a Network

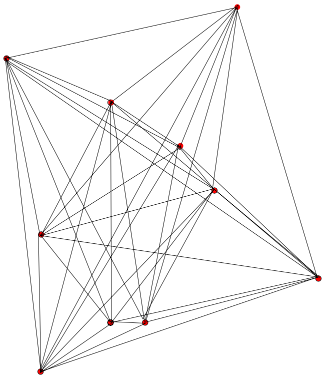
How do we connect these cities?



Connect everybody!

Designing a Network

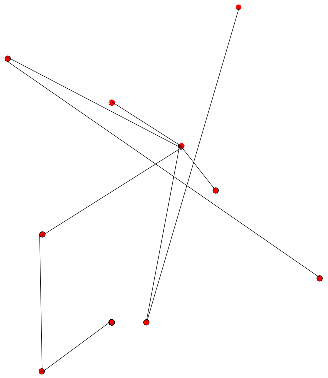
How do we connect these cities?



Too many edges!

Designing a Network

How do we connect these cities?



Too few edges!

Network

Network (Graph)

Edge-weighted undirected graph $G(V, E)$
V= Set of Vertices
E=Set of Edges

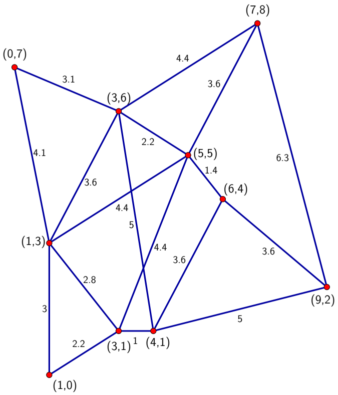
- $\forall e = (u, v) \in E$,

$$wt(e) = \mathbf{d}(u, v).$$

Geometric Network

- V= Set of points in the plane.
- $\mathbf{d}$ = Euclidean distance, i.e., $\forall e = (u, v) \in E$, $wt(e) = |uv|$.

Network



Network (Graph)

Edge-weighted undirected graph $G(V, E)$
V= Set of Vertices
E=Set of Edges

- $\forall e = (u, v) \in E$,

$$wt(e) = \mathbf{d}(u, v).$$

Geometric Network

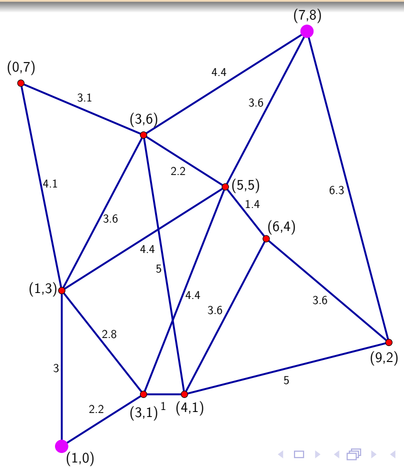
- V= Set of points in the plane.
- $\mathbf{d}$ = Euclidean distance, i.e., $\forall e = (u, v) \in E$, $wt(e) = |uv|$.

t-spanner

Network $G(V, E)$ is a t-spanner ($t \geq 1$) if

- $\forall u, v \in V$,

$$\mathbf{d}_G(u, v) \leq t \times \mathbf{d}(u, v). \qquad (t\text{-path})$$



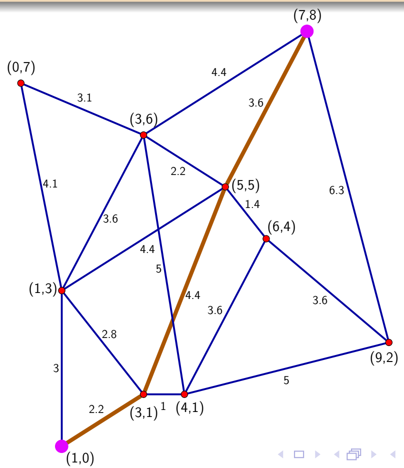
t-spanner

t-spanner

Network $G(V,E)$ is a t-spanner ($t \geq 1$) if

- $\forall u, v \in V$,

$$\mathbf{d}_G(u,v) \leq t \times \mathbf{d}(u,v). \qquad (t\text{-path})$$



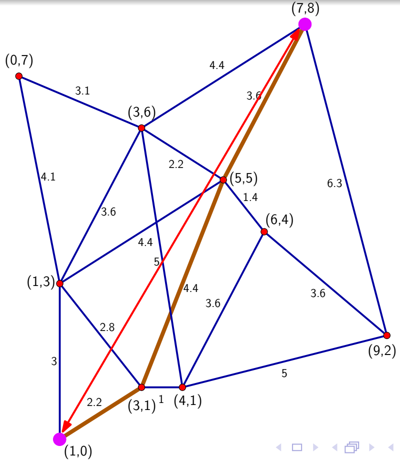
t-spanner

t-spanner

Network $G(V, E)$ is a t-spanner ($t \geq 1$) if

- $\forall u, v \in V$,

$$\mathbf{d}_G(u, v) \leq t \times \mathbf{d}(u, v). \qquad (t\text{-path})$$

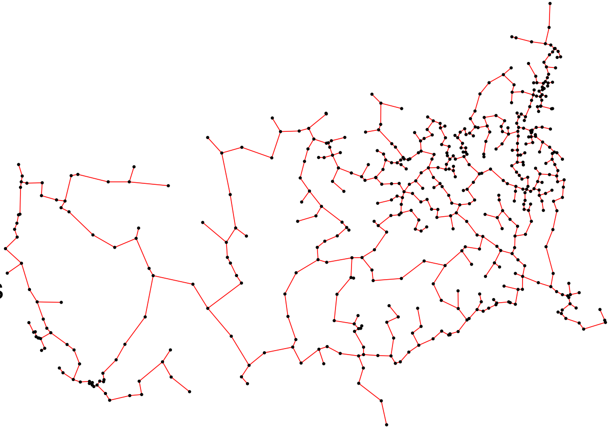


Example

10-spanner

for

532 US-cities

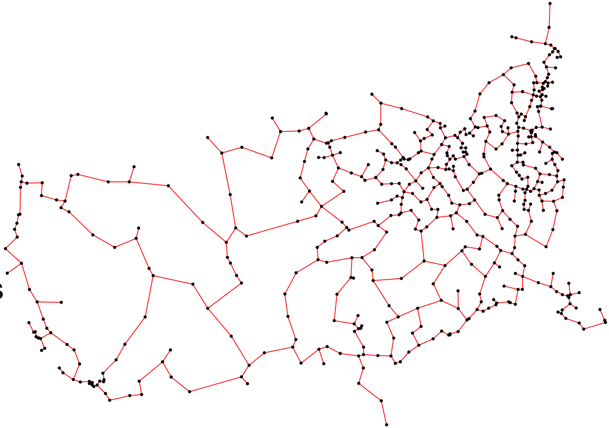


Example

5-spanner

for

532 US-cities

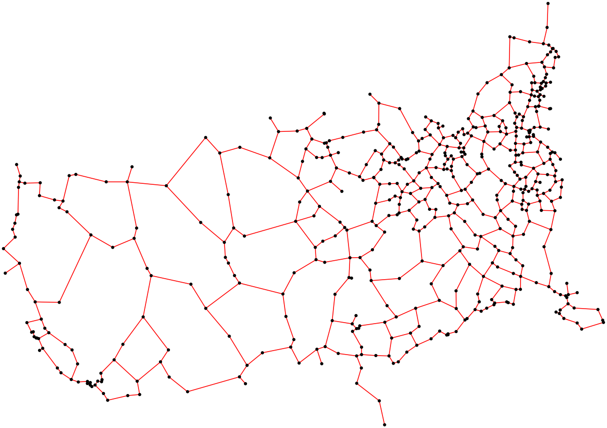


Example

3-spanner

for

532 US-cities

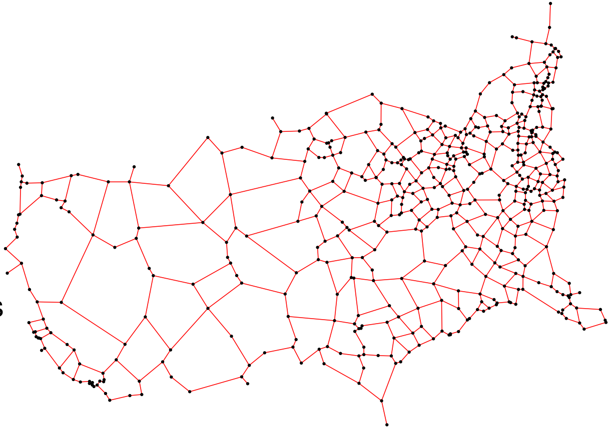


Example

2-spanner

for

532 US-cities

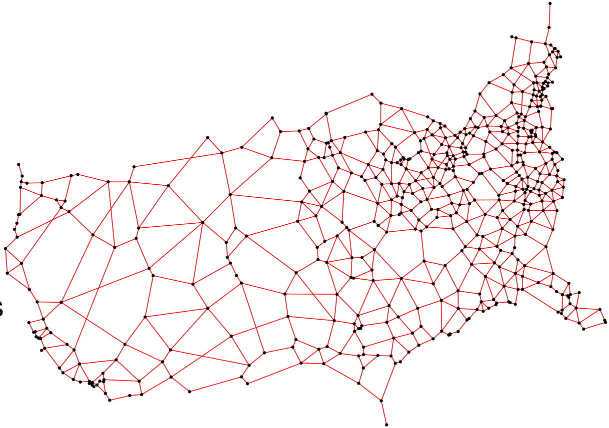


Example

1.5-spanner

for

532 US-cities



Main Question

Problem:

Given a set V and $t > 1$, construct a sparse t-spanner of V.

Sparseness:

- Number of edges (size)
- Weight (compared with weight of MST)
- Maximum degree
- Diameter

Main Question

Problem:

Given a set V and $t > 1$, construct a sparse t-spanner of V.

Sparseness:

- Number of edges (size)
- Weight (compared with weight of MST)
- Maximum degree
- Diameter

Spanner Algorithms

Spanner Algorithms

-	Size	Weight / $wt(MST)$	Degree	Time	
Greedy	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(n^3 \log n)$	Metric
Apx. greedy	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(n \log n)$	Geometric
Θ-graph	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n \log n)$	Geometric
WSPD	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$	$\mathcal{O}(n \log n)$	Geometric
Sink-spanner	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(n \log n)$	Geometric
Skip-list spanner	$\mathcal{O}(n)^*$	$\mathcal{O}(n)^*$	$\mathcal{O}(n)$	$\mathcal{O}(n \log n)^*$	Geometric

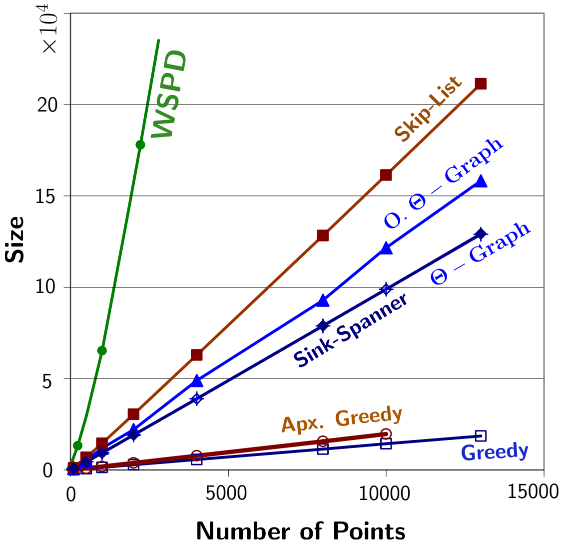
Constructing sparse t-spanners:

- Greedy (Bern (1989) and Althöfer et al. (1993)).
- Θ-graph (Clarkson (1987) and Keil (1988)).
- Well-Separated Pair Decomp. (Callahan and Kosaraju (1992)).

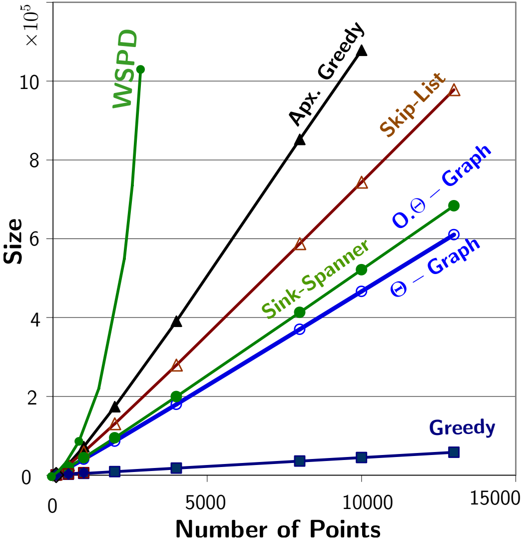
Spanner Algorithms

In Practice

2-spanner



1.1-spanner



Size
$\times 10^5$
0
2
4
6
8
10
0
5000
10000
15000
Number of Points
WSPD
Apx. Greedy
Skip-List
O.Θ – Graph
Sink-Spanner
Θ – Graph
Greedy

Greedy Algorithm

- Works for any metric, but this talk is only about points in the plane and Euclidean distances.
- Sort all pairs of points in increasing order of distances.
- Consider them one by one in that order.
- Let (u, v) be the current pair under consideration.
- Add e to the spanner if the distance between the endpoints u and v is greater than $t \cdot \mathrm{d}(u, v)$.

Greedy Algorithm

- Works for any metric, but this talk is only about points in the plane and Euclidean distances.
- Sort all pairs of points in increasing order of distances.
- Consider them one by one in that order.
- Let (u, v) be the current pair under consideration.
- Add e to the spanner if the distance between the endpoints u and v is greater than $t \cdot \mathrm{d}(u, v)$.

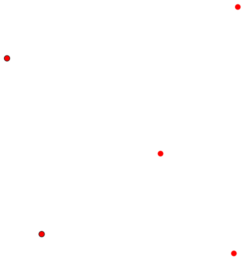
Greedy Algorithm

- Works for any metric, but this talk is only about points in the plane and Euclidean distances.
- Sort all pairs of points in increasing order of distances.
- Consider them one by one in that order.
- Let (u, v) be the current pair under consideration.
- Add e to the spanner if the distance between the endpoints u and v is greater than $t \cdot \mathrm{d}(u, v)$.

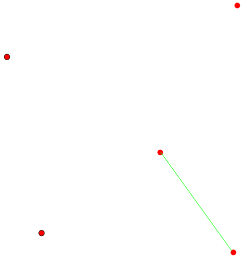
Greedy Algorithm

- Works for any metric, but this talk is only about points in the plane and Euclidean distances.
- Sort all pairs of points in increasing order of distances.
- Consider them one by one in that order.
- Let (u, v) be the current pair under consideration.
- Add e to the spanner if the distance between the endpoints u and v is greater than $t \cdot \mathbf{d}(u, v)$.

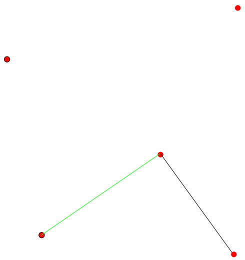
An Example ($t = 2$)



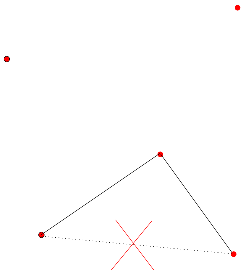
An Example ($t = 2$)



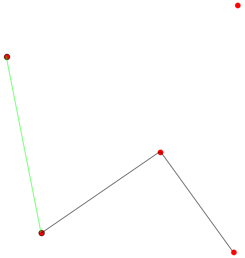
An Example ($t = 2$)



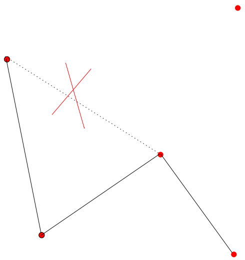
An Example ($t = 2$)



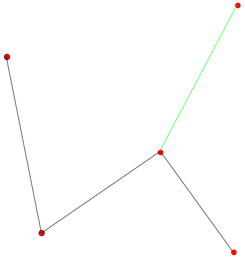
An Example ($t = 2$)



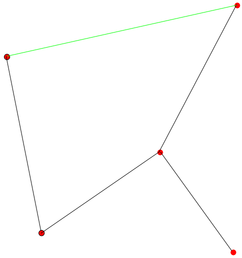
An Example ($t = 2$)



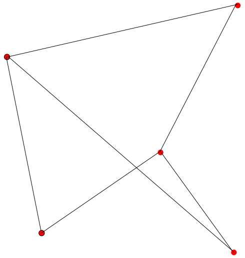
An Example ($t = 2$)



An Example ($t = 2$)



An Example ($t = 2$)



Original Greedy Algorithm

$E := \emptyset; G := (V, E);$

foreach $(u, v) \in V^2$ *(in sorted order)* **do**

  if $\mathbf{d}_G(u, v) > t \cdot \mathbf{d}(u, v)$ **then**

    $E := E \cup \{(u, v)\};$

  end

end

return $G = (V, E);$

Greedy Algorithm

Original Greedy Algorithm

$E := \emptyset; G := (V, E);$

foreach $(u, v) \in V^2$ *(in sorted order)* **do**

  if $\mathrm{d}_G(u, v) > t \cdot \mathrm{d}(u, v)$ **then**

    $E := E \cup \{(u, v)\};$

  end

end

return $G = (V, E);$

Running time:$\mathcal{O}(n^2 \times SSSP) = \mathcal{O}(mn^2 + n^3 \log n)$.
(Dijkstra's SSSP requires $\mathcal{O}(m + n \log n)$ time)

Properties of Greedy Spanner

- Linear size, i.e., number of edges is $\mathcal{O}(n)$.
- Constant degree.
- Weight is $\mathcal{O}(\log n.\text{wt(MST)})$.
- But running time is VERY HIGH, i.e., $\mathcal{O}(n^2 \times SSSP)$.

