Department of Computer Science, Yazd University

CS Course 18-14-205: Geometric Spanner Networks

The Well-Separated Pair Decomposition

Davood Bakhshesh

Spring 2014

Outline

Outline

Outline

Outline

Outline

Outline

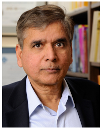
Outline

Outline

Introduction

- The *well-separated pair decomposition*(WSPD) was introduced by Callahan and Kosaraju in 1992.
- WSPD is a data structure that can be used to efficiently solve a large variety of proximity problems
- We will use the WSPD to construct a t-spanner with $O(n)$ edges, for any given set of n points in R^d, and any given constant $t > 1$, in $O(n \log n)$ time.

Paul B. Callahan



S.Rao Kosaraju

Outline

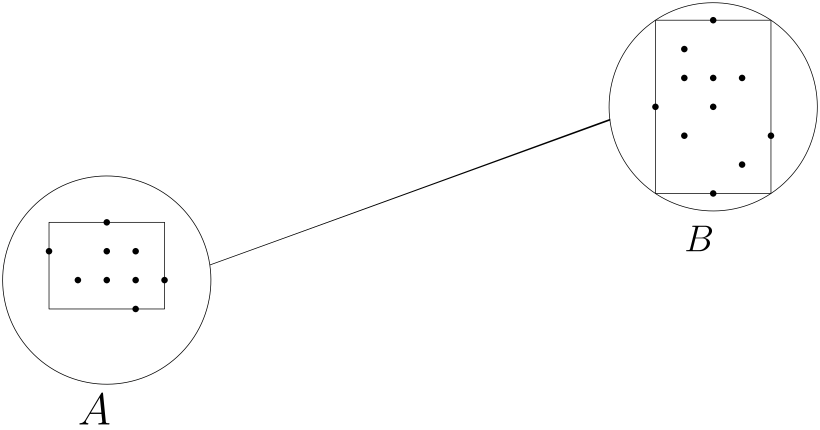
Definition of the well-separated pair decomposition

Definition 9.1.1(Well-Separated Pair)

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

- C_A *and* C_B *have the same radius,*
- C_A *contains the bounding box* $R(A)$ *of* A*,*
- C_B *contains the bounding box* $R(B)$ *of* B
- *the distance between* C_A *and* C_B *is greater than or equal to s times the radius of* C_A*.*

Definition of the well-separated pair decomposition



Definition of the well-separated pair decomposition

Question

Is it possible we test in $\mathcal{O}(1)$ time whether A and B are well-separated with respect to s?

Definition of the well-separated pair decomposition

Lemma 9.1.2.

Let $s > 0$ be a real number, let A and B be two finite sets of points that are well-separated with respect to s, let p and p' be any two points in A, and let q and q' be any two points in B. Then

- $|pp'| \leq (2/s)|pq|$
- $|p'q'| \leq (1 + 4/s)|pq|$

Definition of the well-separated pair decomposition

Proof of Lemma 9.1.2.

Let C_A and C_B are two disjoint balls that contain the points of A and B, respectively, that have the same radius, say ρ, and whose distance is greater than or equal to $s\rho$. Thus, we have

$$\begin{cases} |pp'| \leq 2\rho \\ |pq| \geq s\rho \end{cases} \Rightarrow |pp'| \leq (2/s)|pq|.$$

By symmetric argument we have $|qq'| \leq (2/s)|pq|$. By combining these inequalities and applying the triangle inequality we get

$$|p'q'| \leq |p'p|+|pq|+|qq'| \leq (2/s)|pq|+|pq|+(2/s)|pq| = (1+4/s)|pq|. \blacksquare$$

Definition of the well-separated pair decomposition

Approximating a well-separated

The second inequality in Lemma 9.1.2 implies that the distance between an arbitrary point p in A and an arbitrary point q in B approximates all the $|A|.|B|$ distances between the pairs in the Cartesian product $A \times B$.

Definition of the well-separated pair decomposition

Definition 9.1.3

(**Well-Separated Pair Decomposition.**) Let S be a set of n points in $\mathbb{R}^d$, and let $s > 0$ be a real number. A well-separated pair decomposition (WSPD) for S, with respect to s, is a sequence

$$\{A_1, B_1\}, \{A_2, B_2\}, \ldots, \{A_m, B_m\}$$

of pairs of nonempty subsets of S, for some integer m, such that

- for each i with $1 \leq i \leq m$, A_i and B_i are well-separated with respect to s.
- for any two distinct points p and q of S, there is exactly one index i with $1 \leq i \leq m$, such that
  1. $p \in A_i$ and $q \in B_i$, or
  2. $p \in B_i$ and $q \in A_i$.

The integer m is called the *size* of the WSPD.

Definition of the well-separated pair decomposition

Question

Does a WSPD exist for any set S?

Answer

Yes. We can consider WSPD for S as follows

$$\{\{p_i\}, \{q_i\}\} \quad \forall \text{ distinct points } p_i \text{ and } q_i \text{ of } S$$

WSPD of size $O(n)$

The main result of this chapter will be an algorithm that constructs, in $O(n \log n)$ time, a WSPD of size $O(n)$, for any set S of n points in $\mathbb{R}^d$, and for any constant separation ratio $s > 0$.

Definition of the well-separated pair decomposition

Question

Does a WSPD exist for any set S?

Answer

Yes. We can consider WSPD for S as follows

$$\{\{p_i\}, \{q_i\}\} \quad \forall \text{ distinct points } p_i \text{ and } q_i \text{ of } S$$

WSPD of size $O(n)$

The main result of this chapter will be an algorithm that constructs, in $O(n \log n)$ time, a WSPD of size $O(n)$, for any set S of n points in $\mathbb{R}^d$, and for any constant separation ratio $s > 0$.

Definition of the well-separated pair decomposition

Question

Does a WSPD exist for any set S?

Answer

Yes. We can consider WSPD for S as follows

$$\{\{p_i\}, \{q_i\}\} \quad \forall \text{ distinct points } p_i \text{ and } q_i \text{ of } S$$

WSPD of size $O(n)$

The main result of this chapter will be an algorithm that constructs, in $O(n \log n)$ time, a WSPD of size $O(n)$, for any set S of n points in $\mathbb{R}^d$, and for any constant separation ratio $s > 0$.

Outline

Spanners Based on the WSPD

Basic Spanner Construction

1. Construct a well-separated pair decomposition with separation ratio $s > 4$ for points set S.
2. Take one arbitrary edge for each pair of the decomposition.

This results in a t-spanner with $t = \frac{s+4}{s-4}$.

Spanners Based on the WSPD

Question

Why the above construction results in a t-spanner with $t = \frac{s+4}{s-4}$?

Answer by induction on Euclidean distance p, q

Step1. Suppose distinct points p, q are closest pair.

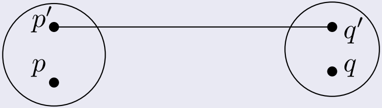
Spanners Based on the WSPD

Question

Why the above construction results in a t-spanner with $t = \frac{s+4}{s-4}$?

Answer by induction on Euclidean distance p, q

Step1. Suppose distinct points p, q are closest pair.



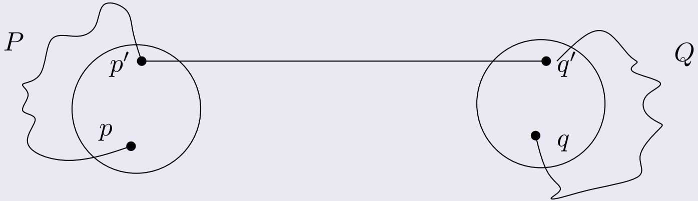
$$\frac{2}{s} = \frac{1}{2} - \frac{1}{t+1} < \frac{1}{2}$$

$$|pp'| \leq (2/s)|pq| < \frac{|pq|}{2} < |pq|$$

Contradiction

Spanners Based on the WSPD

Step2.



t-path between p, q is $L = P + \{p', q'\} + Q$

$$|L| = |P| + |\{p', q'\}| + |Q| \leq t|pp'| + |p'q'| + t|qq'|$$

$$\leq (2/s)|pq| + (1 + 4/s)|pq| + (2/s)|pq|$$

$$= \left(\frac{4(t+1)}{s} + 1\right)|pq| = t|pq|$$

Outline

The Split Tree

Question

How can we find a WSPD of size $O(n)$ for a set of n points with respect to $s > 0$?

The stages of the algorithms

1. In the first stage, a binary tree, called the *split tree*, is constructed. This tree does not depend on s.
2. In the second stage, the split tree is used to construct the WSPD itself.

The Split Tree

Question

How can we find a WSPD of size $O(n)$ for a set of n points with respect to $s > 0$?

The stages of the algorithms

1. In the first stage, a binary tree, called the *split tree*, is constructed. This tree does not depend on s.
2. In the second stage, the split tree is used to construct the WSPD itself.

The Split Tree

Hyperrectangle

A *hyperrectangle* R, or a d-dimensional axes-parallel hyperrectangle, is the Cartesian product of d closed intervals as follows

$$R = [l_1, r_1] \times [l_2, r_2] \times \ldots \times [l_d, r_d],$$

where l_i and r_i are real numbers with $l_i \leq r_i, 1 \leq i \leq d$.
$L_i(R) := r_i - l_i$ is side length of R along the i-th dimension.
$L_{max}(R)$ and $L_{min}(R)$ are as the maximum and minimum side lengths of R along any dimension, respectively.
Let j be the index such that $L_{max}(R) = L_j(R)$. We define

$$h(R) := (l_j + r_j)/2$$

as the center of the largest interval of R.

The Split Tree

Hyperrectangle

A *hyperrectangle* R, or a d-dimensional axes-parallel hyperrectangle, is the Cartesian product of d closed intervals as follows

$$R = [l_1, r_1] \times [l_2, r_2] \times \ldots \times [l_d, r_d],$$

where l_i and r_i are real numbers with $l_i \leq r_i, 1 \leq i \leq d$.
$L_i(R) := r_i - l_i$ is side length of R along the i-th dimension.
$L_{max}(R)$ and $L_{min}(R)$ are as the maximum and minimum side lengths of R along any dimension, respectively.
Let j be the index such that $L_{max}(R) = L_j(R)$. We define

$$h(R) := (l_j + r_j)/2$$

as the center of the largest interval of R.

