

INTRODUCTION TO GEOMETRIC COMPUTATIONS: CS60064, SPRING 2023

Sudebkumar Prasant Pal,
Department of Computer Science and Engineering, IIT Kharagpur, 721302.

April 22, 2023

- 1 KIRKPATRICK'S OPTIMAL PLANAR POINT LOCATION METHOD
- 2 RANDOMIZED INCREMENTAL CONSTRUCTION
- 3 CONFIGURATION SPACES [3] [9]
- 4 RANGE SPACES AND ϵ -NETS
- 5 RANDOM SAMPLING [5]
- 6 DETERMINISTIC CONSTRUCTION OF ϵ -NETS [5]
- 7 ϵ -NETS AND VC-DIMENSION
- 8 RANDOM SAMPLING IN GEOMETRY
- 9 THE WEAK CUTTING LEMMA
- 10 2D LINEAR PROGRAMMING IN EXPECTED LINEAR TIME
- 11 COMPUTING VORONOI DIAGRAMS
- 12 FORTUNE'S SWEEPLINE ALGORITHM
- 13 COMPUTING USING HIGHER DIMENSIONAL STRUCTURE
- 14 OPTIMAL ONLINE CONVEX HULL COMPUTATION
- 15 RANGE SEARCHING
 - 1-d range searching
 - 2-d range searching

- Range trees and kd-trees
- Range searching with 2-d range trees
- Range searching using kd-trees
- Other kinds of range searching

16 INTERVAL TREES

17 COMPUTING VISIBLE REGIONS USING ANGULAR SWEEP

OPTIMAL PLANAR POINT LOCATION BY KIRKPATRICK [8]

- Consider Kirkpatrick's planar point location method. Suppose we have a planar graph of n vertices, embedded in the plane where each face is a triangle.
- Exercise 1: Show that there are no more than $3n - 6$ edges and that the equality actually holds.
- Let D be the set of vertices with degree less than 12. Then, (Exercise 2) show that there are at least $\frac{n}{2}$ vertices with degree less than 12.
- Now greedily proceed to select one vertex at a time from D , so that we generate a maximal independent set in the graph.
- Assume that the outermost face is a triangle which encloses all other edges (in the context of Kirkpatrick's planar point location method); the three vertices of this outermost triangle cannot be selected.

OPTIMAL PLANAR POINT LOCATION BY KIRKPATRICK [8] CONTD.

- Since this developing independent set is a subset of D , whenever we select a vertex, we *may* also kill at most 11 other vertices of D , which cannot be selected in the subsequent steps. So, with each vertex selected in D , a total of no more than 12 vertices of D are out of consideration.
- This helps us estimate a lower bound on the size of the greedy maximal independent set as follows.
- If m is the size of any maximal independent set then show that (Exercise 3) $m \geq \lfloor \frac{1}{12}(\frac{n}{2} - 3) \rfloor$.
- Since we are dropping at least a constant fraction (Exercise 4) of the vertices in each stage, show (Exercise 5) that the total time required for preprocessing is $O(n \log n)$, the total space required is $O(n)$ (Exercise 6), and (Exercise 7) queries can be answered in $O(\log n)$ time (Theorem 2.7, page 60 in [8]).

PLANAR PARTITION (TRAPEZOIDAL) COMPUTATION BY INCREMENTAL CONSTRUCTION [3] SECTION 3.1

- Let n line segments s_1, s_2, \dots, s_n in the plane intersect in $k = O(n^2)$ points. Let us consider a random permutation of these n segments. We process the $(i + 1)$ st segment when the planar partition (trapezoidal decomposition) $H(N^i)$ of the first i segments, say s_1, s_2, \dots, s_i are processed from the random permutation, with k_i intersecting pairs determined between the i segments.
- Inserting the segment s_{i+1} is the incremental step, creating $H(N^{i+1})$, using the DCEL data structure, starting from one end of s_{i+1} and tracing the faces of $H(N^i)$ till we walk to the other end of s_{i+1} .
- Let v denote an intersection point of the k intersections; let I_v be the 0-1 indicator variable for v appearing in $H(N^i)$. The probability that the two segments intersecting in v are in the first i segments of the random permutation is $O(\frac{i^2}{n^2})$. Why? (Exercise 1)

PLANAR PARTITION COMPUTATION BY INCREMENTAL CONSTRUCTION [3] CONTD.