Challenge

Can we somehow avoid $\mathcal{O}(n^2)$ shortest path computations?

Properties of Greedy Spanner

- Linear size, i.e., number of edges is $\mathcal{O}(n)$.
- Constant degree.
- Weight is $\mathcal{O}(\log n.\text{wt}(\text{MST}))$.
- But running time is VERY HIGH, i.e., $\mathcal{O}(n^2 \times SSSP)$.

Challenge

Can we somehow avoid $\mathcal{O}(n^2)$ shortest path computations?

FG-Greedy Algorithm

FG-Greedy Algorithm

$E := \emptyset$; $G := (V, E)$;

foreach $(u, v) \in V^2$ *with* $u \neq v$ **do** $weight(u, v) := \infty$;

foreach $(u, v) \in V^2$ *(in sorted order)* **do**

  if $weight(u, v) > t \cdot \mathbf{d}(u, v)$ **then**

    perform an SSSP in G with source u;

    foreach $w \in V$ **do** update $weight(u, w)$ and $weight(w, u)$;

    if $weight(u, v) > t \cdot \mathbf{d}(u, v)$ **then** $E := E \cup \{(u, v)\}$;

  end

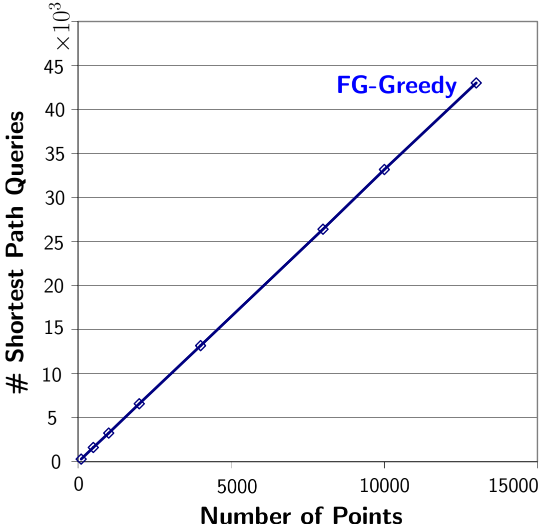
end

return $G = (V, E)$;

FG-Greedy Algorithm

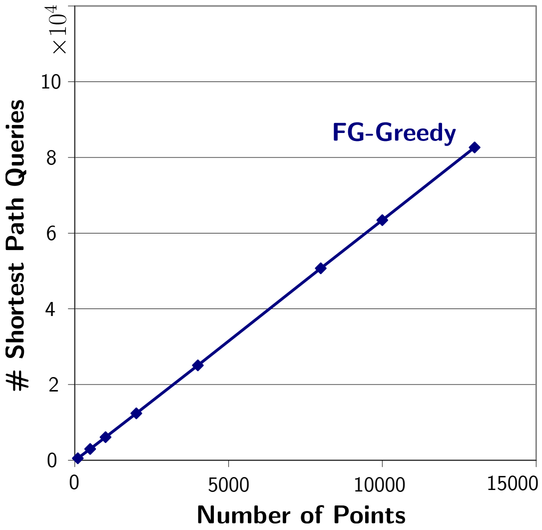
In Practice

$t = 2$



FG-Greedy Algorithm
In Practice

$t = 1.1$



FG-Greedy Algorithm

Counterexample

Farshi-Gudmundsson Conjecture:

The FG-greedy algorithm performs $\mathcal{O}(n)$ SSSP.

FALSE!

FG-Greedy Algorithm

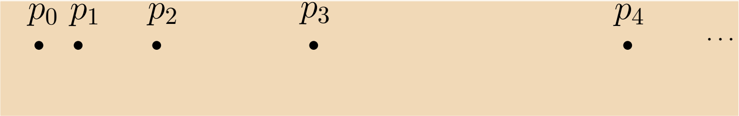
Counterexample

Farshi-Gudmundsson Conjecture:

The FG-greedy algorithm performs $\mathcal{O}(n)$ SSSP.

FALSE!

$p_i = 2^i$



Improved Algorithm(2008)

Main ideas:

- Splitting all pairs of points into $\mathcal{O}(n)$ buckets and processing a bucket at a time.
- Keeping the weight matrix up-to-date for all pairs in the processing bucket.
  - Update the weight matrix at the beginning of processing a bucket (by running bounded Dijkstra's algorithm).
  - After adding an edge, update the weight matrix (partially).
  - Instead of running Dijkstra's algorithm from scratch, we fix the previous run.

Improved Algorithm(2008)

Main ideas:

- Splitting all pairs of points into $\mathcal{O}(n)$ buckets and processing a bucket at a time.
- Keeping the weight matrix up-to-date for all pairs in the processing bucket.
  - Update the weight matrix at the beginning of processing a bucket (by running bounded Dijkstra's algorithm).
  - After adding an edge, update the weight matrix (partially).
  - Instead of running Dijkstra's algorithm from scratch, we fix the previous run.

Improved Algorithm(2008)

Main ideas:

- Splitting all pairs of points into $\mathcal{O}(n)$ buckets and processing a bucket at a time.
- Keeping the weight matrix up-to-date for all pairs in the processing bucket.
    - Update the weight matrix at the beginning of processing a bucket (by running bounded Dijkstra's algorithm).
    - After adding an edge, update the weight matrix (partially).
    - Instead of running Dijkstra's algorithm from scratch, we fix the previous run.

Improved Algorithm(2008)

Main ideas:

- Splitting all pairs of points into $\mathcal{O}(n)$ buckets and processing a bucket at a time.
- Keeping the weight matrix up-to-date for all pairs in the processing bucket.
  - Update the weight matrix at the beginning of processing a bucket (by running bounded Dijkstra's algorithm).
  - After adding an edge, update the weight matrix (partially).
  - Instead of running Dijkstra's algorithm from scratch, we fix the previous run.

Improved Algorithm(2008)

Main ideas:

- Splitting all pairs of points into $\mathcal{O}(n)$ buckets and processing a bucket at a time.
- Keeping the weight matrix up-to-date for all pairs in the processing bucket.
    - Update the weight matrix at the beginning of processing a bucket (by running bounded Dijkstra's algorithm).
    - After adding an edge, update the weight matrix (partially).
    - Instead of running Dijkstra's algorithm from scratch, we fix the previous run.

Improved Algorithm

New-Greedy Algorithm

$E := \emptyset$; $G := (V, E)$;

foreach $(u, v) \in V^2$ *with* $u \neq v$ **do** $weight(u, v) := \infty$;

Split all pair of points to buckets $\{E_i\}$;

foreach i **do**

    $\forall$ vertices, run bounded-Dijkstra and update $weight$;

    foreach $(u, v) \in E_i$ *(in sorted order)* **do**

        if $weight(u, v) > t \cdot \mathbf{d}(u, v)$ **then**

            $E := E \cup \{(u, v)\}$;

            Partially update $weight$;

        end

    end

end

Improved Algorithm

Main ideas:

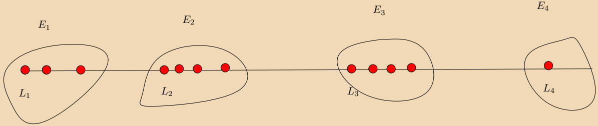
- Splitting all pairs of points into $\mathcal{O}(n)$ buckets and processing a bucket at a time.
- Keeping the weight matrix up-to-date for all pairs in processing bucket.
  - Update the weight matrix at the beginning of processing a bucket (by running bounded Dijkstra's algorithm).
  - After adding an edge, update the weight matrix (partially).
  - Instead of running Dijkstra's algorithm from scratch, we fix the previous run.

Buckets:

- L_1: distance between the closest pair.
- E_1: all pairs with distance in $[L_1, 2L_1)$.

$\vdots$

- L_i: distance between the closest pair between remaining pairs.
- E_i: all pairs with distance in $[L_i, 2L_i)$.



Theorem: (Har-Peled)

For any set of ... points from a metric space, the number

Improved Algorithm

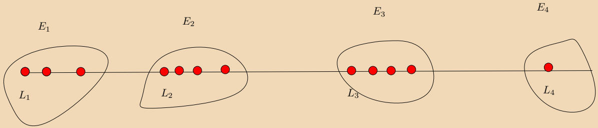
Bucketing

Buckets:

- L_1: distance between the closest pair.
- E_1: all pairs with distance in $[L_1, 2L_1)$.

⋮

- L_i: distance between the closest pair between remaining pairs.
- E_i: all pairs with distance in $[L_i, 2L_i)$.



Theorem: (Har-Peled)

For any set of ... points from a metric space, the number

Buckets:

- L_1: distance between the closest pair.
- E_1: all pairs with distance in $[L_1, 2L_1)$.

$$\vdots$$

- L_i: distance between the closest pair between remaining pairs.
- E_i: all pairs with distance in $[L_i, 2L_i)$.

Theorem: (Har-Peled)

For any set of n points from a metric space, the number of buckets is $\mathcal{O}(n)$.

Algorithm

Main ideas:

- Splitting all pairs of points into $\mathcal{O}(n)$ buckets and processing a bucket at a time.
- Keeping the weight matrix up-to-date for all pairs in processing bucket.
  - Update the weight matrix at the beginning of processing a bucket (by running bounded Dijkstra's algorithm).
  - After adding an edge, update the weight matrix (partially).
  - Instead of running Dijkstra's algorithm from scratch, we fix the previous run.

Main ideas:

- Splitting all pairs of points into $\mathcal{O}(n)$ buckets and processing a bucket at a time.
- Keeping the weight matrix up-to-date for all pairs in processing bucket.
  - Update the weight matrix at the beginning of processing a bucket (by running bounded Dijkstra's algorithm).
  - After adding an edge, update the weight matrix (partially).
  - Instead of running Dijkstra's algorithm from scratch, we fix the previous run.

Improved Algorithm

Bounded Dijkstra's algorithm

- Assume we are about to process bucket E_i.
- We know that $\forall(u,v) \in E_i, \mathbf{d}(u,v) \in [L_i, 2L_i)$.

Claim:

Running Dijkstra's algorithm with bound $2tL_i$ is sufficient.

Note:

To prove this, we need to show that, at the moment the algorithm process $(u,v) \in E_i$, we have $weight(u,v) > t \cdot d_G(u,v)$ if and only if $d_G(u,v) > t \cdot d(u,v)$.

- Assume we are about to process bucket E_i.
- We know that $\forall(u,v) \in E_i, \mathbf{d}(u,v) \in [L_i, 2L_i)$.

Claim:

Running Dijkstra's algorithm with bound $2tL_i$ is sufficient.

The Dijkstra's algorithm stops as soon as the minimum key in the priority queue is larger than $2tL_i$.

Note:

To prove this, we need to show that, at the moment the algorithm process $(u,v) \in E_i$, we have $weight(u,v) > t \cdot d_G(u,v)$ if and only if $d_G(u,v) > t \cdot d(u,v)$.

- Assume we are about to process bucket E_i.
- We know that $\forall(u,v) \in E_i, \mathbf{d}(u,v) \in [L_i, 2L_i)$.

Claim:

Running Dijkstra's algorithm with bound $2tL_i$ is sufficient.

Note:

To prove this, we need to show that, at the moment the algorithm process $(u,v) \in E_i$, we have $weight(u,v) > t \cdot d_G(u,v)$ if and only if $d_G(u,v) > t \cdot d(u,v)$.

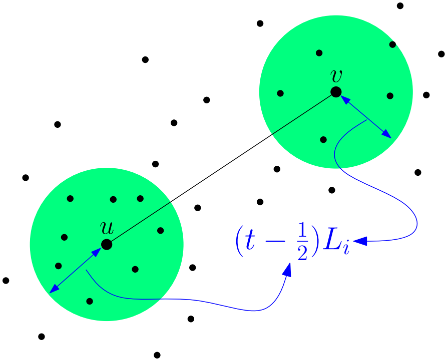
Improved Algorithm

Main ideas:

- Splitting all pairs of points into $\mathcal{O}(n)$ buckets and processing a bucket at a time.
- Keeping the weight matrix up-to-date for all pairs in processing bucket.
    - Update the weight matrix at the beginning of processing a bucket (by running bounded Dijkstra's algorithm).
    - After adding an edge, update the weight matrix (partially).
    - Instead of running Dijkstra's algorithm from scratch, we fix the previous run.

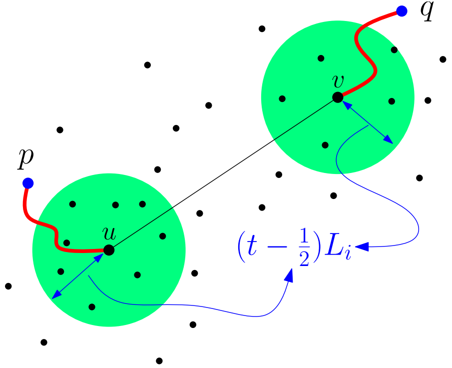
Algorithm

Updating the weight matrix after adding an edge



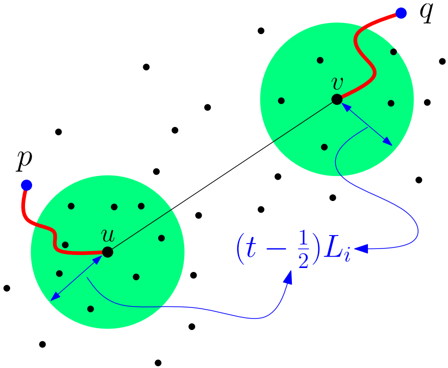
Algorithm

Updating the weight matrix after adding an edge



Algorithm

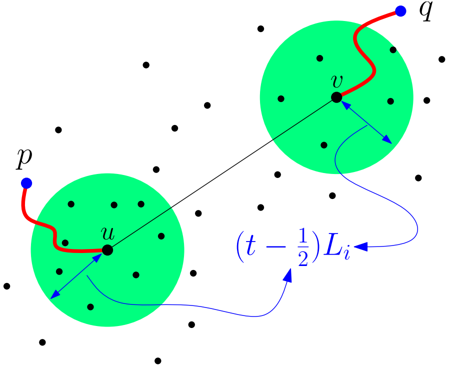
Updating the weight matrix after adding an edge



$$\mathbf{d}_G(p,q) \quad = \quad \mathbf{d}_G(p,u) + \mathbf{d}(u,v) + \mathbf{d}_G(v,q)$$

Algorithm

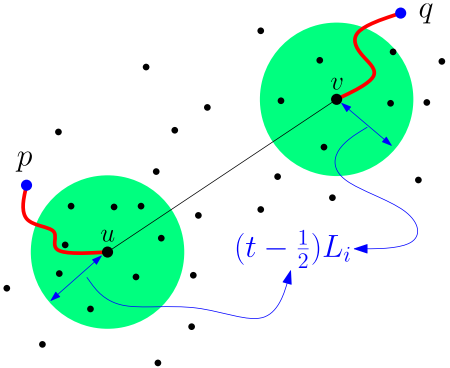
Updating the weight matrix after adding an edge



$$\begin{aligned} \mathbf{d}_G(p,q) &= \mathbf{d}_G(p,u) + \mathbf{d}(u,v) + \mathbf{d}_G(v,q) \\ &\geq \mathbf{d}(p,u) + \mathbf{d}(u,v) + \mathbf{d}(v,q) \end{aligned}$$

Algorithm

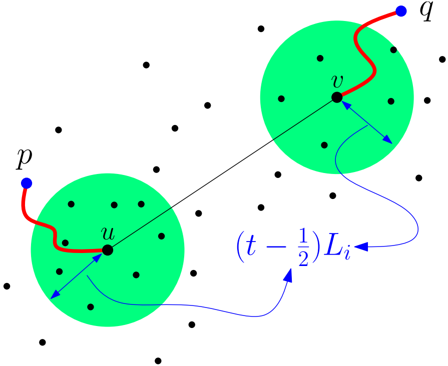
Updating the weight matrix after adding an edge



$$
\begin{aligned}
\mathbf{d}_G(p,q) &= \mathbf{d}_G(p,u) + \mathbf{d}(u,v) + \mathbf{d}_G(v,q) \\
&\geq \mathbf{d}(p,u) + \mathbf{d}(u,v) + \mathbf{d}(v,q) \\
&\geq 2(t - \frac{1}{2})L_i + L_i = 2tL_i
\end{aligned}
$$

Algorithm

Updating the weight matrix after adding an edge



$$
\begin{aligned}
\mathbf{d}_G(p,q) &= \mathbf{d}_G(p,u) + \mathbf{d}(u,v) + \mathbf{d}_G(v,q) \\
&\geq \mathbf{d}(p,u) + \mathbf{d}(u,v) + \mathbf{d}(v,q) \\
&\geq 2(t - \frac{1}{2})L_i + L_i = 2tL_i \\
&> t \cdot \mathbf{d}(p,q).
\end{aligned}
$$

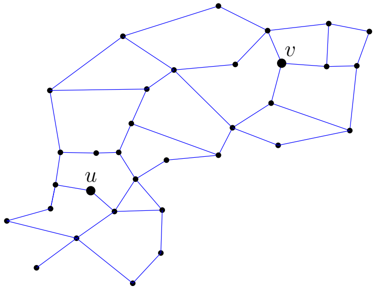
Improved Algorithm

Main ideas:

- Splitting all pairs of points into $\mathcal{O}(n)$ buckets and processing a bucket at a time.
- Keeping the weight matrix up-to-date for all pairs in processing bucket.
  - Update the weight matrix at the beginning of processing a bucket (by running bounded Dijkstra's algorithm).
  - After adding an edge, update the weight matrix (partially).
  - Instead of running Dijkstra's algorithm from scratch, we fix the previous run.