The Split Tree

Hyperrectangle

A *hyperrectangle* R, or a d-dimensional axes-parallel hyperrectangle, is the Cartesian product of d closed intervals as follows

$$R = [l_1, r_1] \times [l_2, r_2] \times \ldots \times [l_d, r_d],$$

where l_i and r_i are real numbers with $l_i \leq r_i, 1 \leq i \leq d$.
$L_i(R) := r_i - l_i$ is side length of R along the i-th dimension.
$L_{max}(R)$ and $L_{min}(R)$ are as the maximum and minimum side lengths of R along any dimension, respectively.
Let j be the index such that $L_{max}(R) = L_j(R)$. We define

$$h(R) := (l_j + r_j)/2$$

as the center of the largest interval of R.

The Split Tree

Hyperrectangle

A *hyperrectangle* R, or a d-dimensional axes-parallel hyperrectangle, is the Cartesian product of d closed intervals as follows

$$R = [l_1, r_1] \times [l_2, r_2] \times \ldots \times [l_d, r_d],$$

where l_i and r_i are real numbers with $l_i \leq r_i, 1 \leq i \leq d$.
$L_i(R) := r_i - l_i$ is side length of R along the i-th dimension.
$L_{max}(R)$ and $L_{min}(R)$ are as the maximum and minimum side lengths of R along any dimension, respectively.
Let j be the index such that $L_{max}(R) = L_j(R)$. We define

$$h(R) := (l_j + r_j)/2$$

as the center of the largest interval of R.

The Split Tree

If S consists of only one point, then the split tree consists of one single node that stores that point.

If $|S| \geq 2$. Split $R(S)$ into two hyperrectangles by cutting its longest interval into two equal parts. Let S_1 and S_2 be the subsets of S that are contained in these two new hyperrectangles. The split tree for S consists of a root having two subtrees, which are recursively defined split trees for S_1 and S_2

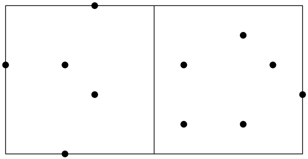
The Split Tree

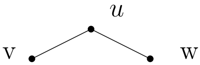
If S consists of only one point, then the split tree consists of one single node that stores that point.

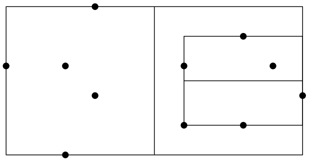
If $|S| \geq 2$. Split $R(S)$ into two hyperrectangles by cutting its longest interval into two equal parts. Let S_1 and S_2 be the subsets of S that are contained in these two new hyperrectangles. The split tree for S consists of a root having two subtrees, which are recursively defined split trees for S_1 and S_2

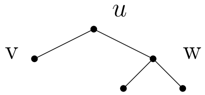
The Split Tree

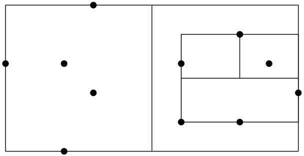
An Example of the Split Tree

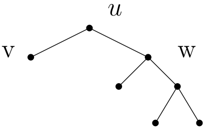






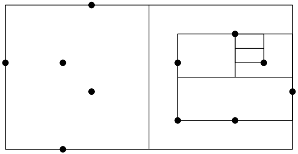

u
v
w

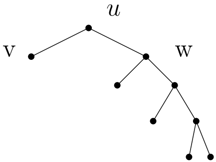


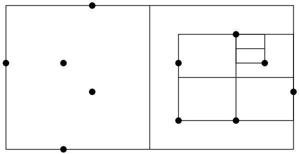

u
v
w

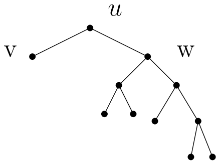
The Split Tree

An Example of the Split Tree

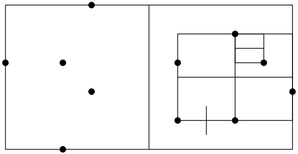


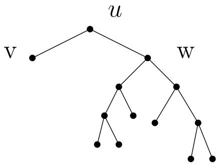




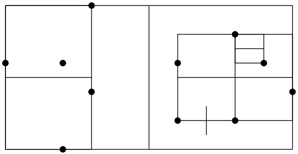

u
V
W

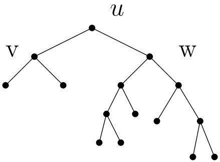
The Split Tree
An Example of the Split Tree



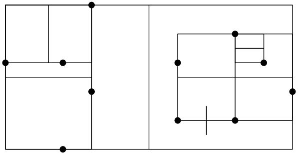


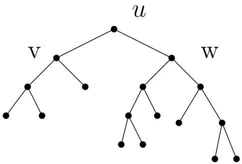
The Split Tree
An Example of the Split Tree



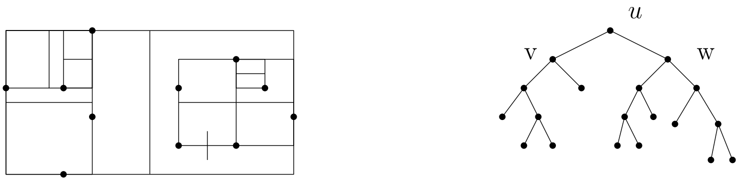


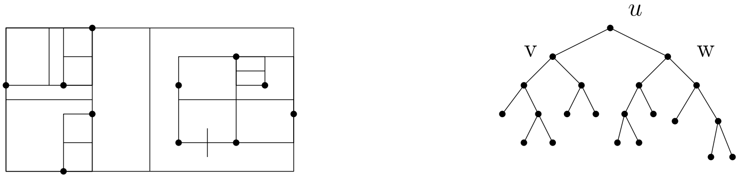
The Split Tree
An Example of the Split Tree





The Split Tree
An Example of the Split Tree



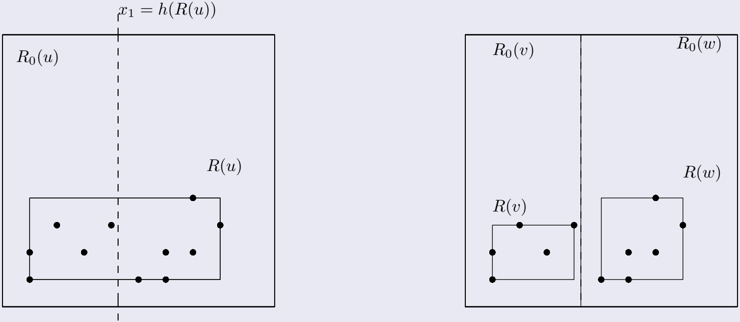

u
V
W

The Split Tree

Bounding Box and Hyperrectangle

$R(u) :=$ The smallest hyperrectangle that contains the points stored in the subtree rooted at u.

$R_0(u)$:=A hyperrectangle that contains $R(u)$



Algorithm of Split Tree

Algorithm SPLITTREE(S, R)

1. **if** $|S| = 1$
2. **then** creat a new node u;
3. $R(u) := R(S)$;
4. $R_0(u) := R$;
5. store with u the only point of S, and the two hyperrectangles $R(u)$ and $R_0(u)$, and set its two children pointers to be nil;
6. return node u
7. **else** compute the bounding box $R(S)$ of S;
8. compute i such that $L_{max}(R(S)) = L_i(R(S))$;
9. let H be the hyperplane with equation $x_i = h(R(S))$;
10. using H, split R into two hyperrectangles R_1 and R_2;
11. $S_1 := S \cap R_1$;
12. $S_2 := S \setminus S_1$;
13. $v :=$ SPLITTREE(S_1, R_1);
14. $w :=$ SPLITTREE(S_2, R_2);
15. create a new node u;
16. $R(u) := R(S)$;
17. $R_0(u) := R$;
18. store with u the two hyperrectangles $R(u)$ and $R_0(u)$, and set its left and right child pointers to v and w, respectively;
19. return node u

The Split Tree

Question

What is the worst-case time complexity of the SPLITTREE algorithm in a direct implementation?

Answer: $\Theta(n^2)$

Answer: Yes. In Section 9.3.2, we will give an improved algorithm that constructs the split tree in $O(n \log n)$ time.

The Split Tree

Question

What is the worst-case time complexity of the SPLITTREE algorithm in a direct implementation?
Answer: $\Theta(n^2)$

Can we improve the above time complexity?
Answer: Yes. In Section 9.3.2, we will give an improved algorithm that constructs the split tree in $\mathcal{O}(n \log n)$ time.

The Split Tree

Question

What is the worst-case time complexity of the SPLITTREE algorithm in a direct implementation?
Answer: $\Theta(n^2)$

Can we improve the above time complexity?
Answer: Yes. In Section 9.3.2, we will give an improved algorithm that constructs the split tree in $\mathcal{O}(n \log n)$ time.

The Split Tree

Lemma 9.3.1

Let R be a hypercube that contains the bounding box $R(S)$ of the set S and that has sides of length $L_{max}(R(S))$. Let T be the tree that is computed by algorithm SPLITTREE(S,R), and let u be any node of T. If u is not the root of T, then

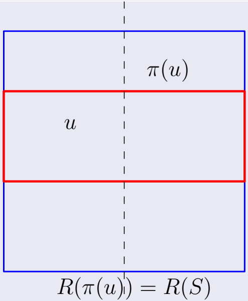
$$L_{min}(R_0(u)) \geq \frac{1}{2} \cdot L_{max}(R(\pi(u))),$$

where $\pi(u)$ is the parent u.

The Split Tree

Proof by induction on the distance between u and the root

Step 1. Assume $\pi(u)$ is the root.



$$L_{max}(R(\pi(u))) = side\_length(R)$$

It follows from the algorithm that
$L_{min}(R_0(u)) = \frac{1}{2} side\_length(R) = \frac{1}{2} L_{max}(R(\pi(u)))$

The Split Tree

Proof by induction on the distance between u and the root

Step 2. Assume $\pi(u)$ is not the root. By the induction hypothesis we have

$$L_{min}(R_0(\pi(u))) \geq \frac{1}{2} \cdot L_{max}(R(\pi(\pi(u)))).$$

We distinguish two cases:

Case 1: $L_{min}(R_0(u)) = L_{min}(R_0(\pi(u)))$.

Case 2: $L_{min}(R_0(u)) \neq L_{min}(R_0(\pi(u)))$.