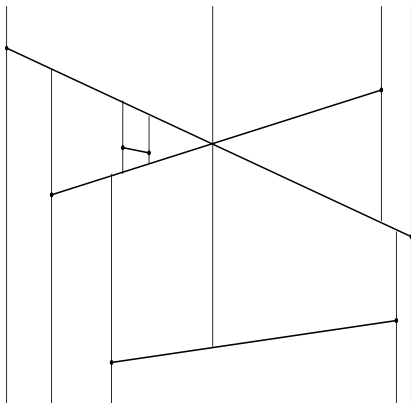


FIGURE:

PLANAR PARTITION COMPUTATION BY INCREMENTAL CONSTRUCTION [3] CONTD.

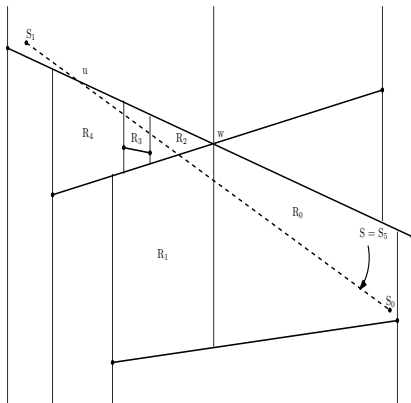


FIGURE:

PLANAR PARTITION COMPUTATION BY INCREMENTAL CONSTRUCTION [3] CONTD.

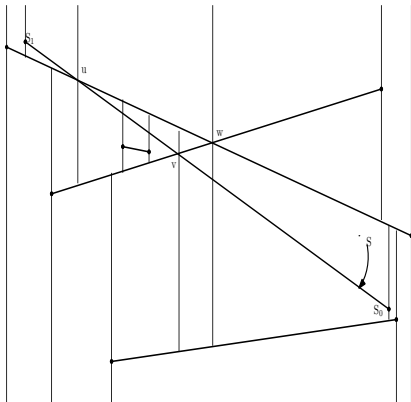


FIGURE:

PLANAR PARTITION COMPUTATION BY INCREMENTAL CONSTRUCTION [3] CONTD.

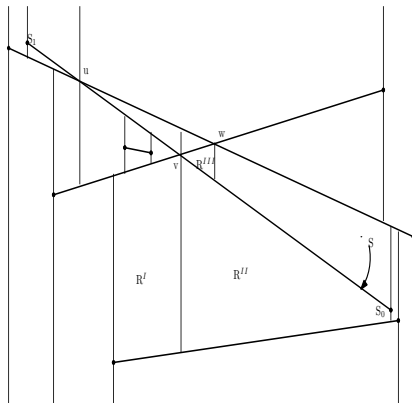


FIGURE:

PLANAR PARTITION COMPUTATION BY INCREMENTAL CONSTRUCTION [3] CONTD.

- In i events of segments' selections, a segment intersecting in v could be picked up with probability roughly $\frac{1}{n}$ in each selection, and thus with upperbounded probability of $O(\frac{i}{n})$ overall, and so for also the other segment meeting v .
- So, the sum of all the k I_v 's has expectation k times the probability upper bound for each v appearing in $H(N^i)$, that is, $O(k \frac{i^2}{n^2})$.
- Now we see the $i + 1$ equally probable cases about the segment that was last inserted when we went from $H(N^i)$ to $H(N^{i+1})$, with cost proportional to the sizes of faces cut by the segment s_{i+1} and the conflict list size of one endpoint of that segment. Here, s_{i+1} can be any of the $i + 1$ segments in N^{i+1} .
- The buckets of end vertices of segments in faces can get split into parts on the insertion of a new segment that cuts the face. Exit/entry of faces as we move across the new segment requires seeing all edges of the faces

PLANAR PARTITION COMPUTATION BY INCREMENTAL CONSTRUCTION [3] CONTD.