Improved Algorithm

Fixing Dijkstra

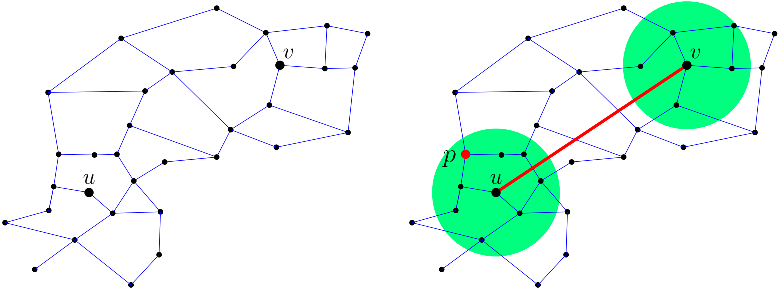


Observations:

- The Dijkstra's algorithm with source p are the same on G and G' until the key of the element on the top of the priority queue is less than $\mathbf{d}_G(p,u) + \mathbf{d}(u,v)$.
- For a fix point p, and a fixed bucket E_i, the number of times bounded SSSP (p) is computed is at most $\mathcal{O}(1/(t-1))$.

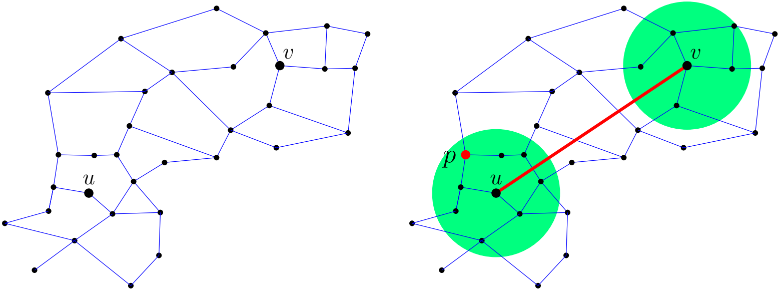
Improved Algorithm

Fixing Dijkstra



Observations:

- The Dijkstra's algorithm with source p are the same on G and G' until the key of the element on the top of the priority queue is less than $\mathbf{d}_G(p,u) + \mathbf{d}(u,v)$.
- For a fix point p, and a fixed bucket E_i, the number of times bounded SSSP (p) is computed is at most $\mathcal{O}(1/(t-1))$.



Observations:

- The Dijkstra's algorithm with source p are the same on G and G' until the key of the element on the top of the priority queue is less than $\mathbf{d}_G(p,u) + \mathbf{d}(u,v)$.
- For a fix point p, and a fixed bucket E_i, the number of times bounded SSSP (p) is computed is at most $\mathcal{O}(1/(t-1))$.

Comments

- This results in an algorithm that uses only $\mathcal{O}(n)$ shortest path computations all together.
- This works for other metrics - but then the number of edges in the greedy graph may not be linear and the number of times shortest paths need to be computed will be proportional to that.
- The algorithm is fairly simple - uses Dijkstra's SSSP + knowing whether a point is in a disc or not.

OPEN PROBLEMS

- Is sorting all pairs of points really necessary to compute the greedy spanner?
- Can we compute greedy spanner for points in the plane under Euclidean metric in sub-quadratic time?

Comments

- This results in an algorithm that uses only $\mathcal{O}(n)$ shortest path computations all together.
- This works for other metrics - but then the number of edges in the greedy graph may not be linear and the number of times shortest paths need to be computed will be proportional to that.
- The algorithm is fairly simple - uses Dijkstra's SSSP + knowing whether a point is in a disc or not.

OPEN PROBLEMS

- Is sorting all pairs of points really necessary to compute the greedy spanner?
- Can we compute greedy spanner for points in the plane under Euclidean metric in sub-quadratic time?

Comments

- This results in an algorithm that uses only $\mathcal{O}(n)$ shortest path computations all together.
- This works for other metrics - but then the number of edges in the greedy graph may not be linear and the number of times shortest paths need to be computed will be proportional to that.
- The algorithm is fairly simple - uses Dijkstra's SSSP + knowing whether a point is in a disc or not.

OPEN PROBLEMS

- Is sorting all pairs of points really necessary to compute the greedy spanner?
- Can we compute greedy spanner for points in the plane under Euclidean metric in sub-quadratic time?

Comments

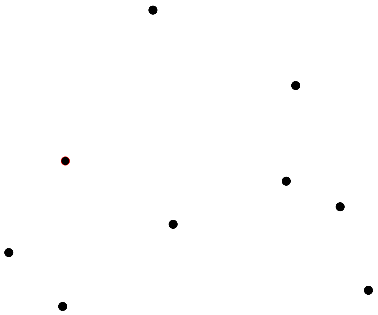
- This results in an algorithm that uses only $\mathcal{O}(n)$ shortest path computations all together.
- This works for other metrics - but then the number of edges in the greedy graph may not be linear and the number of times shortest paths need to be computed will be proportional to that.
- The algorithm is fairly simple - uses Dijkstra's SSSP + knowing whether a point is in a disc or not.

OPEN PROBLEMS

- Is sorting all pairs of points really necessary to compute the greedy spanner?
- Can we compute greedy spanner for points in the plane under Euclidean metric in sub-quadratic time?

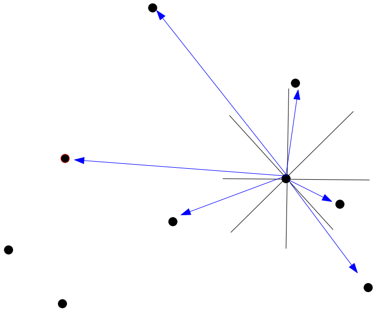
Θ-Graph Construction

- For each point p, divide the plane around p into cones, with angle $\Theta \leq \pi/4$.
- For each cone, connect p to its nearest neighbor in the cone.



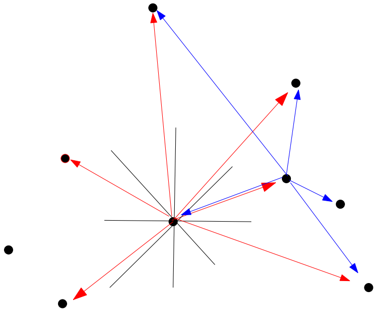
Θ-Graph Construction

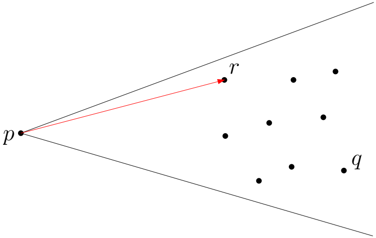
- For each point p, divide the plane around p into cones, with angle $\Theta \leq \pi/4$.
- For each cone, connect p to its nearest neighbor in the cone.



Θ-Graph Construction

- For each point p, divide the plane around p into cones, with angle $\Theta \leq \pi/4$.
- For each cone, connect p to its nearest neighbor in the cone.





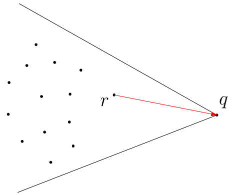
- For each p and for each θ-cone with apex p:
  - edge (p, r), where r is an **approximate** nearest neighbor of p in the cone
  - this edge takes us closer to q by at least $(\cos\theta - \sin\theta)|pr|$
  - stretch factor $t \leq \frac{1}{\cos\theta - \sin\theta} \rightarrow 1$ if $\theta \rightarrow 0$

Properties of the Θ-Graph

- **Good:**
  - time to construct: $O(n \log n)$
  - outdegree: $O(1)$
  - number of edges: $O(n)$
- **Bad:**
  - indegree: $n - 1$
  - weight: $\Omega(n \cdot wt(\text{MST}))$
  - spanner diameter: $n - 1$

q-Sink Spanner: Only Require Short Paths to a Fixed q

- θ-cones with apex q, each one containing at most $n/2$ points
- for each cone:
  - edge (r, q), where r is the nearest neighbor of q in the cone
  - r-sink spanner on the points in the cone

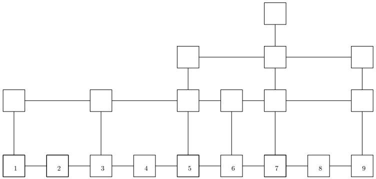


- outdegree $= 1$; indegree $= O(1)$; $O(n \log n)$ time

Variant of the Θ-Graph: Bounded Degree

- $G = \Theta$-graph (or any spanner whose outdegree is $O(1)$)
- For each vertex q:
  - let $\text{IN}(q) = \{p : (p, q) \in G\}$
  - replace incoming edges of q by a q-sink spanner of $\text{IN}(q) \cup \{q\}$
- This gives a spanner whose in- and out-degree is $O(1)$

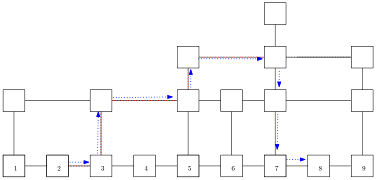
1d-small diameter skip list spanner



Properties

- Height is $\mathcal{O}(\log n)$.
- Total size is $\mathcal{O}(n)$.
- Path has length $\mathcal{O}(\log n)$.

1d-small diameter skip list spanner

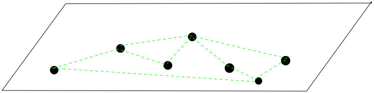


Properties

- Height is $\mathcal{O}(\log n)$.
- Total size is $\mathcal{O}(n)$.
- Path has length $\mathcal{O}(\log n)$.

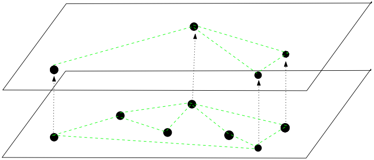
Variant of the Θ-Graph: Reduce the Spanner Diameter

- Generalize **skip lists**
- Define
  - $S_0 = S$
  - S_i contains each point of S_{i-1} with probability $1/2$
- Construct a Θ-graph for each subset S_i



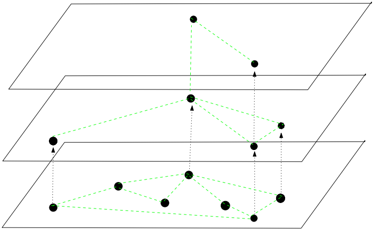
Variant of the Θ-Graph: Reduce the Spanner Diameter

- Generalize **skip lists**
- Define
  - $S_0 = S$
  - S_i contains each point of S_{i-1} with probability $1/2$
- Construct a Θ-graph for each subset S_i



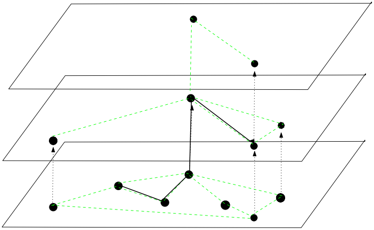
Variant of the Θ-Graph: Reduce the Spanner Diameter

- Generalize **skip lists**
- Define
  - $S_0 = S$
  - S_i contains each point of S_{i-1} with probability $1/2$
- Construct a Θ-graph for each subset S_i



Variant of the Θ-Graph: Reduce the Spanner Diameter

- Generalize **skip lists**
- Define
  - $S_0 = S$
  - S_i contains each point of S_{i-1} with probability $1/2$
- Construct a Θ-graph for each subset S_i



Skip-list Spanner

How to find the path between p and q?

- Follow the nose algorithm from p and q in S_0 till either the spanner path is found or the vertices p_1 and q_1 at the next level are found.
- Repeat the above step for p_1 and q_1 at the next level.
- At some level i, the path from p_i and q_i will meet at a vertex.
- Trace back these two paths all the way back to the first level to obtain the spanner path!

This gives:

- stretch factor $t \leq \frac{1}{\cos\theta - \sin\theta}$
- $O(n)$ edges (w.h.p.)
- spanner diameter $O(\log n)$ with high probability! WHY?

Skip-list Spanner

How to find the path between p and q?

- Follow the nose algorithm from p and q in S_0 till either the spanner path is found or the vertices p_1 and q_1 at the next level are found.
- Repeat the above step for p_1 and q_1 at the next level.
- At some level i, the path from p_i and q_i will meet at a vertex.
- Trace back these two paths all the way back to the first level to obtain the spanner path!

This gives:

- stretch factor $t \leq \frac{1}{\cos\theta - \sin\theta}$
- $O(n)$ edges (w.h.p.)
- spanner diameter $O(\log n)$ with high probability! WHY?

Skip-list Spanner

How to find the path between p and q?

- Follow the nose algorithm from p and q in S_0 till either the spanner path is found or the vertices p_1 and q_1 at the next level are found.
- Repeat the above step for p_1 and q_1 at the next level.
- At some level i, the path from p_i and q_i will meet at a vertex.
- Trace back these two paths all the way back to the first level to obtain the spanner path!

This gives:

- stretch factor $t \leq \frac{1}{\cos\theta - \sin\theta}$
- $O(n)$ edges (w.h.p.)
- spanner diameter $O(\log n)$ with high probability! WHY?

Skip-list Spanner

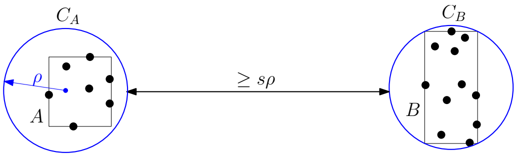
How to find the path between p and q?

- Follow the nose algorithm from p and q in S_0 till either the spanner path is found or the vertices p_1 and q_1 at the next level are found.
- Repeat the above step for p_1 and q_1 at the next level.
- At some level i, the path from p_i and q_i will meet at a vertex.
- Trace back these two paths all the way back to the first level to obtain the spanner path!

This gives:

- stretch factor $t \leq \frac{1}{\cos\theta - \sin\theta}$
- $O(n)$ edges (w.h.p.)
- spanner diameter $O(\log n)$ with high probability! WHY?

Well-Separate Pair Decomposition

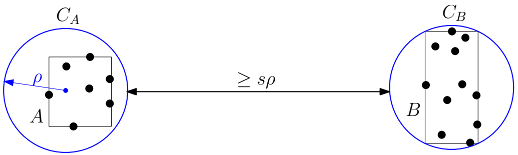


Well-Separated Pair

Let $s > 0$ be a real number, and let A and B two finite sets of points, then we say that A and B are *well-separated with respect to s* if there are two disjoint balls C_A and C_B, such that

- C_A and C_B have the same radius
- C_A contains the bounding box of A
- C_B contains the bounding box of B
- the distance between C_A and C_B is greater than s times ρ

Well-Separate Pair Decomposition

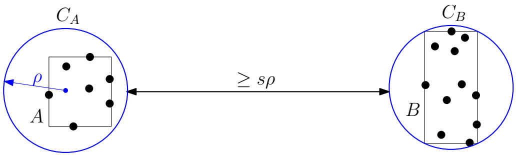


Well-Separated Pair

Let $s > 0$ be a real number, and let A and B two finite sets of points, then we say that A and B are *well-separated with respect to s* if there are two disjoint balls C_A and C_B, such that

- C_A and C_B have the same radius
- C_A contains the bounding box of A
- C_B contains the bounding box of B
- the distance between C_A and C_B is greater than s times ρ

Well-Separate Pair Decomposition

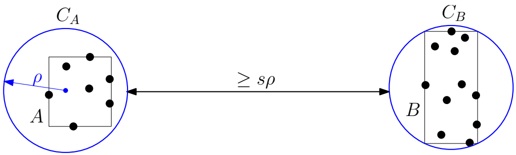


Well-Separated Pair

Let $s > 0$ be a real number, and let A and B two finite sets of points, then we say that A and B are *well-separated with respect to s* if there are two disjoint balls C_A and C_B, such that

- C_A and C_B have the same radius
- C_A contains the bounding box of A
- C_B contains the bounding box of B
- the distance between C_A and C_B is greater than s times ρ

Well-Separate Pair Decomposition

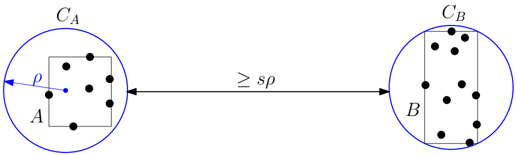


Well-Separated Pair

Let $s > 0$ be a real number, and let A and B two finite sets of points, then we say that A and B are *well-separated with respect to s* if there are two disjoint balls C_A and C_B, such that

- C_A and C_B have the same radius
- C_A contains the bounding box of A
- C_B contains the bounding box of B
- the distance between C_A and C_B is greater than s times ρ

Well-Separate Pair Decomposition

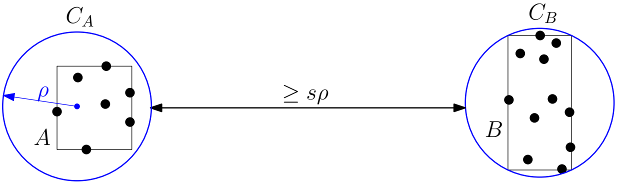


Well-Separated Pair

Let $s > 0$ be a real number, and let A and B two finite sets of points, then we say that A and B are *well-separated with respect to s* if there are two disjoint balls C_A and C_B, such that

- C_A and C_B have the same radius
- C_A contains the bounding box of A
- C_B contains the bounding box of B
- the distance between C_A and C_B is greater than s times ρ

WSPD Contd.