The Split Tree

Proof by induction on the distance between u and the root

Step 2. Assume $\pi(u)$ is not the root. By the induction hypothesis we have

$$L_{min}(R_0(\pi(u))) \geq \frac{1}{2} \cdot L_{max}(R(\pi(\pi(u)))).$$

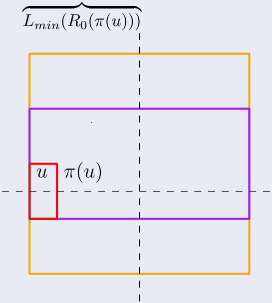
We distinguish two cases:

Case 1: $L_{min}(R_0(u)) = L_{min}(R_0(\pi(u)))$.

Case 2: $L_{min}(R_0(u)) \neq L_{min}(R_0(\pi(u)))$.

The Split Tree

Case 1: $L_{min}(R_0(u)) = L_{min}(R_0(\pi(u)))$.



$$R(\pi(u)) \subseteq R(\pi(\pi(u))) \rightarrow L_{max}(R(\pi(\pi(u)))) \geq L_{max}(R(\pi(u)))$$

Then
$$\begin{aligned} L_{min}(R_0(u)) &= L_{min}(R_0(\pi(u))) \\ &\geq \tfrac{1}{2} \cdot L_{max}(R(\pi(\pi(u)))) \\ &\geq \tfrac{1}{2} \cdot L_{max}(R(\pi(u))) \end{aligned}$$

The Split Tree

Case 2: $L_{min}(R_0(u)) \neq L_{min}(R_0(\pi(u)))$.

We must have

$$L_{min}(R_0(u)) < L_{min}(R_0(\pi(u))).$$

Let i be the index such that $L_{max}(R(\pi(u))) = L_i(R(\pi(u)))$. We can prove that

$$L_i(R_0(u)) = L_{min}(R_0(u)).$$

Using the above claim and the fact $L_i(R_0(u)) \geq \frac{1}{2} \cdot L_i(R(\pi(u)))$, we have

$$L_{min}(R_0(u)) = L_i(R_0(u)) \geq \frac{1}{2} \cdot L_i(R(\pi(u))) = \frac{1}{2} \cdot L_{max}(R(\pi(u))).$$

The Split Tree

Question

How can we construct the split tree in $O(n \log n)$ time?

Answer

We can construct the split tree in $O(n \log n)$ using the *partial split tree*. Here, we will show how we can construct the partial split tree and how we can construct the split tree using the partial split tree.

The Split Tree

Question

How can we construct the split tree in $O(n \log n)$ time?

Answer

We can construct the split tree in $O(n \log n)$ using the *partial split tree*. Here, we will show how we can construct the partial split tree and how we can construct the split tree using the partial split tree.

Partial Split Tree

A *partial split tree* is defined in the same way as the split tree, except that subsets represented by the leaves may have size larger that 1.

Observation

Clearly, the main problem is to compute, in $O(n)$ time, a partial split tree, such that each leaf corresponds to a subset of size at most $n/2$.

Partial Split Tree

A *partial split tree* is defined in the same way as the split tree, except that subsets represented by the leaves may have size larger that 1.

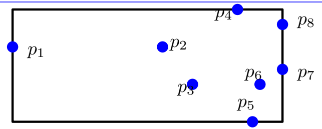
Observation

Clearly, the main problem is to compute, in $O(n)$ time, a partial split tree, such that each leaf corresponds to a subset of size at most $n/2$.

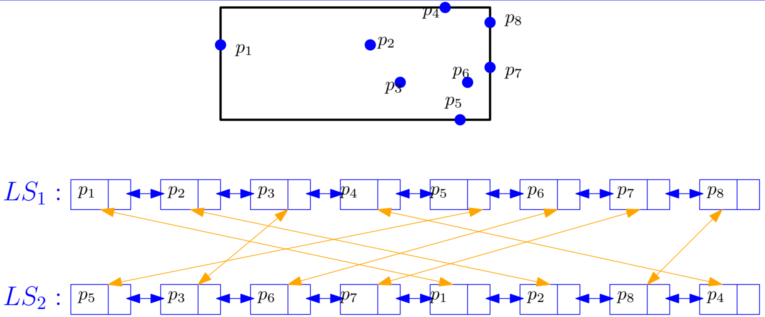
Algorithm $\textsc{PartialSplitTree}(S, R, (LS_i)_{1\leq i\leq d})$

This algorithm constructs a partial split tree for set S in $O(n)$ time. We state the algorithm by an example.