- To get the expected running time of the insertion of the new segment, it is enough to compute the average of the $i + 1$ equally probable costs of insertion of each of the $i + 1$ segments.
- Since each face is defined by only a constant number of segments, the numerator adds up to a quantity proportional to $|H(N^{i+1})|$ plus the count of the remaining $n - i$ endpoints in the buckets of $H(N^i)$. The denominator is $i + 1$.
- This cost is clearly $O(\frac{n+k_{i+1}}{i+1})$ (Exercise 2). Apart from the cost of intersections, this cost is at most linear in n .
- Now we sum this cost over all n iterations to get the expectation of the sum of all the n insertions.
- The $n \log n$ term comes from the harmonic series sum, and the term k is due to use of the expectation $k(i + 1)^2/n^2$ of k_{i+1} , thereby adding up to the total expected cost $O(n \log n + k)$ (Exercise 3).

ABSTRACT CONFIGURATION SPACES SECTION 3.4 IN [3] AND CHAPTER 9, THEOREMS 9.14 AND 9.15 IN [9]

- Based on the definitions of *triggers* set $D(\sigma)$ and *stoppers* set $L(\sigma)$ for a configuration $\sigma \in \Pi(N)$, we note the *degree* $d(\sigma)$ and the *conflict size* or *level* $l(\sigma)$ of the trigger and stopper sets.
- Several configurations in the set of all configurations $\Pi(N)$ may have the same trigger and stopper sets. So, $\Pi(N)$ is a multiset of say $\pi(N)$ elements. Let $\Pi^i(N)$ be the set of $\pi^i(N)$ configurations with level i .
- For any $R \subseteq N$, a trapezoid (configuration) σ is feasible or possible over N if it occurs in the trapezoidal decomposition of $H(R)$. Each such trapezoids can appear in the insertion steps of some random permutation of N .
- $D(\sigma)$ is the set of segments adjacent to the boundary of σ , and $L(\sigma)$ is the set of segments in $N \setminus D(\sigma)$ intersecting σ .

ABSTRACT CONFIGURATION SPACES SECTIONS 3.2 AND 3.4 [3] AND CHAPTER 9, THEOREMS 9.14 AND 9.15 [9]

- This definition is somewhat different from the case of the planar partition incremental construction where we just took endpoints of segments as conflicts.
- Note that $\Pi^0(N)$ is the set of trapezoids in $H(N)$. We can project a configuration σ of $H(N)$ in $H(R)$ if $D(\sigma) \subseteq R$ and call it σ_R if its conflict set is $L(\sigma) \cap R$.
- $\Pi^0(R)$ is the set of trapezoids in $H(R)$. The conflict size of any trapezoid in $H(R)$ relative to N is the number of lines in $N \setminus R$ that intersect the trapezoid.
- Example 3.4.4 is the example of “raquets”, culminating in Theorem 3.4.5, whose proof mimics that of Lemma 3.2.1 and Equation 3.7 for the Example 3.4.2 of convex polytopes.

ABSTRACT CONFIGURATION SPACES (CONTD.)

- For N , a set of half-spaces in R^d , each vertex of the arrangement $G(N)$ has the trigger set of the halfspaces in N bounded by hyperplanes containing the vertex. The stopper set is the set of halfspaces whose complements contain the vertex.

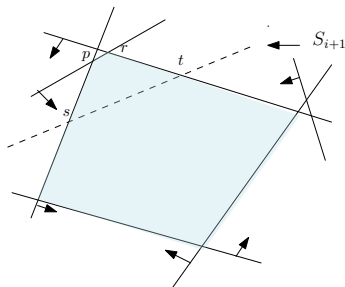


FIGURE: Given set of halfplanes

ABSTRACT CONFIGURATION SPACES (CONTD.)