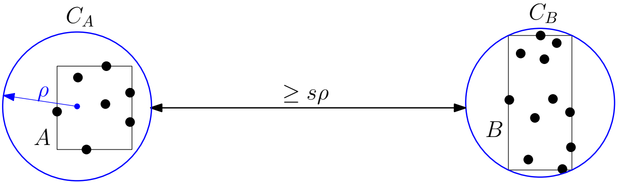


Well-Separated Pair Decomposition with respect to s

Is a sequence $\{A_1, B_1\}, \{A_2, B_2\}, \ldots \{A_m, B_m\}$ of pairs such that

- A_i and B_i are well separated with respect to s,
- for any two points p and q there is exactly one pair, such that $p \in A_i$ and $q \in B_i$, and
- there are linear number of such pairs ($m = \mathcal{O}(n)$).

WSPD Contd.

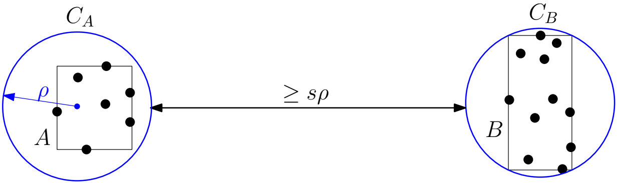


Well-Separated Pair Decomposition with respect to s

Is a sequence $\{A_1, B_1\}, \{A_2, B_2\}, \ldots \{A_m, B_m\}$ of pairs such that

- A_i and B_i are well separated with respect to s,
- for any two points p and q there is exactly one pair, such that $p \in A_i$ and $q \in B_i$, and
- there are linear number of such pairs ($m = \mathcal{O}(n)$).

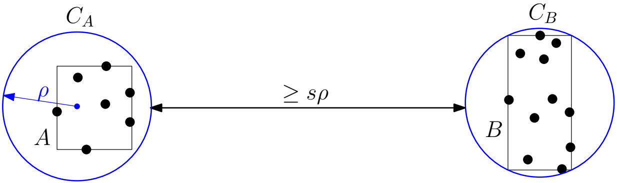
WSPD Contd.



Well-Separated Pair Decomposition with respect to s

Is a sequence $\{A_1, B_1\}, \{A_2, B_2\}, \dots \{A_m, B_m\}$ of pairs such that

- A_i and B_i are well separated with respect to s,
- for any two points p and q there is exactly one pair, such that $p \in A_i$ and $q \in B_i$, and
- there are linear number of such pairs ($m = \mathcal{O}(n)$).



Well-Separated Pair Decomposition with respect to s

Is a sequence $\{A_1, B_1\}, \{A_2, B_2\}, \ldots \{A_m, B_m\}$ of pairs such that

- A_i and B_i are well separated with respect to s,
- for any two points p and q there is exactly one pair, such that $p \in A_i$ and $q \in B_i$, and
- there are linear number of such pairs ($m = \mathcal{O}(n)$).

Remarks

- Proposed by Callahan and Kosaraju in 1992.
- Essentially it says that “the number of distinct distances in the plane among a set of n-points is linear and NOT quadratic”.
- Many problems based on distances between pairs of points can be solved efficiently (e.g., n-body simulation, closest pair, all nearest-neighbors, spanners).
- Works very nicely in higher dimensions (Computation Time of WSPDs $\rightarrow \mathcal{O}(dn \log n)$. Computation Time of Voronoi Diagrams/Range trees etc. $\rightarrow \mathcal{O}(n \log^{\mathcal{O}(d)} n)$. $d =$ dimension).

WSPD Contd.

Remarks

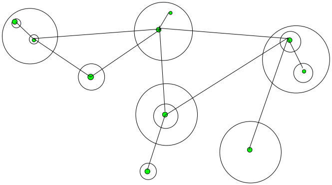
- Proposed by Callahan and Kosaraju in 1992.
- Essentially it says that “the number of distinct distances in the plane among a set of n-points is linear and NOT quadratic”.
- Many problems based on distances between pairs of points can be solved efficiently (e.g., n-body simulation, closest pair, all nearest-neighbors, spanners).
- Works very nicely in higher dimensions (Computation Time of WSPDs $\rightarrow \mathcal{O}(dn \log n)$. Computation Time of Voronoi Diagrams/Range trees etc. $\rightarrow \mathcal{O}(n \log^{\mathcal{O}(d)} n)$. $d =$ dimension).

Remarks

- Proposed by Callahan and Kosaraju in 1992.
- Essentially it says that "the number of distinct distances in the plane among a set of n-points is linear and NOT quadratic".
- Many problems based on distances between pairs of points can be solved efficiently (e.g., n-body simulation, closest pair, all nearest-neighbors, spanners).
- Works very nicely in higher dimensions (Computation Time of WSPDs $\rightarrow \mathcal{O}(dn \log n)$. Computation Time of Voronoi Diagrams/Range trees etc. $\rightarrow \mathcal{O}(n \log^{\mathcal{O}(d)} n)$.
  $d =$ dimension).

WSPD-Spanner

- Given a WSPD $\{A_1, B_1\}, \{A_2, B_2\}, \dots, \{A_m, B_m\}$
- For each i, take one edge (a_i, b_i), where $a_i \in A_i$ and $b_i \in B_i$



- This gives a t-spanner for
$$t = \frac{s+4}{s-4} \rightarrow 1 \text{ if } s \rightarrow \infty$$
- WSPD with $m = O(n)$ can be computed in $O(n \log n)$ time

WSPD-Spanner

- degree can be large, but
  - the edges can be directed such that the outdegree is $O(1)$
  - combined with sink spanners: degree $O(1)$
  - time to construct: $O(n \log n)$
- using the gap property (and more): weight is $O(\text{MST} \cdot \log n)$
- choose the edges more carefully: spanner diameter $O(\log n)$
- WSPD-spanner can be represented by $O(1)$ trees such that for all p, q, one of the trees contains a t-spanner path between p and q.
  - by adding shortcuts to these trees: spanner diameter $O(\alpha(n))$ and $O(n)$ edges

Computation of WSPD

Main Steps in Constructing WSPD

- Compute a SPLIT TREE.
- Extract well-separated pairs from the Split Tree.

Split Tree

Definition

It is a recursively defined binary tree, where each node stores a bounding rectangle of points in its subtree.

- Root stores the bounding rectangle of the whole set S.
- If $|S| = 1$, then the split tree is a single node storing that point.
- Otherwise, split the bounding rectangle of S by splitting the longest side into two halves. This splits the point sets into two - S_1 and S_2. Split tree consists of a node corresponding to S and two children corresponding to recursively defined split trees of S_1 and S_2.

Split Tree

Definition

It is a recursively defined binary tree, where each node stores a bounding rectangle of points in its subtree.

- Root stores the bounding rectangle of the whole set S.
- If $|S| = 1$, then the split tree is a single node storing that point.
- Otherwise, split the bounding rectangle of S by splitting the longest side into two halves. This splits the point sets into two - S_1 and S_2. Split tree consists of a node corresponding to S and two children corresponding to recursively defined split trees of S_1 and S_2.

Split Tree

Definition

It is a recursively defined binary tree, where each node stores a bounding rectangle of points in its subtree.

- Root stores the bounding rectangle of the whole set S.
- If $|S| = 1$, then the split tree is a single node storing that point.
- Otherwise, split the bounding rectangle of S by splitting the longest side into two halves. This splits the point sets into two - S_1 and S_2. Split tree consists of a node corresponding to S and two children corresponding to recursively defined split trees of S_1 and S_2.

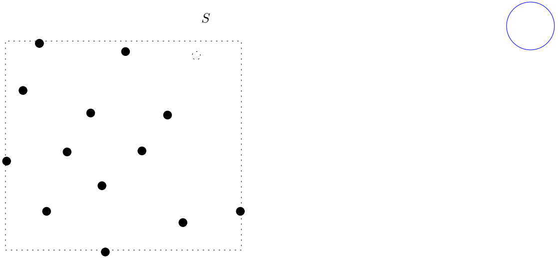
Split Tree

Definition

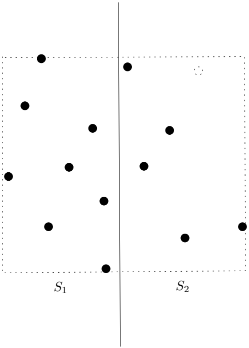
It is a recursively defined binary tree, where each node stores a bounding rectangle of points in its subtree.

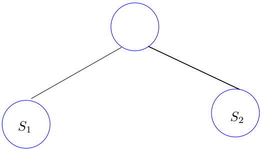
- Root stores the bounding rectangle of the whole set S.
- If $|S| = 1$, then the split tree is a single node storing that point.
- Otherwise, split the bounding rectangle of S by splitting the longest side into two halves. This splits the point sets into two - S_1 and S_2. Split tree consists of a node corresponding to S and two children corresponding to recursively defined split trees of S_1 and S_2.

Split Tree Computation

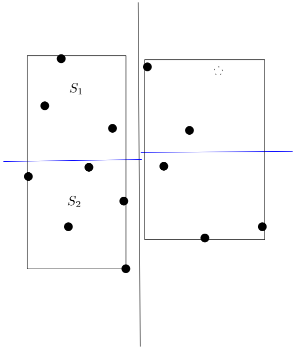


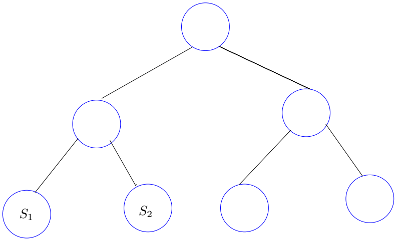
Split Tree Computation



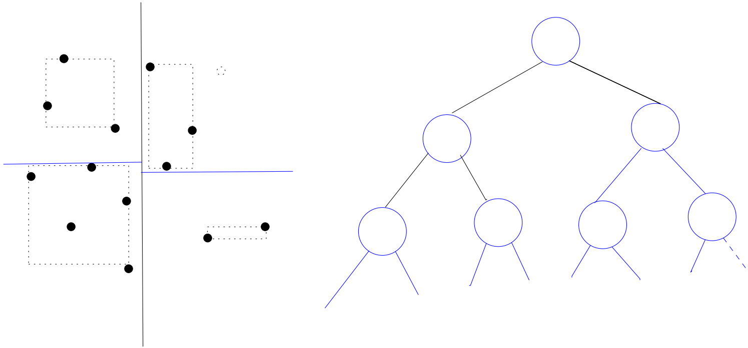


Split Tree Computation





Split Tree Computation



Computing the WSPD

For each internal node u of the Split Tree execute FINDPAIRS(v, w);
v is left child of u and w is the right child.

FINDPAIRS(v, w)

if S_v *and* S_w *are well-separated* **then**
  return the node pair $\{v, w\}$
end
else
  (assume that longer side of bounding rectangle of v is smaller than that of w);
  $w_l :=$ left child of w;
  $w_r :=$ right child of w;
  FINDPAIRS(v, w_l);
  FINDPAIRS(v, w_r)
end

Computing the WSPD

For each internal node u of the Split Tree execute FINDPAIRS(v, w);
v is left child of u and w is the right child.

FINDPAIRS(v, w)

if S_v *and* S_w *are well-separated* **then**
  return the node pair $\{v, w\}$
end
else
  (assume that longer side of bounding rectangle of v is smaller than that of w);
  $w_l :=$ left child of w;
  $w_r :=$ right child of w;
  FINDPAIRS(v, w_l);
  FINDPAIRS(v, w_r)
end

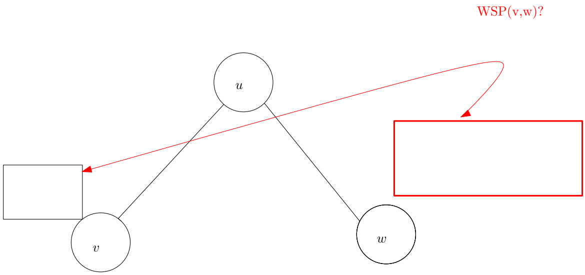
Computing the WSPD

For each internal node u of the Split Tree execute FINDPAIRS(v, w);
v is left child of u and w is the right child.

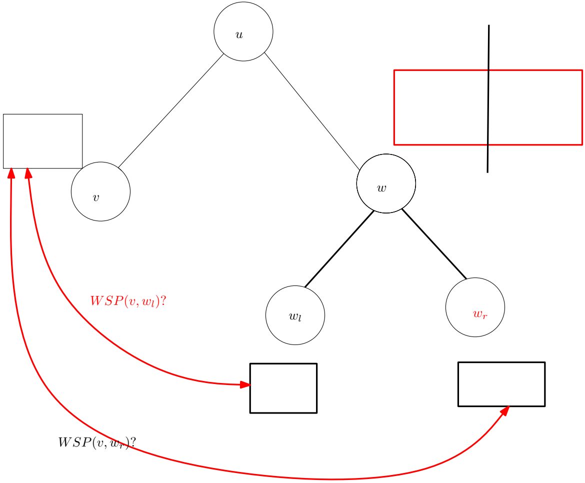
FINDPAIRS(v, w)

if S_v *and* S_w *are well-separated* **then**
  return the node pair $\{v, w\}$
end
else
  (assume that longer side of bounding rectangle of v is smaller than that of w);
  $w_l :=$ left child of w;
  $w_r :=$ right child of w;
  FINDPAIRS(v, w_l);
  FINDPAIRS(v, w_r)
end

Illustration



Illustration



Why does it work?

Lemma

Let $\{v_i, w_i\}$, $1 \leq i \leq m$, be the sequence of node pairs returned by the algorithm FINDPAIRS. The sequence $\{S_{v_1}, S_{w_1}\}, \cdots, \{S_{v_m}, S_{w_m}\}$ is a WSPD for the set S.

Proof

Observe that each of the pairs $\{S_{v_i}, S_{w_i}\}$ is well separated and non-empty.
Consider any two points p and q in S.
Consider $LCA(p, q)$ in the fair split tree
Let $u = LCA(p, q)$ and its left and right children be v and w.
Consider the call to FINDPAIRS(v, w).
If S_v and S_w are well separated, then $p \in S_v$ and $q \in S_w$, or we make calls to the appropriate children, and by induction on the number of recursive calls we can show that p and q are in a unique pair which is well separated.

Why does it work?

Lemma

Let $\{v_i, w_i\}$, $1 \leq i \leq m$, be the sequence of node pairs returned by the algorithm FINDPAIRS. The sequence $\{S_{v_1}, S_{w_1}\}, \cdots, \{S_{v_m}, S_{w_m}\}$ is a WSPD for the set S.

Proof

Observe that each of the pairs $\{S_{v_i}, S_{w_i}\}$ is well separated and non-empty.

Consider any two points p and q in S.

Consider $LCA(p, q)$ in the fair split tree

Let $u = LCA(p, q)$ and its left and right children be v and w.

Consider the call to FINDPAIRS(v, w).

If S_v and S_w are well separated, then $p \in S_v$ and $q \in S_w$, or we make calls to the appropriate children, and by induction on the number of recursive calls we can show that p and q are in a unique pair which is well separated.

Why does it work?

Lemma

Let $\{v_i, w_i\}$, $1 \leq i \leq m$, be the sequence of node pairs returned by the algorithm FINDPAIRS. The sequence $\{S_{v_1}, S_{w_1}\}, \cdots, \{S_{v_m}, S_{w_m}\}$ is a WSPD for the set S.

Proof

Observe that each of the pairs $\{S_{v_i}, S_{w_i}\}$ is well separated and non-empty.
Consider any two points p and q in S.
Consider $LCA(p, q)$ in the fair split tree
Let $u = LCA(p, q)$ and its left and right children be v and w.
Consider the call to FINDPAIRS(v, w).
If S_v and S_w are well separated, then $p \in S_v$ and $q \in S_w$, or we make calls to the appropriate children, and by induction on the number of recursive calls we can show that p and q are in a unique pair which is well separated.

Why does it work?

Lemma

Let $\{v_i, w_i\}$, $1 \leq i \leq m$, be the sequence of node pairs returned by the algorithm FINDPAIRS. The sequence $\{S_{v_1}, S_{w_1}\}, \cdots, \{S_{v_m}, S_{w_m}\}$ is a WSPD for the set S.

Proof

Observe that each of the pairs $\{S_{v_i}, S_{w_i}\}$ is well separated and non-empty.
Consider any two points p and q in S.
Consider $LCA(p, q)$ in the fair split tree
Let $u = LCA(p, q)$ and its left and right children be v and w.
Consider the call to FINDPAIRS(v, w).
If S_v and S_w are well separated, then $p \in S_v$ and $q \in S_w$, or we make calls to the appropriate children, and by induction on the number of recursive calls we can show that p and q are in a unique pair which is well separated.

Why does it work?

Lemma

Let $\{v_i, w_i\}$, $1 \leq i \leq m$, be the sequence of node pairs returned by the algorithm FINDPAIRS. The sequence $\{S_{v_1}, S_{w_1}\}, \cdots, \{S_{v_m}, S_{w_m}\}$ is a WSPD for the set S.

Proof

Observe that each of the pairs $\{S_{v_i}, S_{w_i}\}$ is well separated and non-empty.
Consider any two points p and q in S.
Consider $LCA(p, q)$ in the fair split tree
Let $u = LCA(p, q)$ and its left and right children be v and w.
Consider the call to FINDPAIRS(v, w).
If S_v and S_w are well separated, then $p \in S_v$ and $q \in S_w$, or we make calls to the appropriate children, and by induction on the number of recursive calls we can show that p and q are in a unique pair which is well separated.