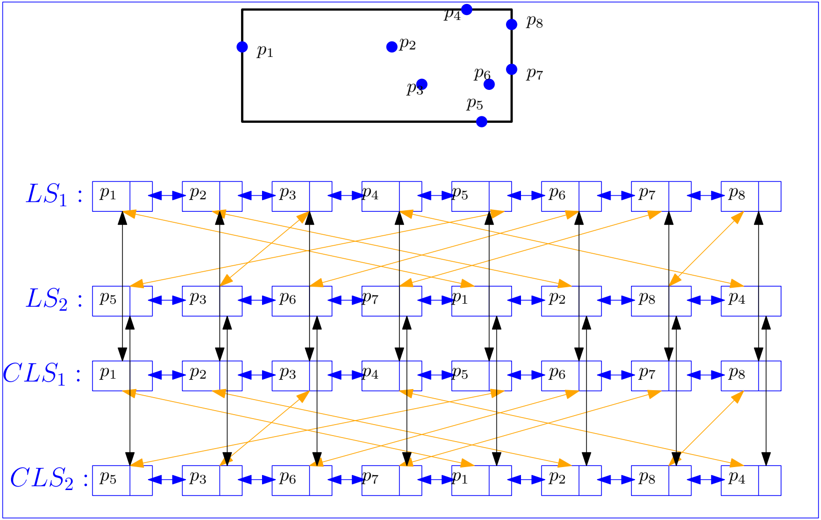
The Split Tree
Partial Split tree



The Split Tree
Partial Split tree

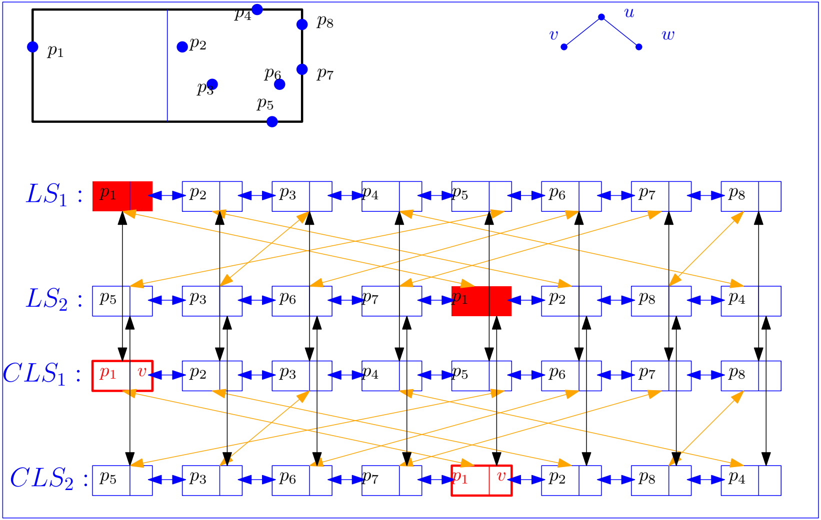


The Split Tree
Partial Split tree



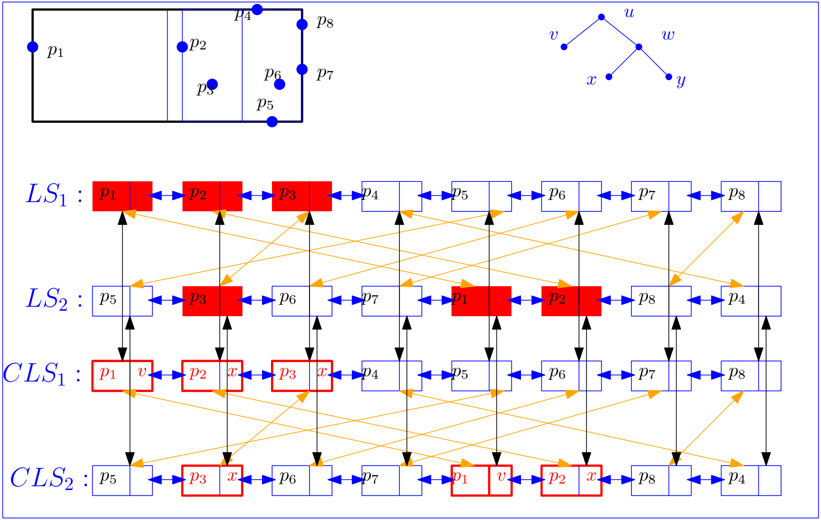
The Split Tree

Partial Split tree



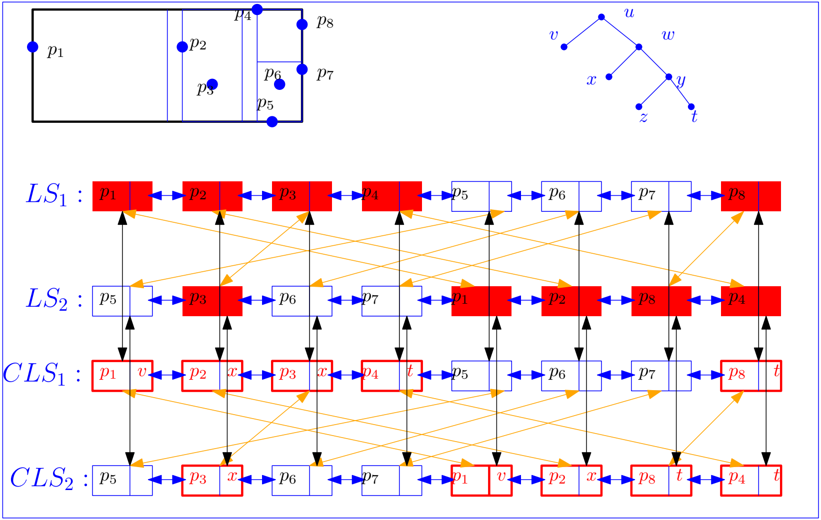
The Split Tree

Partial Split tree

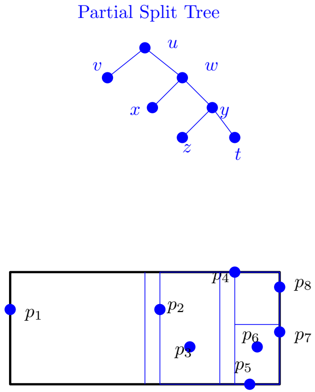


The Split Tree

Partial Split tree



The Split Tree
Partial Split tree



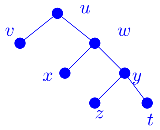
The Split Tree

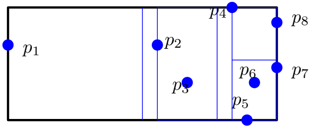
Partial Split tree

LS_1^v : p_1

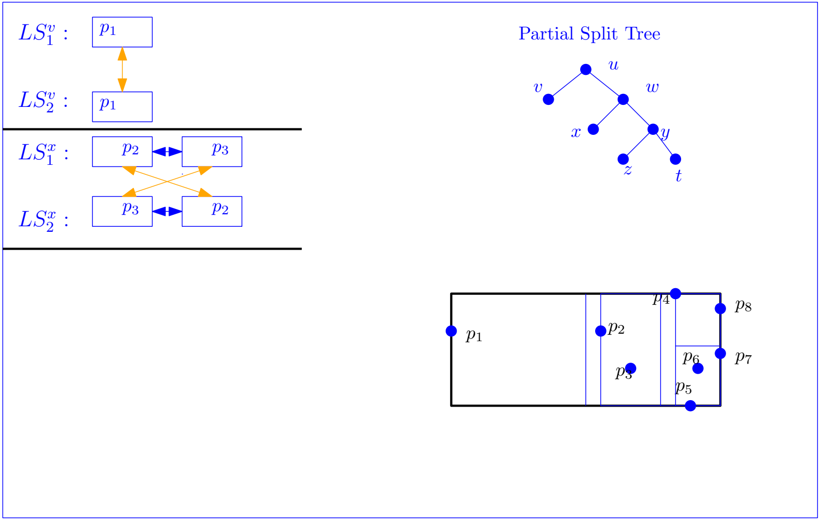
LS_2^v : p_1

Partial Split Tree

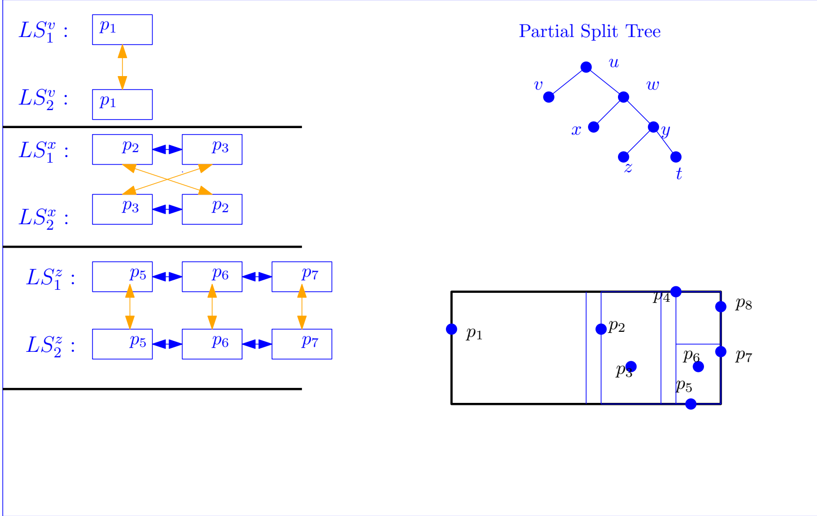




The Split Tree
Partial Split tree

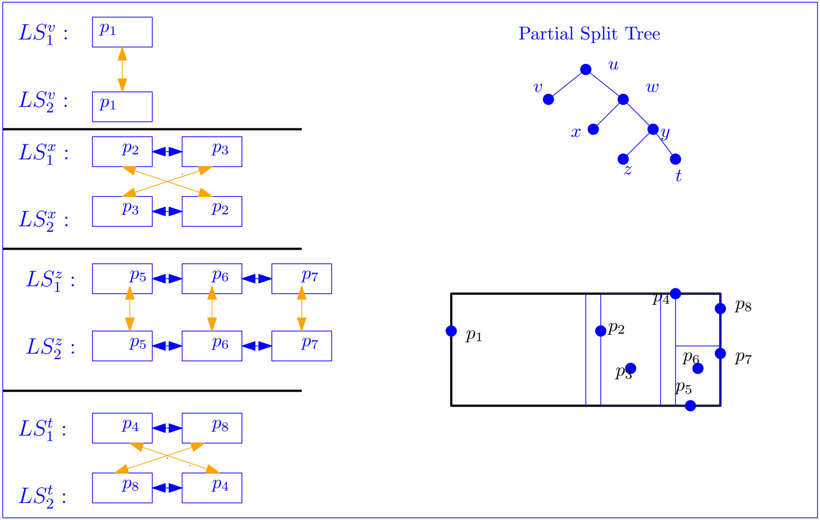


The Split Tree
Partial Split tree



The Split Tree

Partial Split tree



Lemma 9.3.2

Algorithm PartialSplitTree$(S, R, (LS_i)_{1 \leq i \leq d})$ computes a partial split tree T that satisfies following conditions

1. Each leaf u of T corresponds to a subset S_u of size at most $n/2$.
2. Each node u of T stores two hyperrectangles $R(u)$ and $R_0(u)$, which are the same hyperrectangles as in algorithm SplitTree.
3. Each leaf u of T stores the following additional information.
   - A collection of doubly-linked lists $LS_i^u, 1 \leq i \leq d$, where LS_i^u contains the points of S_u, sorted in nondecreasing order of their i-th coordinates.
   - The d lists LS_i^u are connected by cross-pointers.

The running time of the algorithm is $O(n)$.

Question

How can we construct the split tree using the partial split tree?

Answer

Run algorithm PARTIALSPLITTREE$(S, R, (LS_i)_{1 \leq i \leq d})$, that computes, in $O(n)$ time, a partial split tree. Then, for each leaf u of this tree, recursively continue this process. (Observe that in recursive calls, preprocessing is not necessary.)

Theorem 9.3.3

Let S be a set of n points in $\mathbb{R}^d$. The split tree for S can be computed in $O(n \log n)$ time.

Question

How can we construct the split tree using the partial split tree?

Answer

Run algorithm PARTIALSPLITTREE$(S, R, (LS_i)_{1 \le i \le d})$, that computes, in $O(n)$ time, a partial split tree. Then, for each leaf u of this tree, recursively continue this process. (Observe that in recursive calls, preprocessing is not necessary.)

Theorem 9.3.3

Let S be a set of n points in $\mathbb{R}^d$. The split tree for S can be computed in $O(n \log n)$ time.

Question

How can we construct the split tree using the partial split tree?

Answer

Run algorithm $\textsc{PartialSplitTree}(S, R, (LS_i)_{1 \leq i \leq d})$, that computes, in $O(n)$ time, a partial split tree. Then, for each leaf u of this tree, recursively continue this process. (Observe that in recursive calls, preprocessing is not necessary.)

Theorem 9.3.3

Let S be a set of n points in $\mathbb{R}^d$. The split tree for S can be computed in $O(n \log n)$ time.

Outline

Computing the Well-Separated Pair Decomposition

Algorithm COMPUTEWSPD(T, s)
Input: Split Tree T for the point set S and a real number $s > 0$.
Output: WSPD for S.

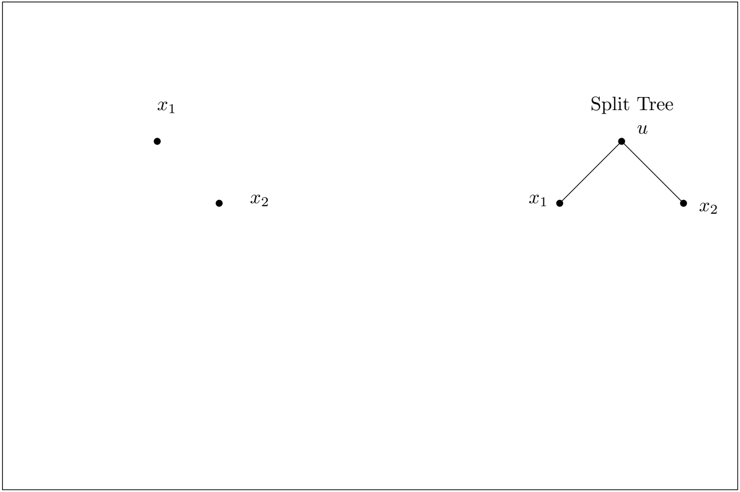
1. **for each** internal node u of T
2. v :=left child of u;
3. w :=right child of u;
4. FINDPAIRS(v,w);
5.