- The cost of adding S_j , is proportional to the number $m(S, N^j)$ of newly created vertices in $H(N^j)$. The expected cost of this j th step is therefore proportional to $\frac{1}{j} \sum_{S \in N^j} m(S, N^j)$.
- For each $j < n$, let $e(j)$ denote the expected size (number of vertices) of $\Pi^0(N^j)$, assuming that the N^j is a random sample (subset) of N of size j . Let $d = d(\Pi)$ denote the maximum degree of a configuration (vertex) in $\Pi(N)$. By the definition of a configuration space, d is bounded by a constant.
- Since each configuration (vertex) in $H(N^j)$ is defined by (adjacent to) d objects (halfspaces) in N^j , the expected cost of the j th step is proportional to the quantity $\frac{1}{j} \sum_{S \in N^j} m(S, N^j)$, which itself is at most $\frac{d}{j}$ times $e(j)$.

ABSTRACT CONFIGURATION SPACES (CONTD.)

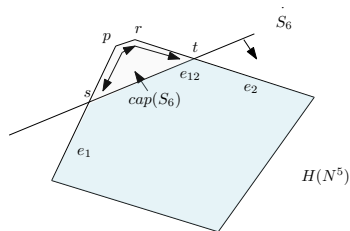
- See Lemma 3.2.1 and equation (3.7) [3] for the configuration space of vertices described in Example 3.4.2. Lemma 3.2.1 and equation (3.7) use the fact every vertex is adjacent to at most d half-spaces. So, the corresponding configuration space has the bounded degree property where every configuration in $\Pi(N)$ is adjacent to at most d objects in N .
- Just mimicking Lemma 3.2.1 and equation (3.7), the present general setting can be mirrored by replacing "vertices" and "half-spaces" by the "configurations" and "objects," respectively. The Lemma 3.2.1 stated verbatim from [3] is—
 “The expected number of newly created vertices in the j th random addition, conditional on a fixed N^j , is bounded by $\frac{d}{j}$ times the number of active vertices over N^j (i.e., the ones in $H(N^j)$). Since N^j is itself the random sample of N , the expected value, without any conditioning, is bounded by $d \frac{e(j)}{j}$, where $e(j)$ denotes the expected number of active vertices at time j .”

ABSTRACT CONFIGURATION SPACES (CONTD.)

- For $d = 2, 3$, this expected number is $O(j)$. Hence, the expected amortized cost of the j th addition is $O(1)$, assuming that we are given a vertex of $H(N^{j-1})$ in conflict with S_j .”
- Now about conflicts—How do we find a vertex of $H(N^{j-1})$ in conflict with S_j ? We must have it “readymade” and so must “maintain” for every half-space $I \in N \setminus N^{j-1}$, a pointer to one vertex in $H(N^{j-1})$ in conflict with it, provided there is one indeed.
- For each time i , we maintain for every halfspace $I \in N \setminus N^i$, a pointer to one vertex in $H(N^i)$ in conflict with it, if there is one such vertex.
- During the addition of $S = S_{i+1}$, we now need to update conflict information accordingly (see Figures 3.5(c)-(e)). If the conflict pointer from some halfspace $I \in N \setminus N^{i+1}$ points to a vertex r in $\text{cap}(S_{i+1})$, then we need to find another vertex in $p \in H(N^{i+1})$ in conflict with I .

ABSTRACT CONFIGURATION SPACES (CONTD.)

- Observe that some newly created vertex of $H(N^{i+l})$, contained in S_{i+1} , must conflict with l .



ABSTRACT CONFIGURATION SPACES (CONTD.)

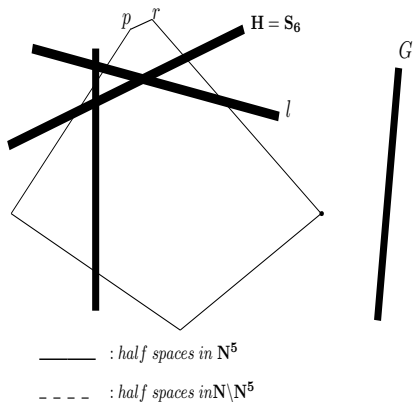


FIGURE:

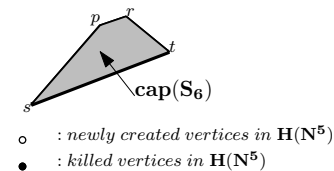


FIGURE:

ABSTRACT CONFIGURATION SPACES (CONTD.)