Why does it work?

Lemma

Let $\{v_i, w_i\}$, $1 \leq i \leq m$, be the sequence of node pairs returned by the algorithm FINDPAIRS. The sequence $\{S_{v_1}, S_{w_1}\}, \cdots, \{S_{v_m}, S_{w_m}\}$ is a WSPD for the set S.

Proof

Observe that each of the pairs $\{S_{v_i}, S_{w_i}\}$ is well separated and non-empty.
Consider any two points p and q in S.
Consider $LCA(p, q)$ in the fair split tree
Let $u = LCA(p, q)$ and its left and right children be v and w.
Consider the call to FINDPAIRS(v, w).
If S_v and S_w are well separated, then $p \in S_v$ and $q \in S_w$, or we make calls to the appropriate children, and by induction on the number of recursive calls we can show that p and q are in a unique pair which is well separated.

Why does it work?

Lemma

Let $\{v_i, w_i\}$, $1 \leq i \leq m$, be the sequence of node pairs returned by the algorithm FINDPAIRS. The sequence $\{S_{v_1}, S_{w_1}\}, \cdots, \{S_{v_m}, S_{w_m}\}$ is a WSPD for the set S.

Proof

Observe that each of the pairs $\{S_{v_i}, S_{w_i}\}$ is well separated and non-empty.
Consider any two points p and q in S.
Consider $LCA(p, q)$ in the fair split tree
Let $u = LCA(p, q)$ and its left and right children be v and w.
Consider the call to FINDPAIRS(v, w).
If S_v and S_w are well separated, then $p \in S_v$ and $q \in S_w$, or we make calls to the appropriate children, and by induction on the number of recursive calls we can show that p and q are in a unique pair which is well separated.

Why is $m = O(n)$?

Why the number of pairs returned by the algorithm is linear and not quadratic?

Where are the $\binom{n}{2}$ distances hiding?

The key in understanding this is the packing argument.

Key Observation

If (S_v, S_w) is a pair in the WSPD, then the pairs $(S_{\pi(v)}, S_w)$ or $(S_v, S_{\pi(w)})$ are not well-separated.

An Implication

Let (S_v, S_{w_i}), $1 \leq i \leq k$, be all the pairs in the WSPD that have S_v as one of the component. Then all the sets S_{w_i}'s are pairwise disjoint.

Why is $m = O(n)$?

Why the number of pairs returned by the algorithm is linear and not quadratic?
Where are the $\binom{n}{2}$ distances hiding?
The key in understanding this is the packing argument.

Key Observation

If (S_v, S_w) is a pair in the WSPD, then the pairs $(S_{\pi(v)}, S_w)$ or $(S_v, S_{\pi(w)})$ are not well-separated.

An Implication

Let (S_v, S_{w_i}), $1 \leq i \leq k$, be all the pairs in the WSPD that have S_v as one of the component. Then all the sets S_{w_i}'s are pairwise disjoint.

Why is $m = O(n)$?

Why the number of pairs returned by the algorithm is linear and not quadratic?
Where are the $\binom{n}{2}$ distances hiding?
The key in understanding this is the packing argument.

Key Observation

If (S_v, S_w) is a pair in the WSPD, then the pairs $(S_{\pi(v)}, S_w)$ or $(S_v, S_{\pi(w)})$ are not well-separated.

An Implication

Let (S_v, S_{w_i}), $1 \leq i \leq k$, be all the pairs in the WSPD that have S_v as one of the component. Then all the sets S_{w_i}'s are pairwise disjoint.

Why is $m = O(n)$?

Why the number of pairs returned by the algorithm is linear and not quadratic?
Where are the $\binom{n}{2}$ distances hiding?
The key in understanding this is the packing argument.

Key Observation

If (S_v, S_w) is a pair in the WSPD, then the pairs $(S_{\pi(v)}, S_w)$ or $(S_v, S_{\pi(w)})$ are not well-separated.

An Implication

Let (S_v, S_{w_i}), $1 \leq i \leq k$, be all the pairs in the WSPD that have S_v as one of the component. Then all the sets S_{w_i}'s are pairwise disjoint.

Why is $m = O(n)$?

Why the number of pairs returned by the algorithm is linear and not quadratic?
Where are the $\binom{n}{2}$ distances hiding?
The key in understanding this is the packing argument.

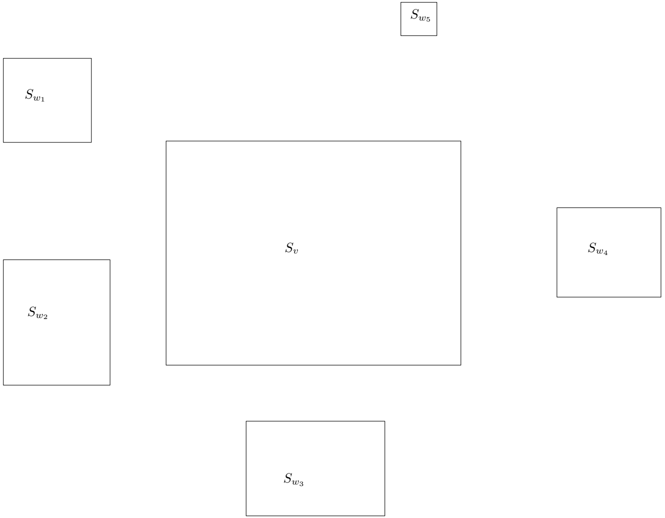
Key Observation

If (S_v, S_w) is a pair in the WSPD, then the pairs $(S_{\pi(v)}, S_w)$ or $(S_v, S_{\pi(w)})$ are not well-separated.

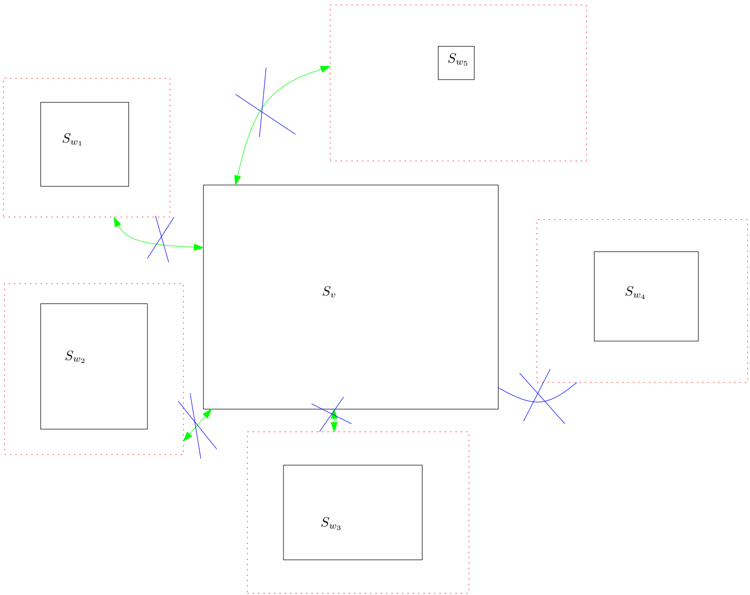
An Implication

Let (S_v, S_{w_i}), $1 \leq i \leq k$, be all the pairs in the WSPD that have S_v as one of the component. Then all the sets S_{w_i}'s are pairwise disjoint.

An Illustration



An Illustration



What do we have now?

Main Theorem

Given a set of n points in the plane, we can compute a well-separated pair decomposition, using fair split tree, of size $\mathcal{O}(n)$, in $\mathcal{O}(n \log n)$ time.

Corollaries

In $\mathcal{O}(n \log n)$ time we can compute

- t-spanner of a point set.
- closest pair in a point set.
- Nearest neighbor of each point in the point set.
- Approximate MST.
- Approximate Diameter.

Main Open Question

Extend the concept of Well-Separated Pair Decomposition to other Metrics.

Applications

Corollaries

In $\mathcal{O}(n \log n)$ time we can compute

- t-spanner of a point set.
- closest pair in a point set.
- Nearest neighbor of each point in the point set.
- Approximate MST.
- Approximate Diameter.

Main Open Question

Extend the concept of Well-Separated Pair Decomposition to other Metrics.

Motivation

Raman and Chebrolu in 2005 proposed a new protocol for communication

- It is not possible for one node to both transmit and receive at the same time
- Therefore, nodes have to alternate between the send and receive states
- In order to solve this they propose a tree structure
- Having chromatic number equal to two

Motivation

Raman and Chebrolu in 2005 proposed a new protocol for communication

- It is not possible for one node to both transmit and receive at the same time
- Therefore, nodes have to alternate between the send and receive states
- In order to solve this they propose a tree structure
- Having chromatic number equal to two

Motivation

Raman and Chebrolu in 2005 proposed a new protocol for communication

- It is not possible for one node to both transmit and receive at the same time
- Therefore, nodes have to alternate between the send and receive states
- In order to solve this they propose a tree structure
- Having chromatic number equal to two

Motivation

Raman and Chebrolu in 2005 proposed a new protocol for communication

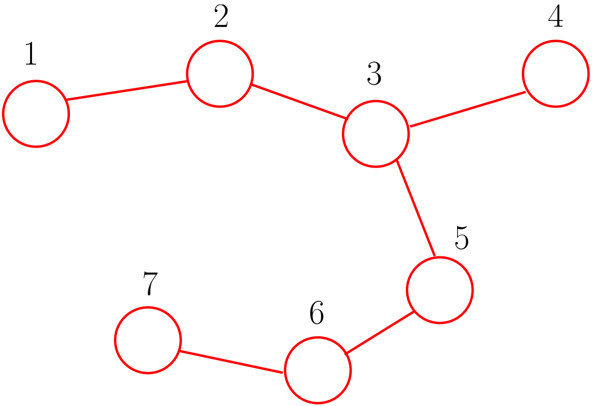
- It is not possible for one node to both transmit and receive at the same time
- Therefore, nodes have to alternate between the send and receive states
- In order to solve this they propose a tree structure
- Having chromatic number equal to two

Motivation

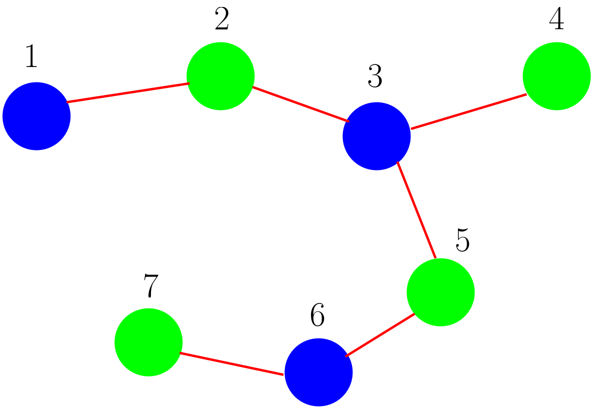
Raman and Chebrolu in 2005 proposed a new protocol for communication

- It is not possible for one node to both transmit and receive at the same time
- Therefore, nodes have to alternate between the send and receive states
- In order to solve this they propose a tree structure
- Having chromatic number equal to two

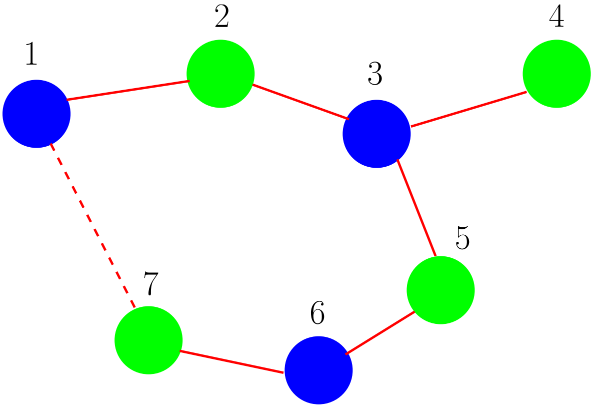
Motivation



Motivation



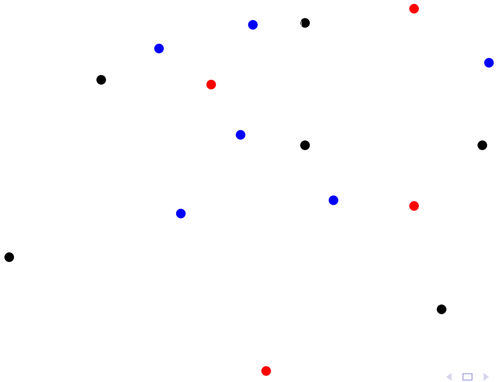
Motivation



Spanners of complete k-partite geometric graphs

Problem I definition

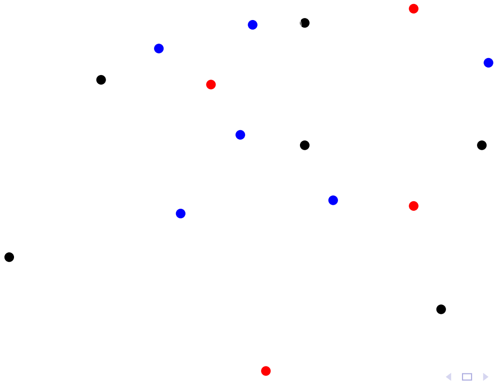
Given a point set P and a coloring of P, compute a t-spanner for this coloring.



Spanners of complete k-partite geometric graphs

Problem definition

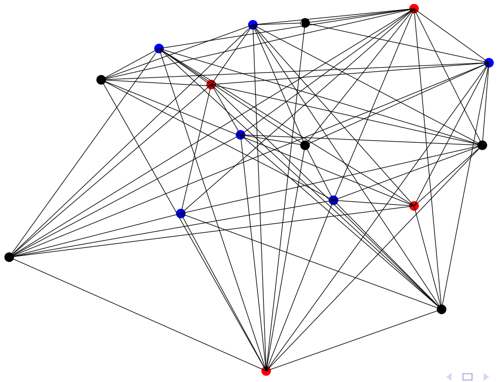
Given a complete k-partite geometric graph G, compute a spanner that has “small” stretch factor and “few” edges.



Spanners of complete k-partite geometric graphs

Problem definition

Given a complete k-partite geometric graph G, compute a spanner that has "small" stretch factor and "few" edges.



Spanners of complete k-partite geometric graphs

Problem definition

Given a complete k-partite geometric graph G, compute a spanner that has “small” stretch factor and “few” edges.

Results

- Algorithm that computes $(5+\epsilon)$-spanner of G, with $O(n)$ edges in $O(n \log n)$ time.
- Algorithm that computes $(3+\epsilon)$-spanner of G, with $O(n \log n)$ edges in $O(n \log n)$ time.
- The later result is optimal.

Spanners of complete k-partite geometric graphs

Problem I definition

Given a complete k-partite geometric graph G, compute a spanner that has "small" stretch factor and "few" edges.

Results

- Algorithm that computes $(5 + \epsilon)$-spanner of G, with $O(n)$ edges in $O(n \log n)$ time.
- Algorithm that computes $(3 + \epsilon)$-spanner of G, with $O(n \log n)$ edges in $O(n \log n)$ time.
- The later result is optimal

Spanners of complete k-partite geometric graphs

Problem definition

Given a complete k-partite geometric graph G, compute a spanner that has “small” stretch factor and “few” edges.

Results

- Algorithm that computes $(5+\epsilon)$-spanner of G, with $O(n)$ edges in $O(n \log n)$ time.
- Algorithm that computes $(3+\epsilon)$-spanner of G, with $O(n \log n)$ edges in $O(n \log n)$ time.
- The later result is optimal

Spanners of complete k-partite geometric graphs

Problem $\mathcal{I}$ definition

Given a complete k-partite geometric graph G, compute a spanner that has "small" stretch factor and "few" edges.

Results

- Algorithm that computes $(5+\epsilon)$-spanner of G, with $O(n)$ edges in $O(n \log n)$ time.
- Algorithm that computes $(3+\epsilon)$-spanner of G, with $O(n \log n)$ edges in $O(n \log n)$ time.
- The later result is optimal

Spanners of complete k-partite geometric graphs

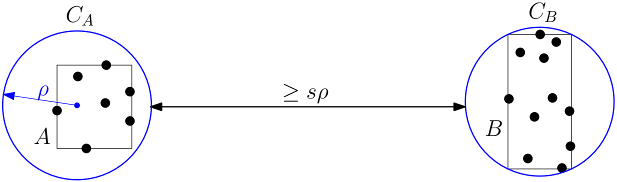
Problem definition

Given a complete k-partite geometric graph G, compute a spanner that has “small” stretch factor and “few” edges.