Computing the Well-Separated Pair Decomposition

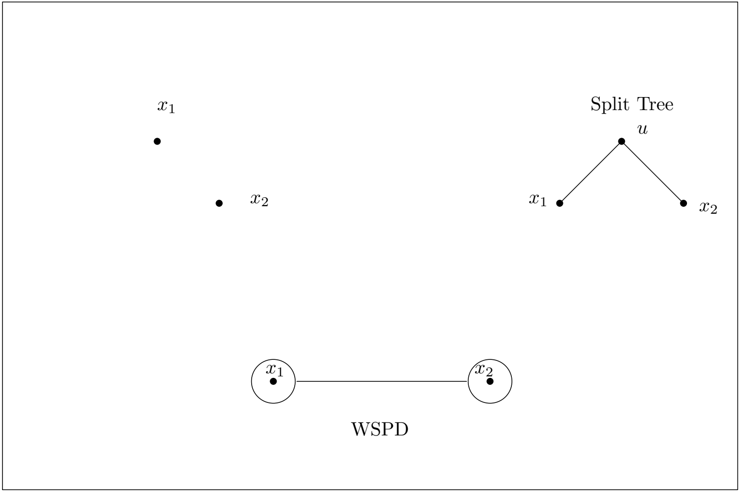
Algorithm FINDPAIRS(v, w)
Input: nodes v and w of the split tree T for S, whose subtrees are disjoint.
Output: A collection of WSP pairs $\{A, B\}$, where A is sorted in subtree of v, and B is sorted in subtree of w.

if S_v and S_w are well-separated with respect to s
    then return the node pair $\{v, w\}$
    else **if** $L_{max}(R(v)) \leq L_{max}(R(w))$
    then
        $w_l :=$ left child of w;
        $w_r :=$ right child of w;
        FINDPAIRS(v, w_l);
        FINDPAIRS(v, w_r);
    else $v_l :=$ left child of v;
        $v_r :=$ right child of v;
        FINDPAIRS(v_l, w);
        FINDPAIRS(v_r, w);

Computing the Well-Separated Pair Decomposition
An Example

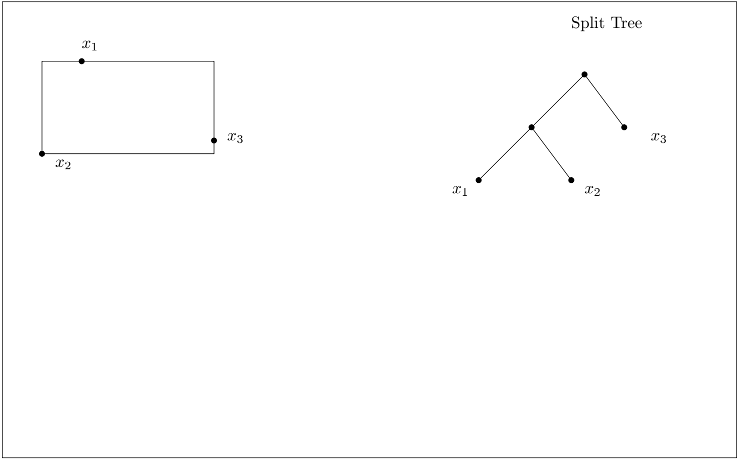


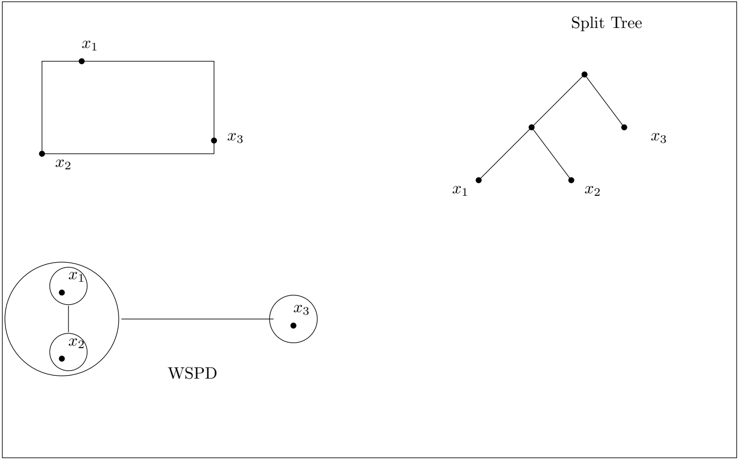
Computing the Well-Separated Pair Decomposition
An Example



Computing the Well-Separated Pair Decomposition

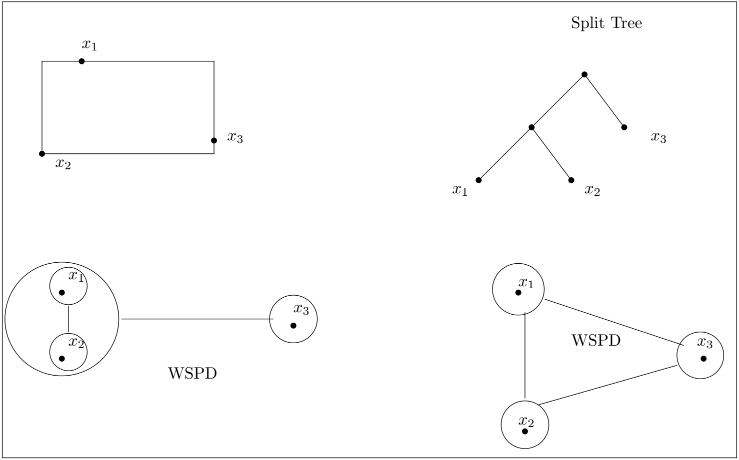
An Example





Split Tree
x_1
x_3
x_2
x_3
x_1
x_2
x_1
x_3
x_2
WSPD

Computing the Well-Separated Pair Decomposition
An Example



Computing the Well-Separated Pair Decomposition

Questions

- Does the algorithm $\textsc{FindPairs}(v, w)$ terminate? Why?
- How do you prove that the algorithm $\textsc{ComputeWSPD}(T, s)$ computes a WSPD for S?
- What is the running time of algorithm $\textsc{ComputeWSPD}(T, s)$?

Computing the Well-Separated Pair Decomposition

Questions

- Does the algorithm FINDPAIRS(v, w) terminate? Why?
- How do you prove that the algorithm COMPUTEWSPD(T, s) computes a WSPD for S?
- What is the running time of algorithm COMPUTEWSPD(T, s)?

Computing the Well-Separated Pair Decomposition

Questions

- Does the algorithm FindPairs(v, w) terminate? Why?
- How do you prove that the algorithm ComputeWSPD(T, s) computes a WSPD for S?
- What is the running time of algorithm ComputeWSPD(T, s)?

Computing the Well-Separated Pair Decomposition

Lemma 9.4.1

Let $v_i, w_i, 1 \leq i \leq m$, be the sequence of node pairs returned by algorithm ComputeWSPD(T, s). The sequence

$$\{S_{v_1}, S_{w_1}\}, \{S_{v_2}, S_{w_2}\}, \ldots, \{S_{v_m}, S_{w_m}\}$$

is a WSPD for the set S with respect to s.

Lemma 9.4.2

Let m be the size of the WSPD for S that is computed by algorithm ComputeWSPD(T, s). The running time of this algorithm is $O(m)$. (This does not include the time to compute the split tree T.)

Computing the Well-Separated Pair Decomposition

Lemma 9.4.1

Let $v_i, w_i, 1 \leq i \leq m$, be the sequence of node pairs returned by algorithm ComputeWSPD(T, s). The sequence

$$\{S_{v_1}, S_{w_1}\}, \{S_{v_2}, S_{w_2}\}, \ldots, \{S_{v_m}, S_{w_m}\}$$

is a WSPD for the set S with respect to s.

Lemma 9.4.2

Let m be the size of the WSPD for S that is computed by algorithm ComputeWSPD(T, s). The running time of this algorithm is $O(m)$. (This does not include the time to compute the split tree T.)

Computing the Well-Separated Pair Decomposition

Representation of the WSPD

For each pair A, B in the WSPD, there are two nodes v and w in the split tree T such that $A = S_v$ and $B = S_w$. Hence, the WSPD can be represented by m pairs of nodes of T.

Computing the Well-Separated Pair Decomposition

Question

What is the good upper bound of m which is determined by the algorithm ComputeWSPD(T, s)?

Proposition

There may be a node a and $\Theta(n)$ nodes b in split tree T for which $\{S_a, S_b\}$ is a pair in the WSPD. Thus, the upper bound is

$$m = \Theta(n^2).$$

Did we compute the upper bound correctly?

We can use the exact analysis of the algorithm ComputeWSPD(T, s) to get the better upper bound on m. How?

Proposition

There may be a node a and $\Theta(n)$ nodes b in split tree T for which $\{S_a, S_b\}$ is a pair in the WSPD. Thus, the upper bound is

$$m = \Theta(n^2).$$

Did we compute the upper bound correctly?

We can use the exact analysis of the algorithm COMPUTEWSPD(T, s) to get the better upper bound on m. How?

Proposition

There may be a node a and $\Theta(n)$ nodes b in split tree T for which $\{S_a, S_b\}$ is a pair in the WSPD. Thus, the upper bound is

$$m = \Theta(n^2).$$

Did we compute the upper bound correctly?

We can use the exact analysis of the algorithm COMPUTEWSPD(T, s) to get the better upper bound on m. How?

The Main Idea

- Give to every well-separated pair a direction.
- Use packing argument to show that every node is involved in at most small number (dependent only on s) of directed pairs.

The Main Idea

- Give to every well-separated pair a direction.
- Use packing argument to show that every node is involved in at most small number (dependent only on s) of directed pairs.

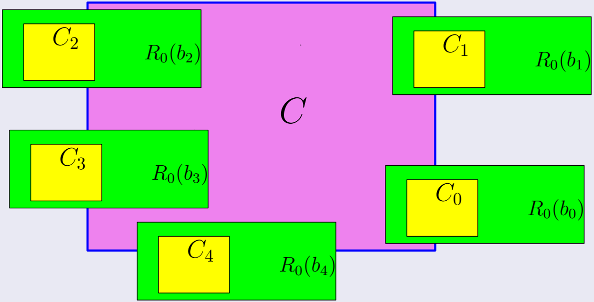
Lemma 9.4.3.

Let C be a hypercube in $\mathbb{R}^d$, let l be the side length of C, and let α be a positive real number. Let $b_1, b_2, \ldots b_k$ be nodes of the split tree T such that

1. b_i is not the root of T for all i with $1 \leq i \leq k$.
2. the sets S_{b_i}, $1 \leq i \leq k$, are pairwise disjoint.
3. $L_{max}(R(\pi(b_i))) \geq l/\alpha$ for all i with $1 \leq i \leq k$.
4. $R(b_i) \cap C \neq \emptyset$ for all i with $1 \leq i \leq k$.

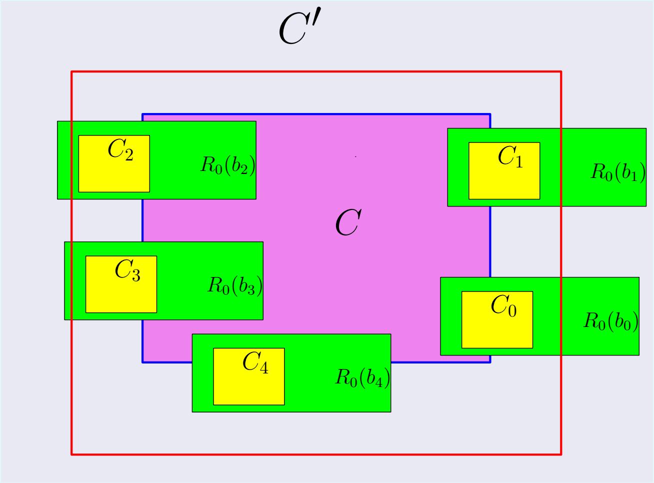
Then $k \leq (2\alpha + 2)^d$.