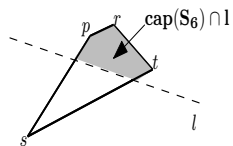


FIGURE:

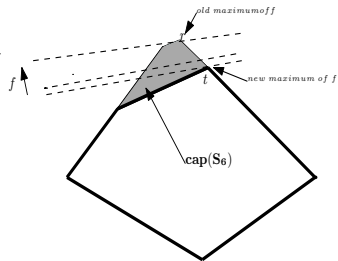


FIGURE:

ABSTRACT CONFIGURATION SPACES (CONTD.)

- We set $j = i + 1$. The part of $H(N^i)$ cut off by S_{i+1} is called $cap(S_{i+1})$. If the conflict vertex r for a certain $I \in N \setminus N^{i+1}$ is in $cap(S_{i+1})$ then we need to search for a new conflict vertex for I , which must be a newly created vertex in $H(N_{i+1})$.
- The search will see old vertices in the cap but they get deleted anyway and thus we focus only on the number $k(N^j, S^j, I)$ of *newly created vertices* in $H(N^j)$ in conflict with I , and adjacent to S_j .
- The sum of the number of all such newly created vertices over all adjacent $S \in N^j$, is no more than d times the number $k(N^j, I)$ of vertices in $H(N^j)$ in conflict with I . Exercise: Explain why so.
- The expected value $\frac{1}{j} \sum_{S \in N^j} k(N^j, S, I)$ in the $j(= i + 1)$ th step is thus at most $\frac{d \times k(N^j, I)}{j}$, just for the the halfspace I , where $k(N^j, I)$ denotes the number of vertices in $H(N^j)$ in conflict with I .

ABSTRACT CONFIGURATION SPACES (CONTD.)

- But this needs to be summed up over all $I \in N \setminus N^j$, giving the upper bound of Equation 3.2 of [3]

$$\frac{d}{j} \sum_{I \in N \setminus N^j} k(N^j, I) \quad (1)$$

- Looking from another angle, for $j + 1 = i + 2$, the expected value by definition of $k(N^j, S_{j+1})$ is

$$E[k(N^j, S_{j+1})] = \frac{1}{n-j} \sum_{I \in N \setminus N^j} k(N^j, I) \quad (2)$$

because each halfspace in $N \setminus N^j$ is equally likely to occur as S_{j+1} .

- Now we can write Equation 1 using Equation 2 above as

$$\frac{d}{j} (n-j) E[k(N^j, S_{j+1})]$$

ABSTRACT CONFIGURATION SPACES (CONTD.)

- Exercise: Note that the conflicting vertices as in $k(N^j, S_{j+1})$ are deleted when S_{j+1} is processed. So, the same cost can be written as the expectation of

$$\sum_{\sigma} d \frac{n - (j(\sigma) - 1)}{j(\sigma) - 1}$$

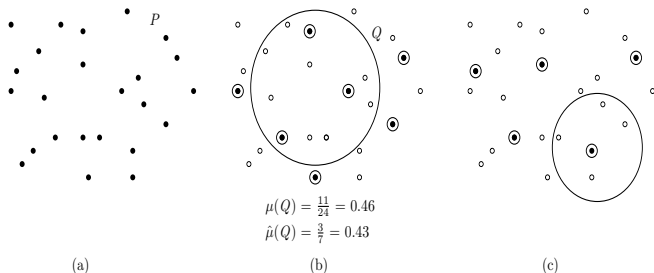
where σ varies over all vertices created (eventually destroyed) and $j(\sigma)$ is the time instance of destruction of σ .