Results

- Algorithm that computes $(5 + \epsilon)$-spanner of G, with $O(n)$ edges in $O(n \log n)$ time.
- Algorithm that computes $(3 + \epsilon)$-spanner of G, with $O(n \log n)$ edges in $O(n \log n)$ time.
- The later result is optimal

Well-Separate Pair Decomposition (WSPD) [Callahan and Kosaraju 1992]

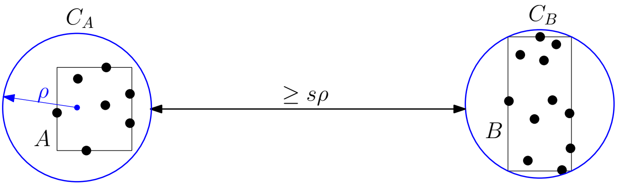


Well-Separated Pair

Let $s > 0$ be a real number, and let A and B two finite sets of points in $\mathbb{R}$, then we say that A and B are *well-separated with respect to s* if there are two disjoint balls C_A and C_B, such that

- C_A and C_B have the same radius
- C_A contains the bounding box of A
- C_B contains the bounding box of B
- the distance between C_A and C_B is greater than s times ρ

Well-Separate Pair Decomposition (WSPD) [Callahan and Kosaraju 1992]

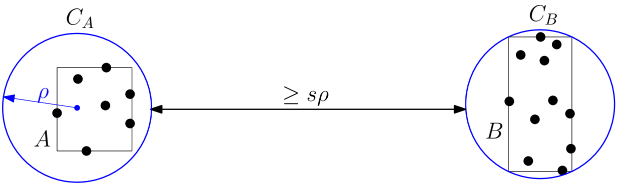


Well-Separated Pair

Let $s > 0$ be a real number, and let A and B two finite sets of points in $\mathbb{R}$, then we say that A and B are *well-separated with respect to s* if there are two disjoint balls C_A and C_B, such that

- C_A and C_B have the same radius
- C_A contains the bounding box of A
- C_B contains the bounding box of B
- the distance between C_A and C_B is greater than s times ρ

Well-Separate Pair Decomposition (WSPD) [Callahan and Kosaraju 1992]

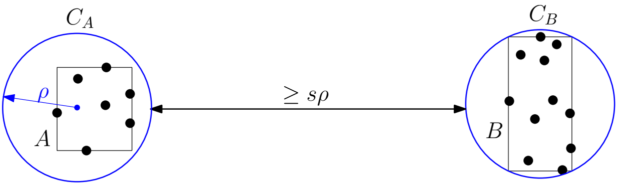


Well-Separated Pair

Let $s > 0$ be a real number, and let A and B two finite sets of points in $\mathbb{R}$, then we say that A and B are *well-separated with respect to s* if there are two disjoint balls C_A and C_B, such that

- C_A and C_B have the same radius
- C_A contains the bounding box of A
- C_B contains the bounding box of B
- the distance between C_A and C_B is greater than s times ρ

Well-Separate Pair Decomposition (WSPD) [Callahan and Kosaraju 1992]

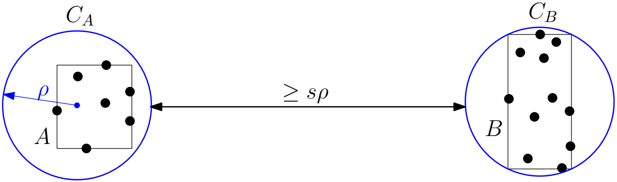


Well-Separated Pair

Let $s > 0$ be a real number, and let A and B two finite sets of points in $\mathbb{R}$, then we say that A and B are *well-separated with respect to s* if there are two disjoint balls C_A and C_B, such that

- C_A and C_B have the same radius
- C_A contains the bounding box of A
- C_B contains the bounding box of B
- the distance between C_A and C_B is greater than s times ρ

Well-Separate Pair Decomposition (WSPD) [Callahan and Kosaraju 1992]

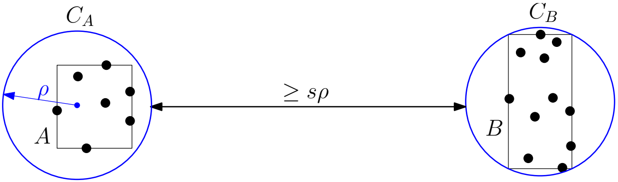


Well-Separated Pair

Let $s > 0$ be a real number, and let A and B two finite sets of points in $\mathbb{R}$, then we say that A and B are *well-separated with respect to s* if there are two disjoint balls C_A and C_B, such that

- C_A and C_B have the same radius
- C_A contains the bounding box of A
- C_B contains the bounding box of B
- the distance between C_A and C_B is greater than s times ρ

Well-Separate Pair Decomposition (WSPD) [Callahan and Kosaraju 1992]

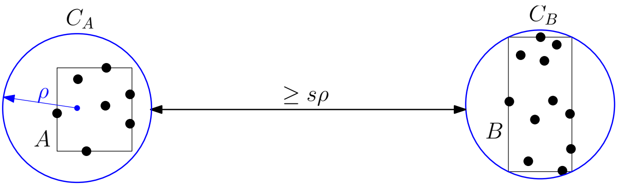


Well-Separated Pair Decomposition with respect to s

Is a sequence $\{A_1, B_1\}, \{A_2, B_2\}, \ldots \{A_m, B_m\}$ of pairs such that

- A_i and B_i are well separated with respect to s
- for any two points p and q there is exactly one pair, such that $p \in A_i$ and $q \in B_i$
- there are linear number of such pairs

Well-Separate Pair Decomposition (WSPD) [Callahan and Kosaraju 1992]

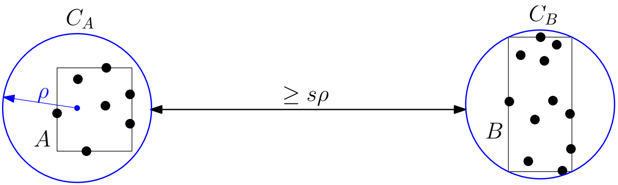


Well-Separated Pair Decomposition with respect to s

Is a sequence $\{A_1, B_1\}, \{A_2, B_2\}, \ldots \{A_m, B_m\}$ of pairs such that

- A_i and B_i are well separated with respect to s
- for any two points p and q there is exactly one pair, such that $p \in A_i$ and $q \in B_i$
- there are linear number of such pairs

Well-Separate Pair Decomposition (WSPD) [Callahan and Kosaraju 1992]

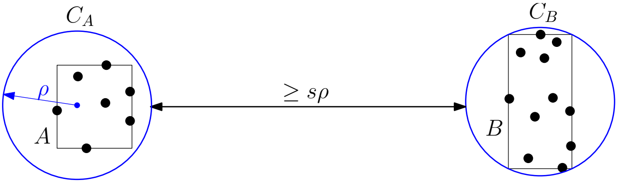


Well-Separated Pair Decomposition with respect to s

Is a sequence $\{A_1, B_1\}, \{A_2, B_2\}, \ldots \{A_m, B_m\}$ of pairs such that

- A_i and B_i are well separated with respect to s
- for any two points p and q there is exactly one pair, such that $p \in A_i$ and $q \in B_i$
- there are linear number of such pairs

Well-Separate Pair Decomposition (WSPD) [Callahan and Kosaraju 1992]



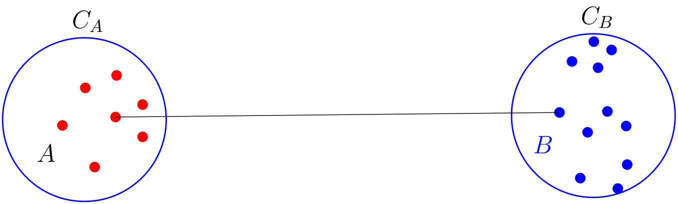
Well-Separated Pair Decomposition with respect to s

Is a sequence $\{A_1, B_1\}, \{A_2, B_2\}, \ldots \{A_m, B_m\}$ of pairs such that

- A_i and B_i are well separated with respect to s
- for any two points p and q there is exactly one pair, such that $p \in A_i$ and $q \in B_i$
- there are linear number of such pairs

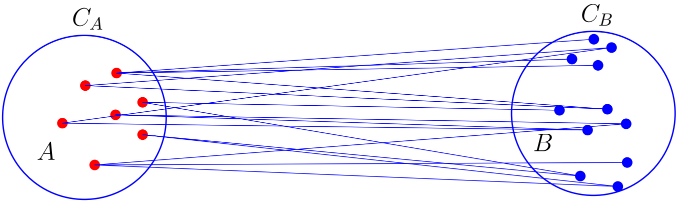
First attempt

First attempt



In WSPD we pick one edge

First attempt



Now we need lots of edges

Second attempt

1. compute a WSPD
2. for each pair $\{A, B\}$ do as follows
   - if both A and B are all red or all blue, then ignore

Second attempt

1. compute a WSPD
2. for each pair $\{A, B\}$ do as follows
   - if both A and B are bi-chromatic, then add an edge

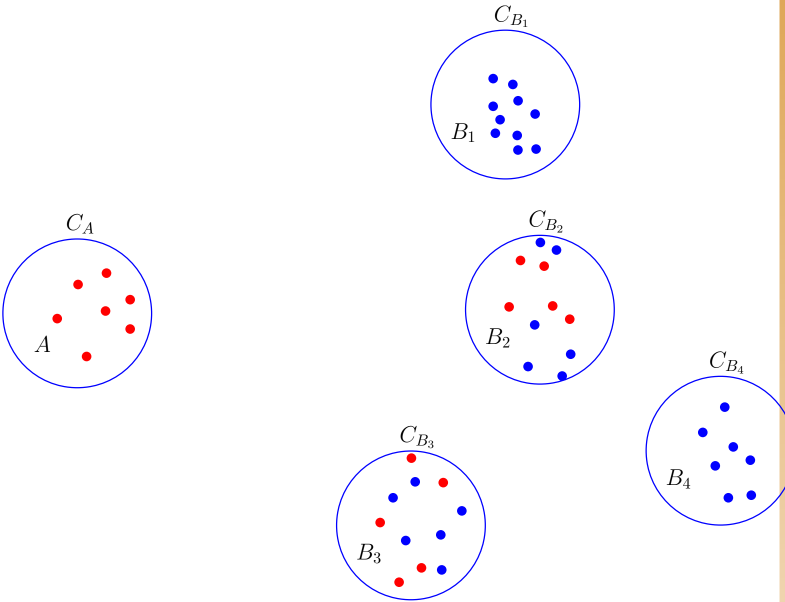
Second attempt

1. compute a WSPD
2. for each pair $\{A, B\}$ do as follows
   - if both A and B are bi-chromatic, then add an edge

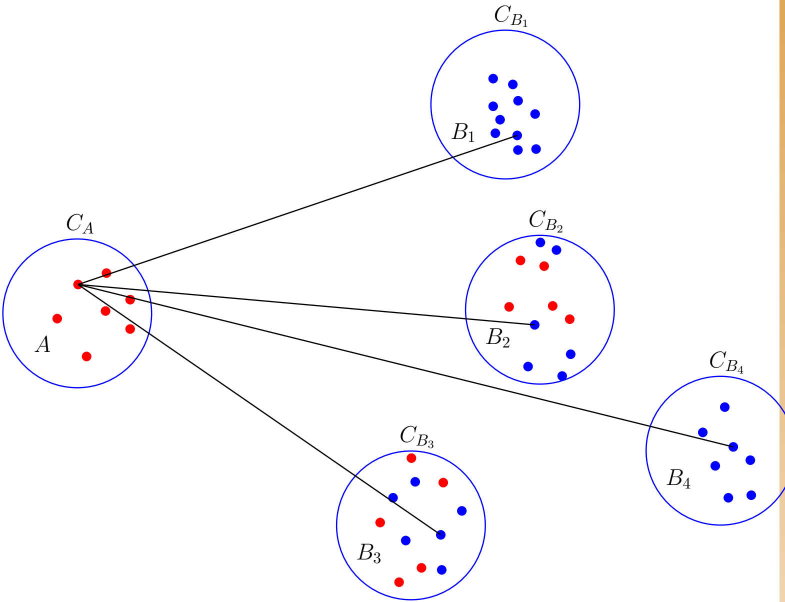
Second attempt

1. compute a WSPD
2. for each pair $\{A, B\}$ do as follows
   - if A is all red

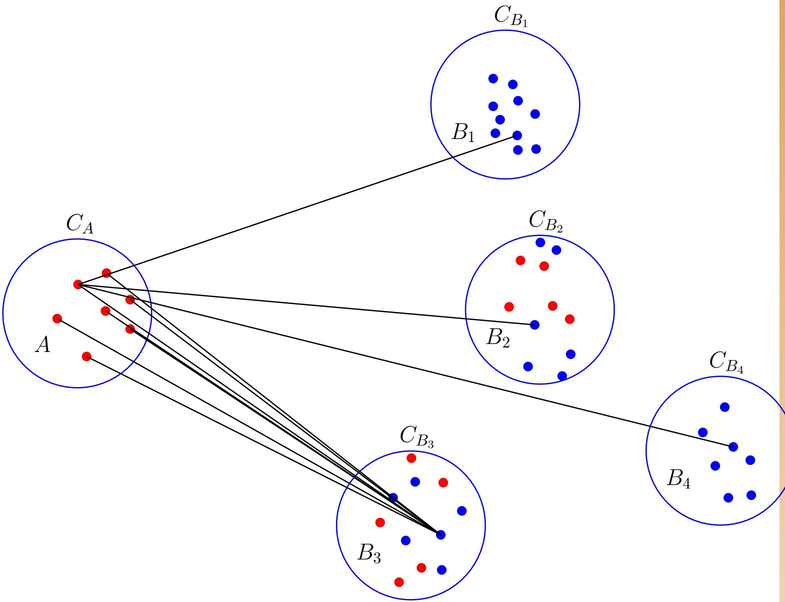
Second attempt



Second attempt

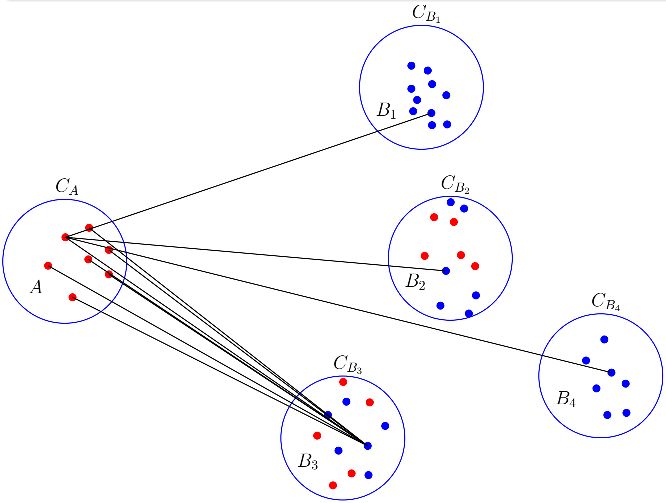


Second attempt

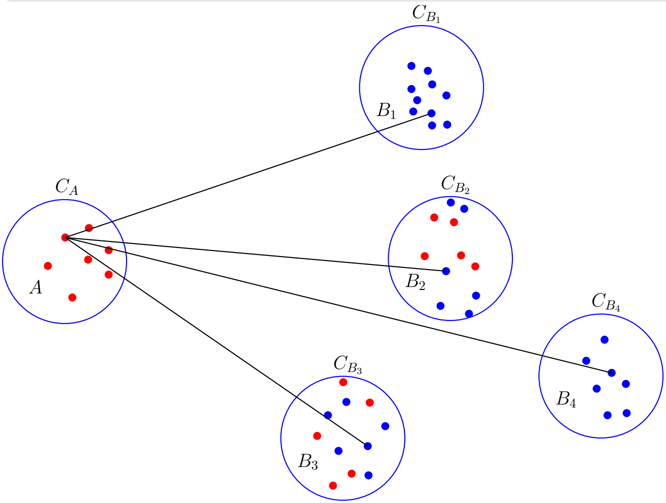


Second attempt

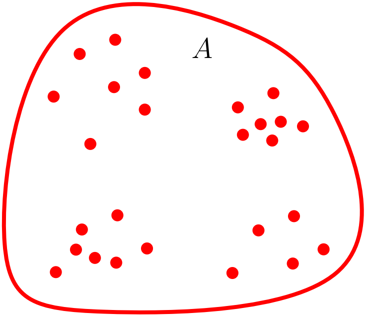
This will lead to quadratic number of edges