Sketch of Proof



Computing the Well-Separated Pair Decomposition
The analysis of algorithm COMPUTEWSPD(T, s)

Sketch of Proof



Lemma 9.4.4.

Let a and b be two nodes of the split tree T, and assume that $\{S_a, S_b\}$ is a pair in the WSPD constructed by algorithm COMPUTEWSPD(T, s). Then at least one of the following two claims holds.

1. $S_{\pi(a)}$ and S_b are not well-separated and

$$\begin{cases} L_{max}(R(b)) \leq L_{max}(R(\pi(a))) \\ L_{max}(R(\pi(a))) \leq L_{max}(R(\pi(b))) \end{cases}$$

2. $S_{\pi(b)}$ and S_a are not well-separated and

$$\begin{cases} L_{max}(R(a)) \leq L_{max}(R(\pi(b))) \\ L_{max}(R(\pi(b))) \leq L_{max}(R(\pi(a))) \end{cases}$$

Lemma 9.4.5.

Let a be any node of the split tree T. There are at most $((2s+4)\sqrt{d}+4)^d$ nodes b in T such that (S_a, S_b) is a directed pair in the WSPD computed by algorithm ComputeWSPD(T, s).

Proof

Suppose (S_a, S_b) is a directed pair in the WSPD. It is clear that

$$L_{max}(R(b)) \leq L_{max}(R(\pi(a))).$$

Let C_a and C_b be the balls of radius $\frac{\sqrt{d}}{2} \cdot L_{max}(R(\pi(a)))$ that are centered at the centers of $R(\pi(a))$ and $R(b)$, respectively. We claim that

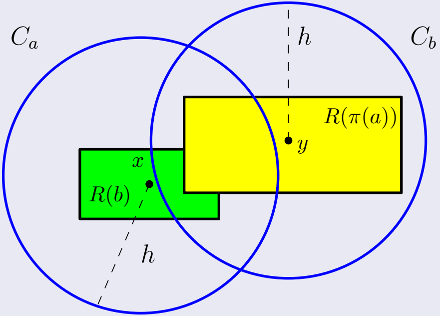
$$|R(\pi(a))R(b)| \leq (s/2+1)\sqrt{d} \cdot L_{max}(R(\pi(a))).$$

We consider two cases:
Case 1: C_a and C_b are not disjoint.

Case 2: C_a and C_b are disjoint.

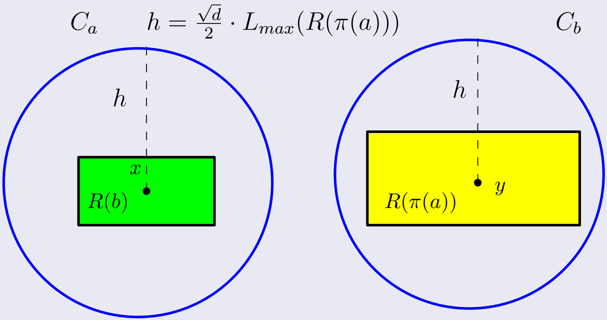
Case 1: C_a and C_b are not disjoint.

$$h = \frac{\sqrt{d}}{2} \cdot L_{max}(R(\pi(a)))$$



$$|R(\pi(a))R(b)| \leq |xy| \leq \sqrt{d} \cdot L_{max}(R(\pi(a))).$$
$$\leq (s/2+1)\sqrt{d} \cdot L_{max}(R(\pi(a)))$$

Case 2: C_a and C_b are disjoint.



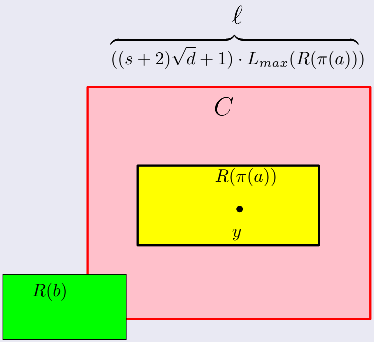
$|C_aC_b| = |xy| - \sqrt{d} \cdot L_{max}(R(\pi(a)))$

$\geq |R(\pi(a)R(b))| - \sqrt{d} \cdot L_{max}(R(\pi(a)))$

$|C_aC_b| < s(\sqrt{d}) \cdot L_{max}(R(\pi(a)))$.

$\boxed{|R(\pi(a))R(b)| \leq (s/2 + 1)\sqrt{d} \cdot L_{max}(R(\pi(a)))}$

Rest of the Proof



$$R(b) \cap C \neq \emptyset$$

$$L_{max}(R(\pi(b))) \geq \frac{\ell}{(s+2)\sqrt{d}+1}$$

Theorem 9.4.6(WSPD Theorem)

Let S be a set of n points in $\mathbb{R}^d$ and let $s > 0$ be a real number.

1. The split tree for S can be computed in $O(n \log n)$ time. This tree has size $O(n)$ and does not depend on the value of s.
2. Given the split tree, we can compute in $O(s^d n)$ time, a WSPD for S with respect to s of size $O(s^d n)$. This WSPD can be represented implicitly in $O(s^d n)$ space.

Corollary 9.4.7(WSPD-Spanner)

Let S be a set of n points in $\mathbb{R}^d$ and let $t > 1$ be a real number. In $O(n \log n + n/(t-1)^d)$ time, we can construct a t-spanner for S having $O(n/(t-1)^d)$ edges.

The WSPD-spanner has $O(n)$ edges and can be constructed in $O(n \log n)$ time.

Corollary 9.4.7(WSPD-Spanner)

Let S be a set of n points in $\mathbb{R}^d$ and let $t > 1$ be a real number. In $O(n \log n + n/(t-1)^d)$ time, we can construct a t-spanner for S having $O(n/(t-1)^d)$ edges.

The WSPD-spanner has $O(n)$ edges and can be constructed in $O(n \log n)$ time.

Question

What is the quality measures such as degree, diameter, weight and etc for the WSPD-spanner?

Answer

1. **Degree:**There is no nontrivial bound on the degree of the WSPD-spanner.Why?
2. **diameter:**There is no nontrivial bound on the diameter of the WSPD-spanner. Why?
3. **weight:**$wt(\text{WSPD-spanner}(S)) = O(\log n \cdot wt(\text{MST}(S)))$ Why?

Computing the Well-Separated Pair Decomposition

The analysis of algorithm COMPUTEWSPD(T, s)

Question

What is the quality measures such as degree, diameter, weight and etc for the WSPD-spanner?

Answer

1. **Degree:** There is no nontrivial bound on the degree of the WSPD-spanner. Why?
2. **diameter:** There is no nontrivial bound on the diameter of the WSPD-spanner. Why?
3. **weight:** $wt(\text{WSPD-spanner}(S)) = O(\log n \cdot wt(\text{MST}(S)))$. Why?

Question

What is the quality measures such as degree, diameter, weight and etc for the WSPD-spanner?

Answer

1. **Degree:**There is no nontrivial bound on the degree of the WSPD-spanner.Why?
2. **diameter:**There is no nontrivial bound on the diameter of the WSPD-spanner. Why?
3. **weight:**$wt(\text{WSPD-spanner}(S)) = O(\log n \cdot wt(\text{MST}(S)))$. Why?

Question

What is the quality measures such as degree, diameter, weight and etc for the WSPD-spanner?

Answer

1. **Degree:**There is no nontrivial bound on the degree of the WSPD-spanner.Why?
2. **diameter:**There is no nontrivial bound on the diameter of the WSPD-spanner. Why?
3. **weight:**$wt(\text{WSPD-spanner}(S)) = O(\log n \cdot wt(\text{MST}(S)))$. Why?

Outline

Finding the pair that separate two points

PAIR QUERY PROBLEM

Given a WSPD $\{A_i, B_i\}$, $1 \leq i \leq m$, for a set S of n points in $\mathbb{R}^d$, and given two distinct points p and q in S, compute the index i for which $p \in A_i$ and $q \in B_i$, or $p \in B_i$ and $q \in A_i$.

How do you solve the PAIR QUERY PROBLEM?

Finding the pair that separate two points

Pair Query Problem

Given a WSPD $\{A_i, B_i\}$, $1 \leq i \leq m$, for a set S of n points in $\mathbb{R}^d$, and given two distinct points p and q in S, compute the index i for which $p \in A_i$ and $q \in B_i$, or $p \in B_i$ and $q \in A_i$.

How do you solve the Pair Query Problem?

Finding the pair that separate two points

Methods

1. Centroid edges.
2. Path decomposition.

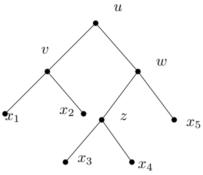
Finding the pair that separate two points

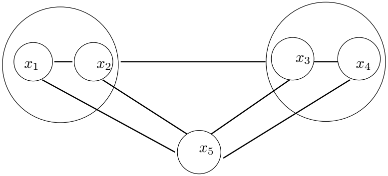
Binary Recursion Tree

For each internal node u of the split tree T, we define a *binary recursion tree* $RT(v, w)$ where v and w are two children of u, as follows:

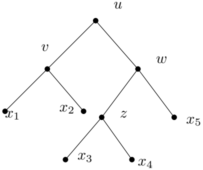
- If S_v and S_w are well-separated, then $RT(v, w)$ is a single node storing $R(v)$ and $R(w)$, and two pointers to the nodes v and w in T.
- Otherwise
  - if $L_{max}(R(v)) \leq L_{max}(R(w))$. Then $RT(v, w)$ is a root storing the $R(v)$ and $R(w)$. It's two children are recursion trees $RT(v, w_l)$ and $RT(v, w_r)$.
  - if $L_{max}(R(v)) > L_{max}(R(w))$. Then $RT(v, w)$ is a root storing the $R(v)$ and $R(w)$. It's two children are recursion trees $RT(v_l, w)$ and $RT(v_r, w)$.