- The fraction $\frac{n - (j(\sigma) - 1)}{j(\sigma) - 1}$ can again be written as $\frac{n - i(\sigma)}{i(\sigma)}$ where $i(\sigma)$ is the time instant when σ was created. Exercise: Explain why.
- So, we rewrite the expected cost bound as the expectation of $\sum_{\sigma} d \frac{n - i(\sigma)}{i(\sigma)} = \sum_{i=1}^n d \frac{n-j}{j} \times$ the expected number of vertices created at time $j = \sum_{\sigma} d \frac{n - i(\sigma)}{i(\sigma)} = \sum_{i=1}^n d \frac{n-j}{j} d \frac{e(j)}{j}$. See Lemma 3.2.1 from [3] for the bound on the expected number $\frac{d}{j} e(j)$ of created vertices.
- Exercise: Show that $e(j) = O(j)$ in this case of polytope computation.
- Exercise: Derive Theorem 3.2.2 based on Lemma 3.2.1 in [3].

RANGE SPACES [2, 1, 3, 4]

- Given a set P of n points in d -dimensional real space \mathbb{R}^d , the power set 2^P of P is the set of all the $2^{|P|} = 2^n$ subsets of P .
- Limiting our attention to geometrically defined subsets alone, we can consider considerably smaller subsets than the power set of P .
- Consider geometric set systems as follows. A *range space* is defined to be a pair (X, \mathcal{R}) , where X is an arbitrary set, not necessarily finite, and \mathcal{R} is a subset of the power set of X .
- Given a set $P \subseteq X$, define the projection of \mathcal{R} on P as $\mathcal{R}|_P = \{P \cap Q \mid Q \in \mathcal{R}\}$.
- Let $X = \mathbb{R}^d$ and P be a set of n points in \mathbb{R}^d . Let \mathcal{R} consist of the subsets of real space contained within axis-parallel rectangles. Observe that $\mathcal{R}|_P$ therefore has the subsets of P within the rectangles.

RANGE SPACES: MEASURES AND SAMPLES

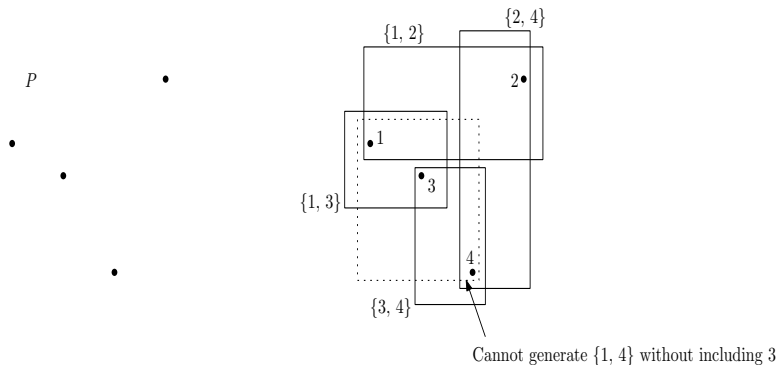
 ϵ -samples and ϵ -nets.

Given a range space (P, R) and any $\epsilon > 0$, a subset $S \subseteq P$ is an ϵ -net if for any range $Q \in R$, if $\mu(Q) \geq \epsilon$ then Q contains at least one point of S . For example, if $\epsilon = 0.2$ and $|P| = 24$, then any range Q that contains at least $0.2 \times 24 \approx 5$ points of P must contain at least one point of the ϵ -net (See Fig. (c)).

RANGE SPACES: MEASURES AND SAMPLES

- Not all subsets of P may lie in $\mathcal{R}|_P$. For example, if the point set $\{1, 4\}$ defines the opposite corners of an axis-oriented rectangle then for any point 3 inside that rectangle the set $\{1, 3, 4\}$ is in \mathcal{R}_P but not the set $\{1,4\}$.
- Given a range space (P, \mathcal{R}) and any $\epsilon > 0$, a subset $S \subseteq P$ is an ϵ -net if for any range $Q \in \mathcal{R}$, if $\mu(Q) \geq \epsilon$ then Q contains at least one point of S . See [2]. For example, if $\epsilon = 0.2$ and $|P| = 25$, then any range Q that contains at least $0.2 \times 25 = 5$ points of P must contain at least one point of the ϵ -net.
- The μ *measure* function over a subset Q of P is the fraction of points in Q with respect to P . Instead of the whole set P we may take a subset S of P and find the fraction of the points in Q that are in S . See [2].
- This approximation of $\mu(Q)$ is termed $\mu'(Q)$. We would like this approximation to be bounded by ϵ . We call such samples S of P ϵ -samples; we will see that such ϵ -samples (or *epsilon*-approximations) are ϵ -nets.