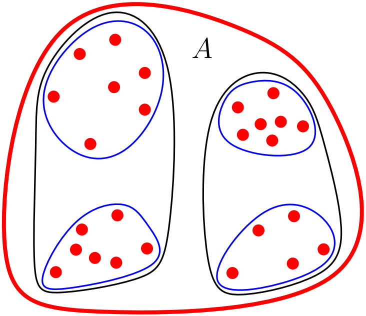
fixing the problem



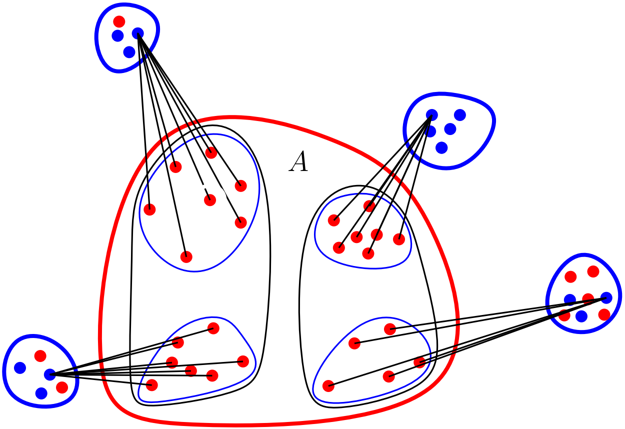
The algorithm



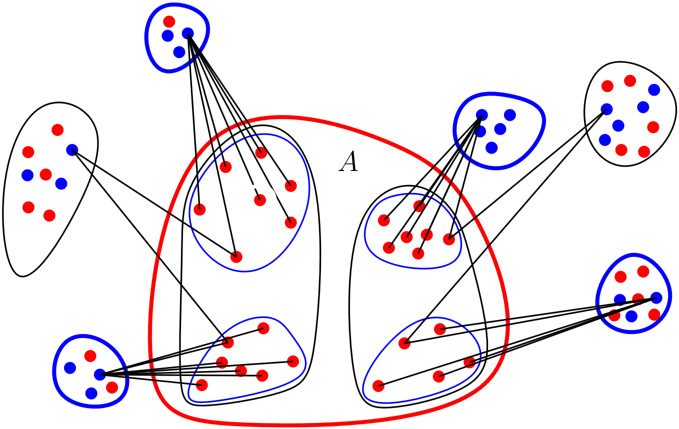
The algorithm



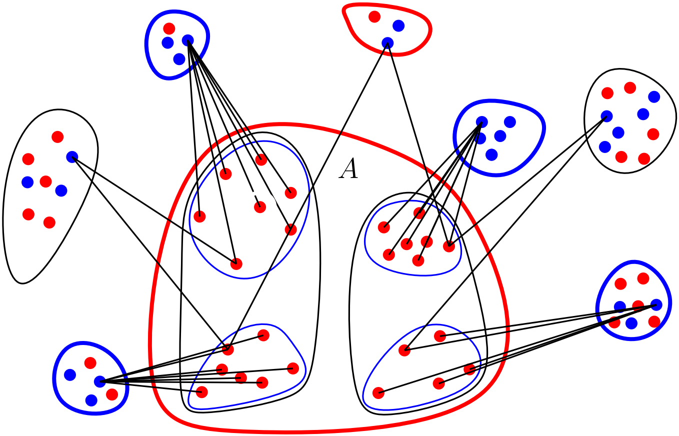
The algorithm



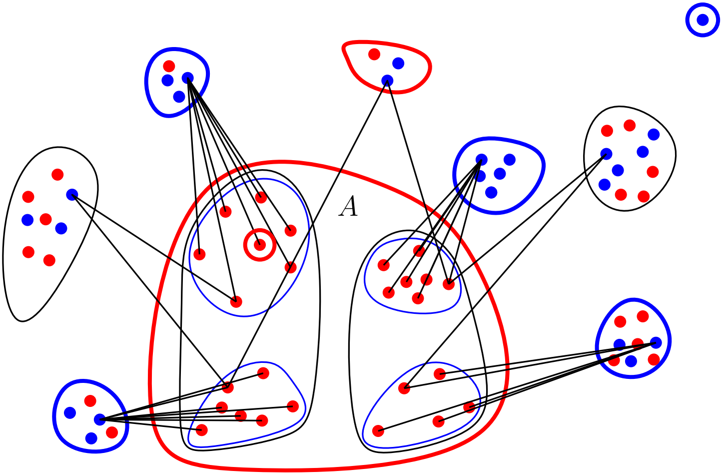
The algorithm



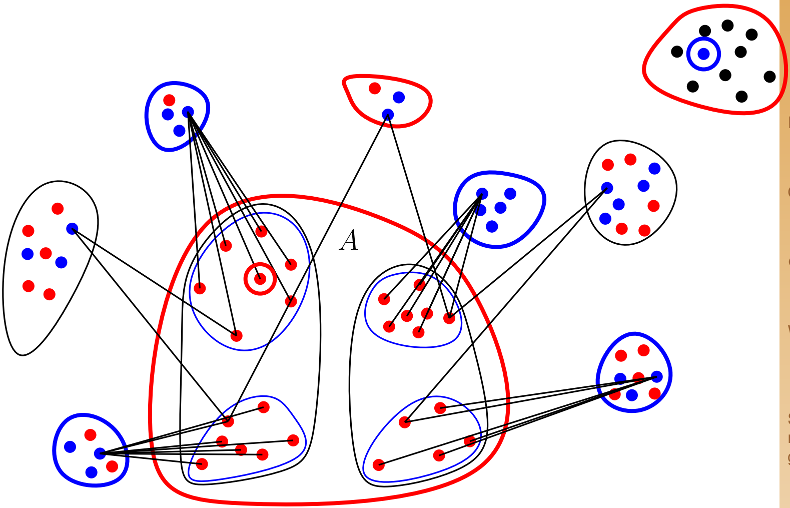
The algorithm



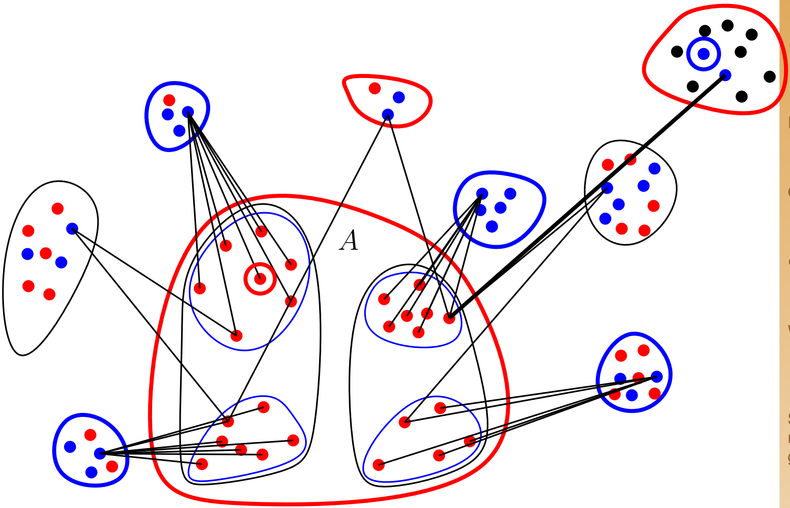
The algorithm



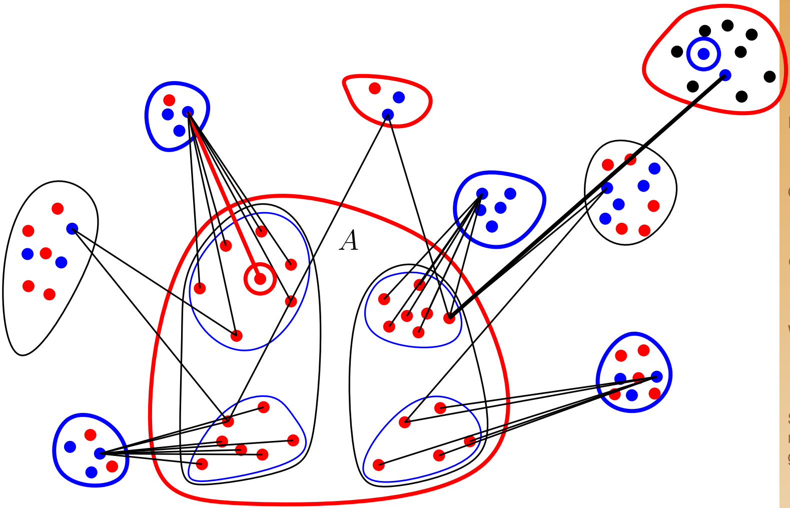
The algorithm



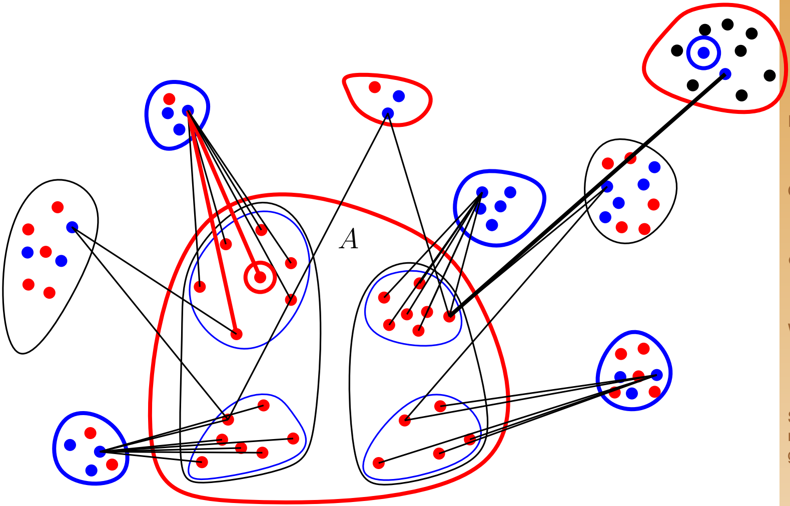
The algorithm



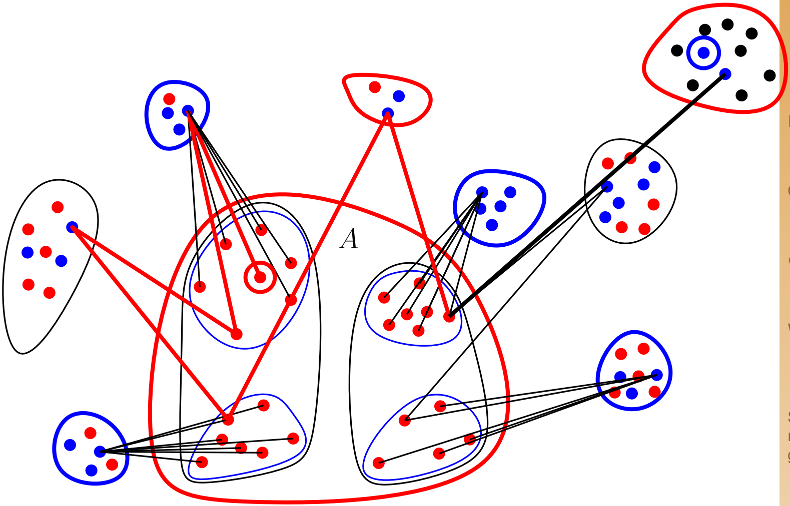
The algorithm



The algorithm



The algorithm



Spanners of complete k-partite geometric graphs

Summary

Given a complete k-partite geometric graph G, compute a spanner that has “small” stretch factor and “few” edges.

Results

Algorithm that computes $(5+\epsilon)$-spanner of G, with $O(n)$ edges in $O(n \log n)$ time.

Spanners of complete k-partite geometric graphs

Summary

Given a complete k-partite geometric graph G, compute a spanner that has “small” stretch factor and “few” edges.

Results

Algorithm that computes $(5 + \epsilon)$-spanner of G, with $O(n)$ edges in $O(n \log n)$ time.

Spanners with small chromatic number

Problem II (the points are not colored)

Given a point set P and an integer $k > 1$, compute a t-spanner for P whose chromatic number is at most k.

Upper and lower bounds

Given an integer $k \geq 2$, compute the smallest value $t(k)$ such that for each set P of points in the plane,

- a $t(k)$-spanner for P exists
- its chromatic number is at most k

Spanners with small chromatic number

Problem II (the points are not colored)

Given a point set P and an integer $k > 1$, compute a t-spanner for P whose chromatic number is at most k.

Upper and lower bounds

Given an integer $k \geq 2$, compute the smallest value $t(k)$ such that for each set P of points in the plane,

- a $t(k)$-spanner for P exists
- its chromatic number is at most k

Spanners with small chromatic number

Problem II (the points are not colored)

Given a point set P and an integer $k > 1$, compute a t-spanner for P whose chromatic number is at most k.

Upper and lower bounds

Given an integer $k \geq 2$, compute the smallest value $t(k)$ such that for each set P of points in the plane,

- a $t(k)$-spanner for P exists
- its chromatic number is at most k

Spanners with small chromatic number

Problem II (the points are not colored)

Given a point set P and an integer $k > 1$, compute a t-spanner for P whose chromatic number is at most k.

Upper and lower bounds

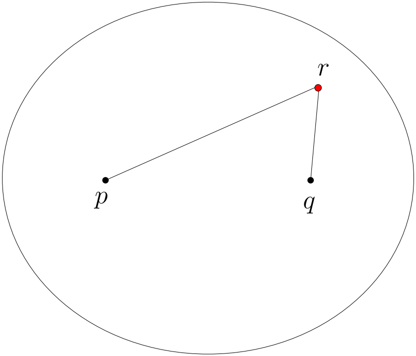
Given an integer $k \geq 2$, compute the smallest value $t(k)$ such that for each set P of points in the plane,

- a $t(k)$-spanner for P exists
- its chromatic number is at most k

The t-ellipse property

A coloring satisfies the t-*ellipse property*

if $\forall p, q \in P$ with $c(p) = c(q)$, there $\exists r \in P$ such that $c(r) \neq c(p)$ and $|pr| + |rq| \leq t|pq|$.



The t-ellipse property

The equivalence

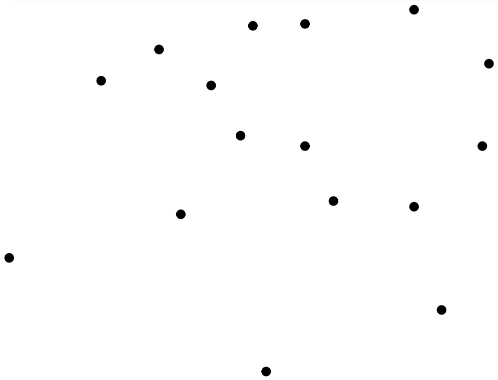
Determining the smallest value of t such that a k-chromatic t-spanner exists for any point set P

$$\equiv$$

Minimizing the value of t such that any point set can be colored using k colors in a way that satisfies the t-ellipse property

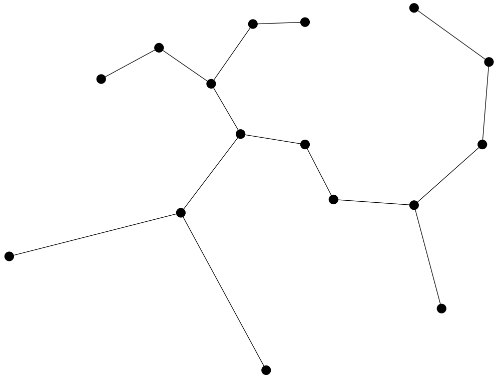
Two colors ($t(k) = 3$)

Input: P, a set of points in the plane



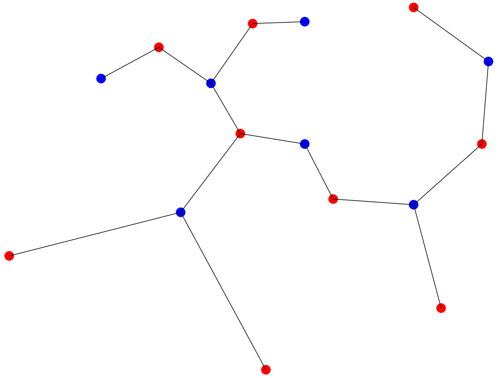
Two colors ($t(k) = 3$)

Compute an Euclidean minimum spanning tree T of P

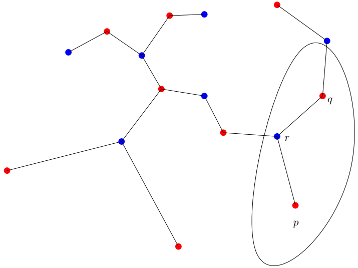


Two colors ($t(k) = 3$)

$c \leftarrow$ a 2-coloring of T



2-coloring with the 3-ellipse property



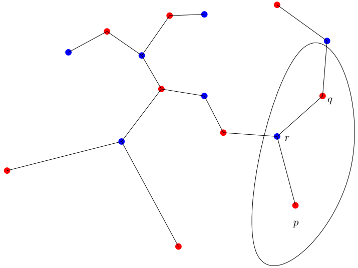
Let r be nearest neighbor of p, i.e. $|pr| \leq |pq|$.

$$Color(r) \neq Color(p)$$

$$|pr| + |rq| \leq |pr| + |rp| + |pq| = 2|pr| + |pq| \leq 2|pq| + |pq| = 3|pq|.$$

Two colors ($t(k) = 3$)

2-coloring with the 3-ellipse property



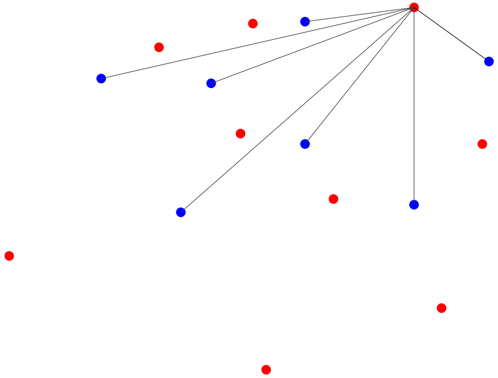
Let r be nearest neighbor of p, i.e. $|pr| \leq |pq|$.

$$Color(r) \neq Color(p)$$

$$|pr| + |rq| \leq |pr| + |rp| + |pq| = 2|pr| + |pq| \leq 2|pq| + |pq| = 3|pq|.$$

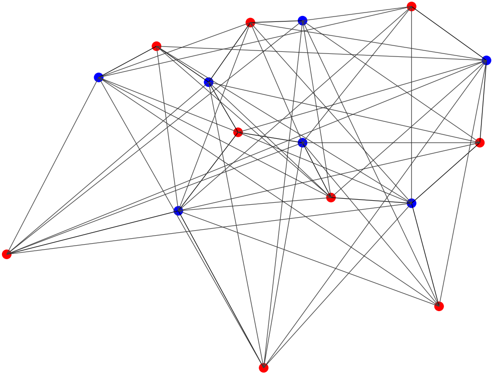
Two colors ($t(k) = 3$)

Draw an edge from each red point to all blue points



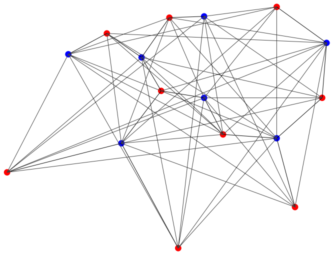
Two colors ($t(k) = 3$)

The complete bipartite graph



Two colors ($t(k) = 3$)

The complete bipartite graph has the 3-Ellipse property



Two colors ($t(k) = 3$)

We obtain a coloring that has the 3-Ellipse property

Theorem by J. Gudmundsson, C. Levcopoulos, G. Narasimhan, and M. Smid, 2002

For any t-spanner for P contains a subgraph with $O(n)$ edges which is a $((1+\epsilon)t)$-spanner for P.