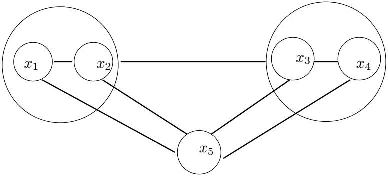
Finding the pair that separate two points





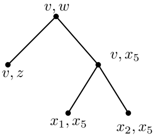
Finding the pair that separate two points



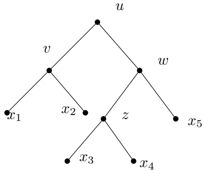


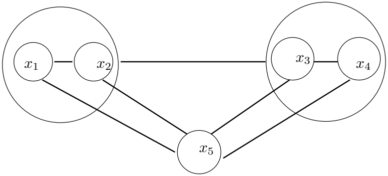
Internal node: u

$RT(v, w)$



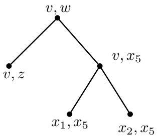
Finding the pair that separate two points





Internal node: u

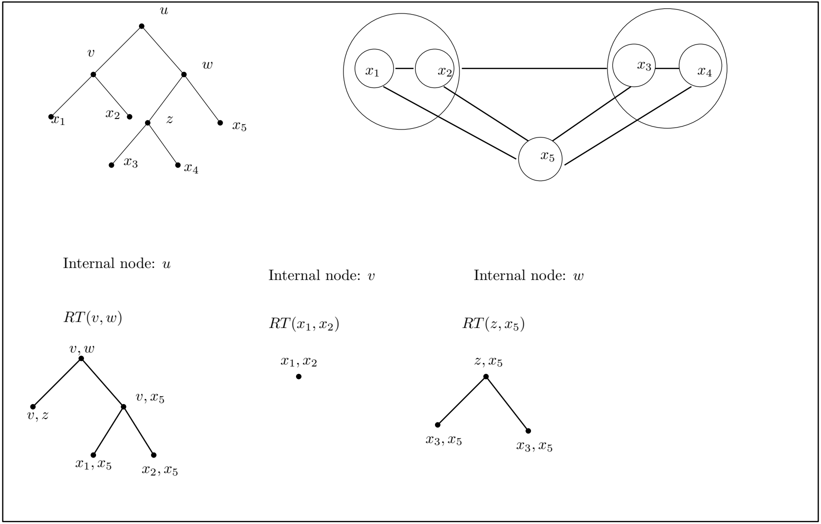
$RT(v, w)$



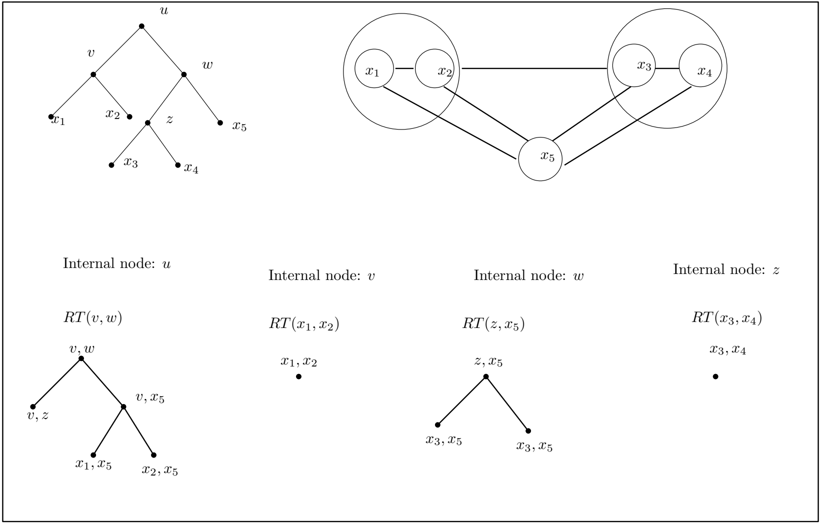
Internal node: v

$RT(x_1, x_2)$

Finding the pair that separate two points



Finding the pair that separate two points



Finding the pair that separate two points

Observation

Each pair in the WSPD corresponds to a unique leaf in exactly one of the recursion trees. Conversely, any leaf in any recursion tree corresponds to exactly one pair in the WSPD.

Finding the pair that separate two points

An algorithm to solve the PAIR QUERY problem for two distinct points p and q

1. Find the lowest common ancestor u of the p and q in the split tree T. $v := left\_child(u)$ and $w := right\_child(u)$
2. Walking down the tree $RT(v, w)$ to find the leaf ℓ that stores pointers to a and b such that $p \in R(a)$ and $q \in R(b)$. How?

Finding the pair that separate two points

An algorithm to solve the PAIR QUERY problem for two distinct points p and q

1. Find the lowest common ancestor u of the p and q in the split tree T. $v := left\_child(u)$ and $w := right\_child(u)$
2. Walking down the tree $RT(v, w)$ to find the leaf ℓ that stores pointers to a and b such that $p \in R(a)$ and $q \in R(b)$. **How?**

Finding the pair that separate two points

Questions

- Dose the above algorithm work correctly?
- What is the time complexity of the above algorithm?

Finding the pair that separate two points

Questions

- Dose the above algorithm work correctly?
- What is the time complexity of the above algorithm?

Finding the pair that separate two points

Centroid edges

Answering pair queries using centroid edges

1. Find the lowest common ancestor u of the p and q in the split tree T. $v := left\_child(u)$ and $w := right\_child(u)$
2. Find the centroid edge $e = (y, x)$ of $RT(v, w)$. Let x be the endpoint of e that is farthest away from the root of $RT(v, w)$.
3. Let a_x and b_x be the two nodes of T that correspond to x, where $R(a_x) \subseteq R(v)$ and $R(b_x) \subseteq R(w)$
4. If $p \in R(a_x)$ and $q \in R(b_x)$, walking down the subtree of $RT(v, w)$ rooted at x.
5. Otherwise, walking down the tree obtained from $RT(v, w)$ by deleting the subtree rooted at x.

Finding the pair that separate two points

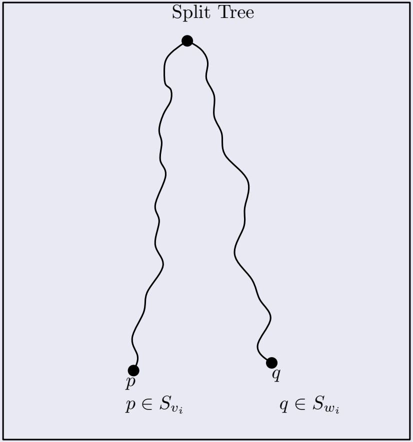
Centroid edges

Theorem 9.5.2

Let S be a set of n points in $\mathbb{R}^d$, and let $s > 0$ be a real number. The WSPD of Theorem 9.4.6 can be represented in $O(s^d n)$ space, such that for any two distinct points p and q in S, a pair query can be answered in $O(\log n)$ time. This representation can be computed in $O(s^d n \ \log n)$ time.

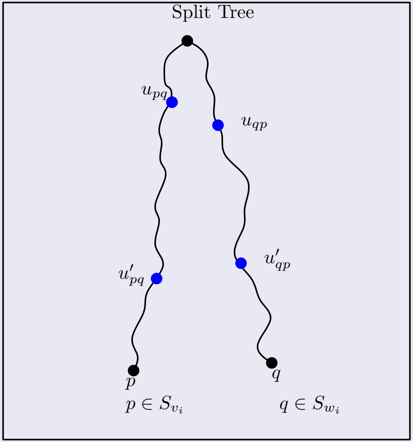
Finding the pair that separate two points
Path decomposition

Basic Idea of the Path Decomposition

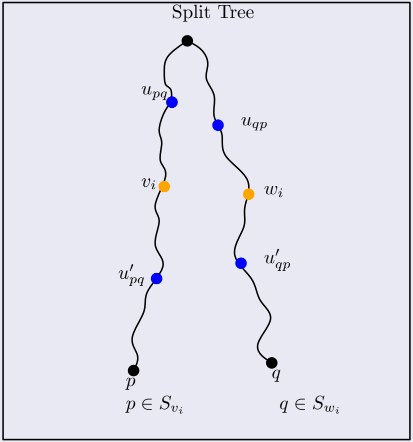


Finding the pair that separate two points
Path decomposition

Basic Idea of the Path Decomposition



Basic Idea of the Path Decomposition



Lemma 9.5.3.

Let b and b' be two nodes in the split tree T such that b is in the subtree of b' and the path between them contains at least d edges. Then

$$L_{max}(R(b)) \leq \frac{1}{2} \cdot L_{max}(R(b')).$$

Proof

It is sufficient to prove that for each $1 \leq i \leq d$,

$$L_i(R(b)) \leq \frac{1}{2} \cdot L_{max}(R(b')).$$

Let b'' be the child of b' such that b is in the subtree of b''.

We have two cases:

1. There is a node u on the path between b and b'' such that algorithm SPLITTREE splits $R(\pi(u))$ along dimension i.
2. For each node u on the path between b and b'' algorithm SPLITTREE splits $R(\pi(u))$ along a dimension different from i.

Finding the pair that separate two points

Path decomposition

Proof of Case 1

$$
\begin{aligned}
L_i(R(b)) \leq & L_i(R(u)) \\
\leq & \frac{1}{2} \cdot L_i(R(\pi(u))) \\
\leq & \frac{1}{2} \cdot L_i(R(b')) \\
\leq & \frac{1}{2} \cdot L_{max}(R(b')).
\end{aligned}
$$

Finding the pair that separate two points

Path decomposition

Proof of Case 2

Based on the *pigeonhole principle*, there is an index $j \neq i$, and two distinct nodes u and v on the path, such that $R(\pi(u))$ and $R(\pi(v))$ are split along dimension j. W.L.G u is in the subtree of v. Then

$$
\begin{aligned}
L_i(R(b)) \leq & L_i(R(\pi(u))) \\
\leq & L_j(R(\pi(u))) \\
\leq & L_j(R(v)) \\
\leq & \frac{1}{2} \cdot L_j(R(\pi(v))) \\
\leq & \frac{1}{2} \cdot L_j(R(b')) \\
\leq & \frac{1}{2} \cdot L_{max}(R(b')).
\end{aligned}
$$

Lemma 9.5.4

Let A and B be two bounded subsets of $\mathbb{R}^d$, let p be a point in A, let q be a point in B, and let $s > 0$ be a real number and $\alpha := \frac{2}{(s+4)\sqrt{d}}$.