RANGE SPACES: SHATTERING



A 4-point set and the range space of axis-parallel rectangles.

RANGE SPACES: SHATTERING

- Suppose that we do not need such a good estimate like an ϵ -sample, but we just wish that any sufficiently large group of the population should contribute at least one member to the sample.
- An ϵ -sample always does so but we may simply use an ϵ -net instead.
- Given an arbitrary range space (X, \mathcal{R}) and finite point set P , we say that \mathcal{R} shatters P if $\mathcal{R}|_P$ is equal to the power set of P , that is, we can form any of the $2^{|P|}$ subsets of P by taking intersections of P with the ranges of \mathcal{R} .
- Note that if the range space is of axis-parallel rectangles then no 5-point set can be shattered. However, there are 4-point subsets that can be shattered.
- Similarly, 3-point subsets can be shattered by the range space of circles but no 4-point subset can be shattered.

RANGE SPACES: VC-DIMENSIONS: SAUER'S LEMMA. SEE [2].

- Sauer's lemma says that the size of the range space \mathcal{R} , is bounded by $\phi_d(n)$, where d is the VC-dimension of the range space defined on n points. This function $\phi_d(n)$ is the number $\sum_{i=1}^d \binom{n}{i}$ of subsets of size at most d over a ground set of size n .
- We show that $\phi_d(n) = \phi_d(n-1) + \phi_{d-1}(n-1)$ by induction on n and d . If a subset has a fixed element x then we need choose only $d-1$ more elements from the remaining $n-1$ elements, and if it does not have x then we need to choose all the d elements from the remaining $n-1$ elements.
- The proof of Sauer's lemma is by induction on d and n . It holds trivially for $d=0$ or $n=0$.
- Let \mathcal{R}_x be formed from pairs of ranges from \mathcal{R} that are identical except that one contains x and the other does not.
- The set $\mathcal{R} \setminus \{x\}$ is the result of throwing x entirely out of the point set and considering the induced sub-hypergraph of \mathcal{R} .

RANGE SPACES: VC-DIMENSIONS: SAUER'S LEMMA. SEE [2].

- We show that $|\mathcal{R}| = |\mathcal{R}_x| + |\mathcal{R} \setminus \{x\}|$.
- Charge each range of \mathcal{R} to its corresponding range in $\mathcal{R} \setminus \{x\}$. Every range of $\mathcal{R} \setminus \{x\}$ receives at least one charge, but it receives two charges if there exist two ranges that are identical except that one contains x and one does not have x .
- These extra charges are the elements of \mathcal{R}_x .
- The VC-dimension of $\mathcal{R} \setminus \{x\}$ cannot be larger than that of \mathcal{R} ; it is at most d .
- The range space $(X \setminus \{x\}, \mathcal{R}_x)$ has VC-dimension $d - 1$; no set P' of size d can be shattered. If this was the case and we were to return x back into P' , then pairs of ranges of \mathcal{R} that gave rise to the ranges of \mathcal{R}_x would then shatter the $d + 1$ element set $P' \cup \{x\}$.
- Now apply the induction hypothesis on the ϕ upper bounds for VC-dimensions of $(X \setminus \{x\}, \mathcal{R}_x)$ and $\mathcal{R} \setminus \{x\}$ to bound the cardinality of \mathcal{R} by $\phi_d(n)$.