Using this theorem we obtain $(1+\epsilon)3$-spanner with $O(n)$ edges

Two colors ($t(k) = 3$)

We obtain a coloring that has the 3-Ellipse property

Theorem by J. Gudmundsson, C. Levcopoulos, G. Narasimhan, and M. Smid, 2002

For any t-spanner for P contains a subgraph with $O(n)$ edges which is a $((1+\epsilon)t)$-spanner for P.

Using this theorem we obtain $(1+\epsilon)3$-spanner with $O(n)$ edges

Two colors ($t(k) = 3$)

We obtain a coloring that has the 3-Ellipse property

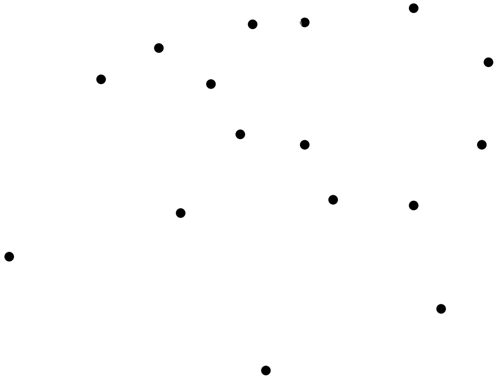
Theorem by J. Gudmundsson, C. Levcopoulos, G. Narasimhan, and M. Smid, 2002

For any t-spanner for P contains a subgraph with $O(n)$ edges which is a $((1+\epsilon)t)$-spanner for P.

Using this theorem we obtain $(1+\epsilon)3$-spanner with $O(n)$ edges

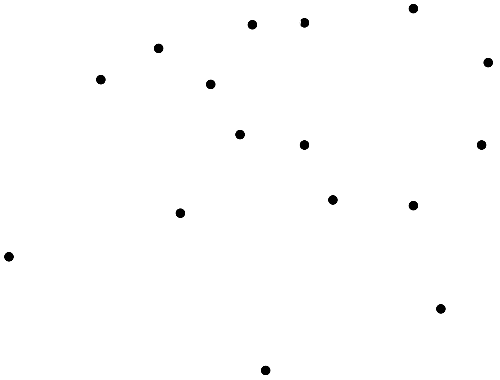
Three colors ($t(k) = 2$)

Input: P, a set of points in the plane



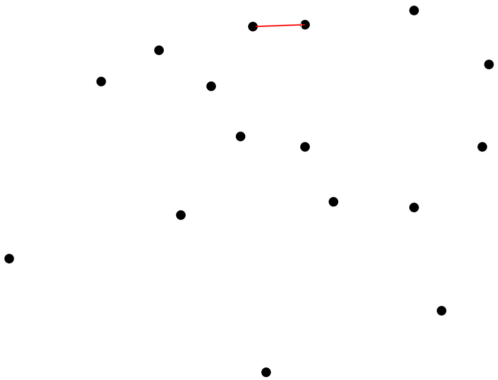
Three colors $(t(k) = 2)$

Sorting the edges of the complete Euclidean graph on P



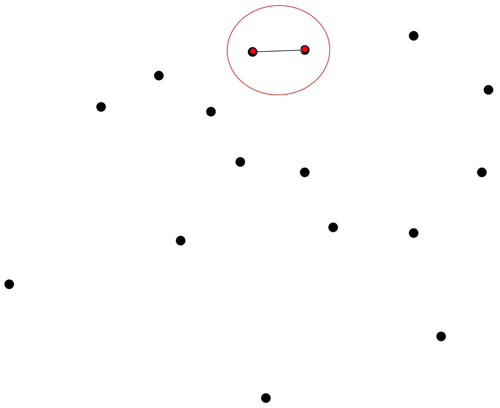
Three colors ($t(k) = 2$)

Pick the shortest edge



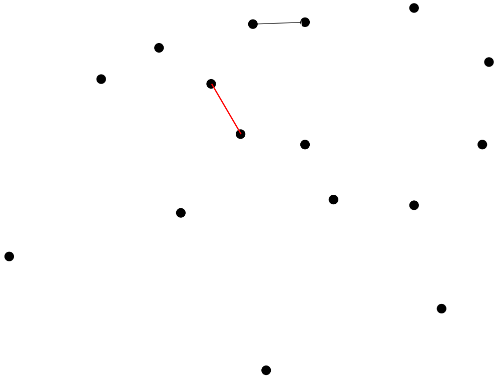
Three colors ($t(k) = 2$)

Add the edge if the 2-ellipse does not contain other edge



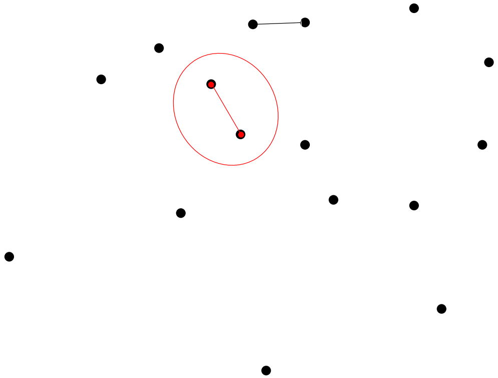
Three colors ($t(k) = 2$)

Pick the next shortest edge



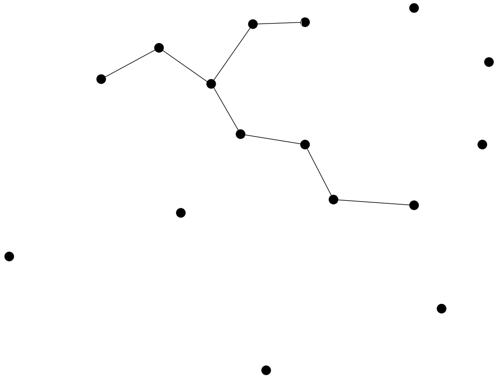
Three colors ($t(k) = 2$)

Add the edge if the 2-ellipse does not contain other edge



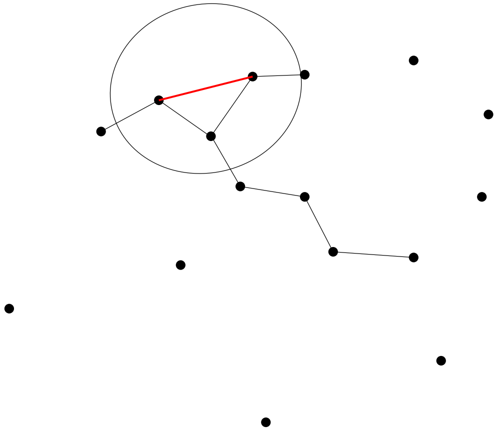
Three colors ($t(k) = 2$)

Pick the next shortest edge



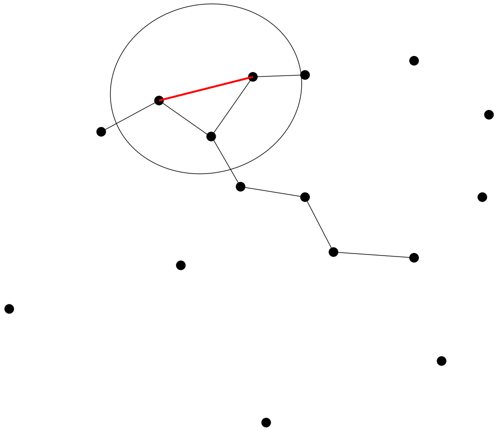
Three colors ($t(k) = 2$)

Pick the next shortest edge



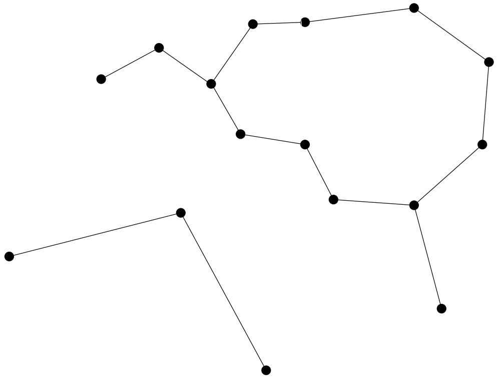
Three colors ($t(k) = 2$)

Add the edge if the 2-ellipse does not contain other edge



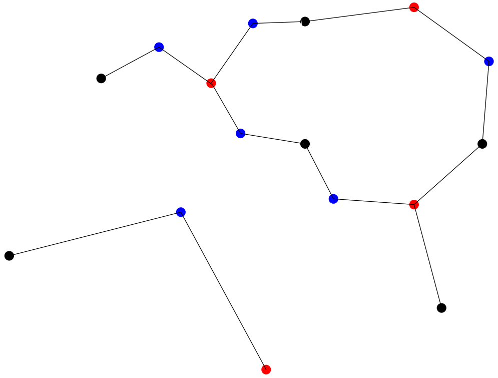
Three colors ($t(k) = 2$)

Planar graph without triangles



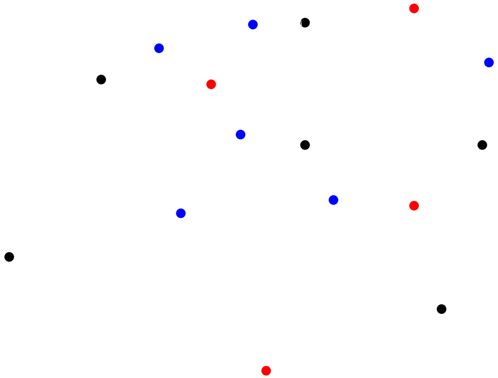
Three colors ($t(k) = 2$)

Planar graph without triangles, thus it is 3-colorable [Grozsch 1959]



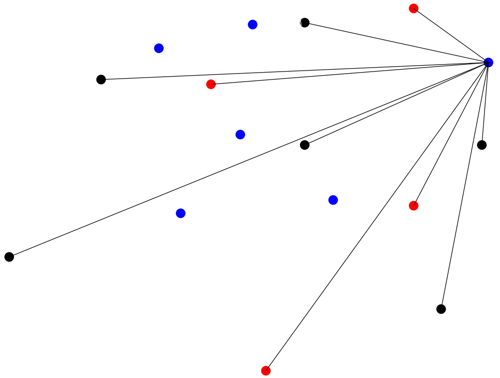
Three colors ($t(k) = 2$)

3-Coloring that has the 2-ellipse property



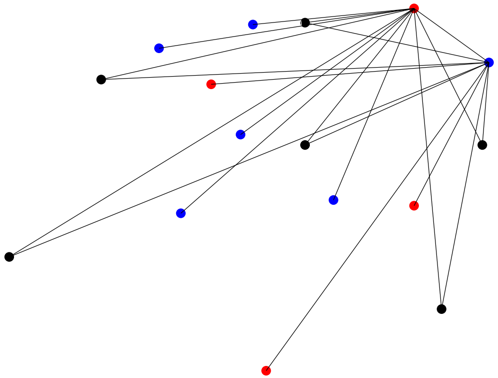
Three colors ($t(k) = 2$)

Complete 3-partite graph



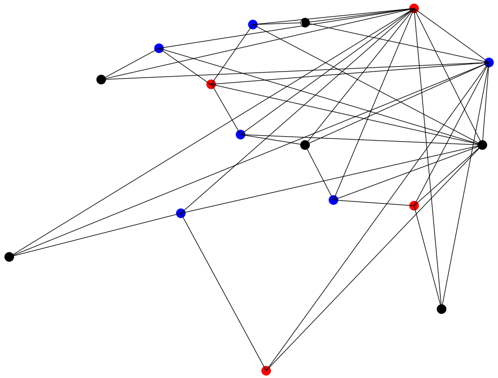
Three colors $(t(k) = 2)$

Complete 3-partite graph



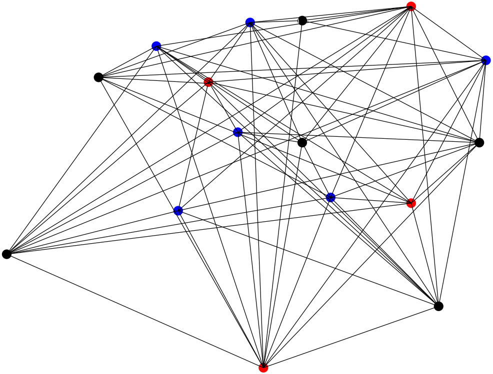
Three colors ($t(k) = 2$)

Complete 3-partite graph



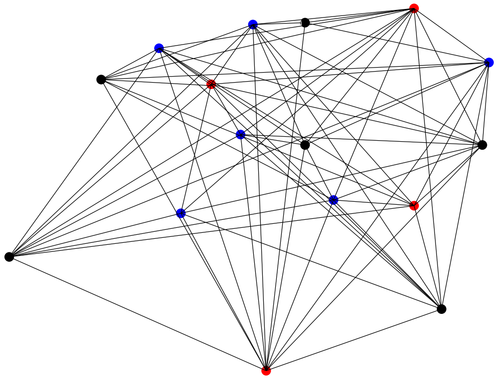
Three colors ($t(k) = 2$)

Complete 3-partite graph

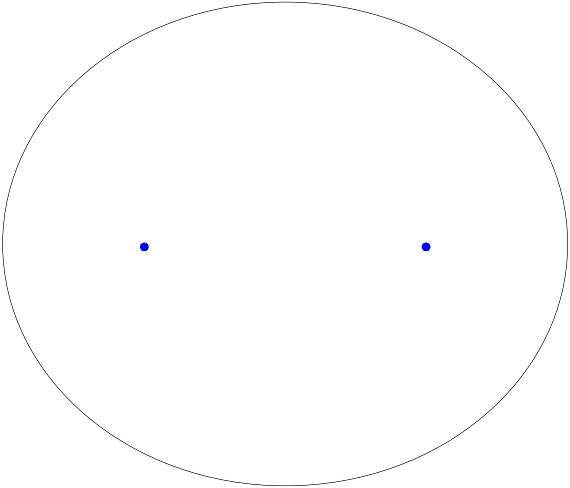


Three colors ($t(k) = 2$)

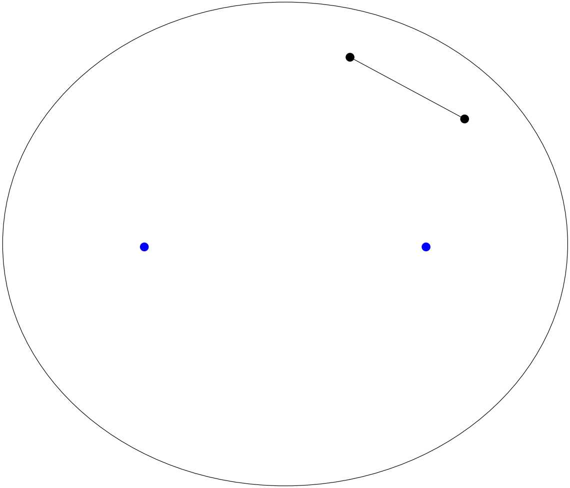
Complete 3-partite graph, with the 2-ellipse property



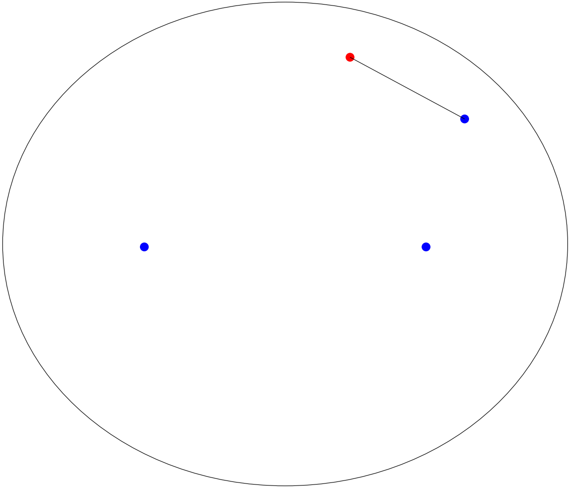
Three colors ($t(k) = 2$)



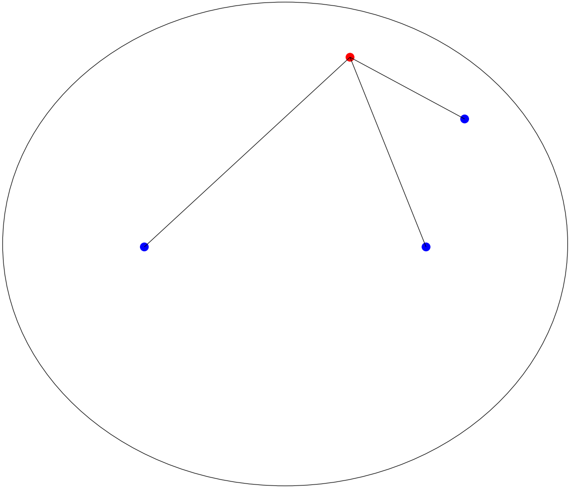
Three colors ($t(k) = 2$)



Three colors ($t(k) = 2$)



Three colors ($t(k) = 2$)



Four colors

Using the Delaunay triangulation we obtain a graph that is:

- planar
- has $(\sqrt{2})$-ellipse property

References

- Book by Narsimhan and Smid on "Geometric Spanners" published in 2008.
- Websites of: Anil Maheshwari/Michiel Smid/Paz Carmi/Prosenjit Bose
  www.cg.carleton.ca
  Carleton University, Ottawa, Canada.

Thank You.