1. If A and B are well-separated with respect to s, then both $L_{max}(R(A))$ and $L_{max}(R(B))$ are less than or equal to $(2/s)|pq|$.
2. If both $L_{max}(R(A))$ and $L_{max}(R(B))$ are less than or equal to $\alpha|pq|$, then A and B are well-separated with respect to s.

Definition

For any ordered pair (p, q) of distinct points in the point set S, we define the following two nodes in T:

- u_{pq} is the highest node u on the path in T from the leaf storing p to the root, such that $L_{max}(R(u)) \leq (2/s)|pq|$.
- u'_{pq} is the highest node u on the path in T from the leaf storing p to the root, such that $L_{max}(R(u)) \leq \alpha|pq|$.

For each i with $1 \leq i \leq m$, let v_i and w_i be the nodes in T such that $A_i = S_{v_i}$ and $B_i = S_{w_i}$.

Lemma 9.5.5

Let p and q be two distinct points of S, and let i be the index such that (i) $p \in A_i$ and $q \in B_i$, or (ii) $p \in B_i$ and $q \in A_i$. Assume without loss of generality that (i)holds.

1. If we follow the path in T from the leaf storing p to the root, then we encounter, in this order, the nodes u'_{pq}, v_i , and u_{pq}.
2. If we follow the path in T from the leaf storing q to the root, then we encounter, in this order, the nodes u'_{qp}, w_i , and u_{qp}.
3. The path in T between u'_{pq} and u_{pq} contains $O(\log \frac{1}{s})$ nodes.
4. The path in T between u'_{qp} and u_{qp} contains $O(\log \frac{1}{s})$ nodes.
5. Given pointers to the nodes u_{pq} and u_{qp}, we can compute the nodes v_i and w_i in $O(\log \frac{1}{s})$ time.

Finding the pair that separate two points
Path decomposition

Algorithm PAIRQUERY(p, q)

1. Partition the Split tree T into pairwise disjoint paths.%see section 2.3.2.
2. For each path P in the partition of T, find the T_P.
3. Find the paths P and Q in the partition of T that contains the node u_{pq} and u_{qp}, respectively .
4. Find nodes u_{pq} on P using T_P.
5. Find nodes u_{qp} on Q using T_Q.
6. Find nodes v_i which $p \in S_{v_i}$ and $q \in S_{w_i}$.
7. return v_i and w_i.

Algorithm PAIRQUERY(p, q)

1. Partition the Split tree T into pairwise disjoint paths.%see section 2.3.2. $O(n)$
2. For each path P in the partition of T, find the T_P.
3. Find the paths P and Q in the partition of T that contains the node u_{pq} and u_{qp}, respectively.
4. Find nodes u_{pq} on P using T_P.
5. Find nodes u_{qp} on Q using T_Q.
6. Find nodes v_i which $p \in S_{v_i}$ and $q \in S_{w_i}$.
7. return v_i and w_i.

Algorithm PAIRQUERY(p, q)

1. Partition the Split tree T into pairwise disjoint paths.%see section 2.3.2. $O(n)$
2. For each path P in the partition of T, find the T_P.$O(n)$
3. Find the paths P and Q in the partition of T that contains the node u_{pq} and u_{qp}, respectively.
4. Find nodes u_{pq} on P using T_P.
5. Find nodes u_{qp} on Q using T_Q.
6. Find nodes v_i which $p \in S_{v_i}$ and $q \in S_{w_i}$.
7. return v_i and w_i.

Algorithm PAIRQUERY(p, q)

1. Partition the Split tree T into pairwise disjoint paths.%see section 2.3.2. $O(n)$
2. For each path P in the partition of T, find the T_P.$O(n)$
3. Find the paths P and Q in the partition of T that contains the node u_{pq} and u_{qp}, respectively.$O(\log n)$
4. Find nodes u_{pq} on P using T_P.
5. Find nodes u_{qp} on Q using T_Q.
6. Find nodes v_i which $p \in S_{v_i}$ and $q \in S_{w_i}$.
7. return v_i and w_i.

Algorithm PAIRQUERY(p, q)

1. Partition the Split tree T into pairwise disjoint paths.%see section 2.3.2. $O(n)$
2. For each path P in the partition of T, find the T_P.$O(n)$
3. Find the paths P and Q in the partition of T that contains the node u_{pq} and u_{qp}, respectively.$O(\log n)$
4. Find nodes u_{pq} on P using T_P.$O(\log n)$
5. Find nodes u_{qp} on Q using T_Q.
6. Find nodes v_i which $p \in S_{v_i}$ and $q \in S_{w_i}$.
7. return v_i and w_i.

Algorithm PAIRQUERY(p, q)

1. Partition the Split tree T into pairwise disjoint paths.%see section 2.3.2. $O(n)$
2. For each path P in the partition of T, find the T_P.$O(n)$
3. Find the paths P and Q in the partition of T that contains the node u_{pq} and u_{qp}, respectively.$O(\log n)$
4. Find nodes u_{pq} on P using T_P.$O(\log n)$
5. Find nodes u_{qp} on Q using T_Q.$O(\log n)$
6. Find nodes v_i which $p \in S_{v_i}$ and $q \in S_{w_i}$.
7. return v_i and w_i.

Algorithm PAIRQUERY(p, q)
1. Partition the Split tree T into pairwise disjoint paths.%see section 2.3.2. $O(n)$
2. For each path P in the partition of T, find the T_P.$O(n)$
3. Find the paths P and Q in the partition of T that contains the node u_{pq} and u_{qp}, respectively.$O(\log n)$
4. Find nodes u_{pq} on P using T_P.$O(\log n)$
5. Find nodes u_{qp} on Q using T_Q.$O(\log n)$
6. Find nodes v_i which $p \in S_{v_i}$ and $q \in S_{w_i}$.$O(\log \frac{1}{s})$
7. return v_i and w_i.

Theorem 9.5.6.

Let S be a set of n points in $\mathbb{R}^d$, and let $s > 0$ be a real number. The WSPD of Theorem 9.4.6 can be represented in $O(s^d n)$ space, such that for any two distinct points p and q in S, a pair query can be answered in $O(\log n + \log 1/s)$ time. Given the split tree T and this WSPD, this representation can be computed in $O(n)$ time.

Outline

Extention to Other Metrics

Metric Space

A metric space is a pair (S, δ), where S is a (finite or infinite) set, whose elements are called points, and $\delta : S \times S \longrightarrow \mathbb{R}$ is a function that assigns a distance $\delta(p, q)$ to any two points p and q in S, and that satisfies the following four conditions:

1. For all points p and q in S, $\delta(p, q) \geq 0$.
2. For all points p and q in S, $\delta(p, q) = 0$ if and only if $p = q$.
3. For all points p and q in S, $\delta(p, q) = \delta(q, p)$.
4. For all points p, q, and r in S, $\delta(p, q) \geq \delta(p, r) + \delta(r, q)$.

Extention to Other Metrics

Metric Space

A metric space is a pair (S, δ), where S is a (finite or infinite) set, whose elements are called points, and $\delta : S \times S \longrightarrow \mathbb{R}$ is a function that assigns a distance $\delta(p, q)$ to any two points p and q in S, and that satisfies the following four conditions:

1. For all points p and q in S, $\delta(p, q) \geq 0$.
2. For all points p and q in S, $\delta(p, q) = 0$ if and only if $p = q$.
3. For all points p and q in S, $\delta(p, q) = \delta(q, p)$.
4. For all points p, q, and r in S, $\delta(p, q) \geq \delta(p, r) + \delta(r, q)$.

Extention to Other Metrics

Metric Space

A metric space is a pair (S, δ), where S is a (finite or infinite) set, whose elements are called points, and $\delta : S \times S \longrightarrow \mathbb{R}$ is a function that assigns a distance $\delta(p, q)$ to any two points p and q in S, and that satisfies the following four conditions:

1. For all points p and q in S, $\delta(p, q) \geq 0$.
2. For all points p and q in S, $\delta(p, q) = 0$ if and only if $p = q$.
3. For all points p and q in S, $\delta(p, q) = \delta(q, p)$.
4. For all points p, q, and r in S, $\delta(p, q) \geq \delta(p, r) + \delta(r, q)$.

Extention to Other Metrics

Metric Space

A metric space is a pair (S, δ), where S is a (finite or infinite) set, whose elements are called points, and $\delta : S \times S \longrightarrow \mathbb{R}$ is a function that assigns a distance $\delta(p, q)$ to any two points p and q in S, and that satisfies the following four conditions:

1. For all points p and q in S, $\delta(p, q) \geq 0$.
2. For all points p and q in S, $\delta(p, q) = 0$ if and only if $p = q$.
3. For all points p and q in S, $\delta(p, q) = \delta(q, p)$.
4. For all points p, q, and r in S, $\delta(p, q) \geq \delta(p, r) + \delta(r, q)$.

Extention to Other Metrics

Diameter and Distance

- The *diameter* $D(A)$ of a subset A of S is defined as

$$D(A) := max\{\delta(a, b) : a, b \in A\}.$$

- The *distance* $\delta(A, B)$ of two subsets A and B of S is defined as

$$\delta(A, B) := min\{\delta(a, b) : a \in A, b \in B\}.$$

Extention to Other Metrics

Diameter and Distance

- The *diameter* $D(A)$ of a subset A of S is defined as

$$D(A) := max\{\delta(a,b) : a,b \in A\}.$$

- The *distance* $\delta(A,B)$ of two subsets A and B of S is defined as

$$\delta(A,B) := min\{\delta(a,b) : a \in A, b \in B\}.$$

Extention to Other Metrics

WSPD in a metric space

For a real number $s > 0$, we say that the subsets A and B of S are well-separated with respect to s if

$$\delta(A, B) \geq s \cdot max(D(A), D(B)).$$

Using this generalized notion of being well-separated, we define a well-separated pair decomposition (WSPD) for S, with respect to the separation ratio s, as in Definition 9.1.3.

Extention to Other Metrics

Open Problem

Which metric spaces (S, δ) admit a WSPD of subquadratic size? Design efficient algorithms that compute such a WSPD.

Extention to Other Metrics

Open Problem

Which metric spaces (S, δ) admit a WSPD of subquadratic size? Design efficient algorithms that compute such a WSPD.


May 2 2014
Thank you. Question?
JOIN THE DEBATE
Geometric Spanner Networks
Giri Narasimhan
Relegation worries