THE METHOD OF RANDOM SAMPLING FOR A SET SYSTEM OR HYPERGRAPH [5]

- Take any set system (hypergraph) $G(V, S)$ where $V = \{v_1, \dots, v_n\}$ and $S = \{e_1, \dots, e_m\}$; here, $e_i \subseteq V$ for all $1 \leq i \leq m$.
- Given any integer $1 \leq r \leq n$, we wish to find a subset $N \subseteq V$ that intersects every e_i of size greater than $\frac{n}{r}$.
- We can assume that $|e_i| > \frac{n}{r}$, for any i , and $m > 1$.
- Let $p = \frac{cr(\log m)}{n}$, for some large enough constant c .
- Sample the set V by Binomial distribution, that is, construct the set N by including in it v_i with probability p .
- Then, $N \cap e_i = \phi$ with probability less than $(1 - p)^{\frac{n}{r}}$ for a single value of i .
- The probability that N does not intersect some e_i is less than $m(1 - p)^{\frac{n}{r}}$.

BINOMIAL RANDOM SAMPLING OF THE SET OF VERTICES

- We can make this probability smaller than any constant. Why?
- The probability can be shown to be bounded by $\frac{1}{m^{c-1}}$, which can be made smaller than any given limiting constant as m grows beyond a certain value for each valid choice of the constant c .
- So, the sample N of expected size $np = cr \log m$ intersects every set e_i of size $\frac{n}{r}$; note also that the random sample N is of size $O(r \log m)$ with high probability.
- Therefore, we have a way of getting random samples such as N of size $O(r \log m)$ with high probability, so that N intersects all the sets e_i of size greater than $\frac{n}{r}$.
- This was a simple randomized construction of such a set N (with high probability), that intersects all the sets of size not lesser than $\frac{n}{r}$.
- We now discuss a greedy deterministic method, and later we consider another method, which is a derandomization of the above randomized sampling technique.

DETERMINISTIC CONSTRUCTION OF ϵ -NETS [5]

- We state a greedy and deterministic way of generating an ϵ -net N as follows.
- Find a vertex v_i contained in most sets $e_j \subseteq S$.
- Remove this vertex from further consideration and add it to N .
- Then, remove all sets containing v_i from future consideration.
- Repeat these steps until all hyperedges from S are removed.
- Show that this algorithm can be made to run in $O(mn)$ time.

DETERMINISTIC CONSTRUCTION OF ϵ -NETS

- Does N turn out to be quite small as required?
- Let m_k be the number of hyperedges remaining after k iterations. Clearly, $m_0 = m$.
- Sticking to the assumption that each set e_i has at least $\frac{n}{r}$ elements, we now have m_k sets left after the k th iteration with at least $\frac{n}{r}$ elements.
- We can select any of the $n - k$ remaining vertices in the next iteration.
- We have a distribution of $n - k$ distinct vertices in at least $m_k \times \frac{n}{r}$ instances over the m_k sets.
- So, the most frequent vertex of the m_k sets must be in at least $\frac{m_k \times \frac{n}{r}}{n-k} \geq \frac{m_k}{r}$ sets. Why?

DETERMINISTIC CONSTRUCTION OF ϵ -NETS

- Hint: What is the expected number of vertices in these sets?
- Thus, $m_{k+1} \leq m_k(1 - \frac{1}{r})$.
- So, $m_k \leq m(1 - \frac{1}{r})^k$.
- For a large enough constant $c > 0$, and any $k \geq cr \log m$, we have $m_k < 1$, and therefore $m_k = 0$.
- In other words, picking any sufficiently large number $k \geq cr \log m$ of vertices we can ensure that we hit all the hyperedges that have at least $\frac{n}{r}$ vertices.
- So, we can deterministically compute a $\frac{1}{r}$ -net N of size $O(r \log m)$, greedily as above. See Theorem 4.3 in Chazelle's book [5].

ϵ -NETS, INFINITE SETS, MEASURES [1]

- We have seen the measure μ and its approximation μ' , defining ϵ -samples or ϵ -approximations in [2]. To deal with finite subsets in infinite continuous spaces, we need such measures.
- For infinite sets we cannot measure the size as the number of points; suppose μ is concentrated on finitely many points in a finite set $Y \subseteq X$ and there is a positive function w mapping $w(y)$ into $(0, 1]$, for each $y \in Y$, with normalization $\sum_{y \in Y} w(y) = 1$; so μ is given by $\mu(A) = \sum_{y \in A \cap Y} w(y)$, for any $A \subseteq Y$.
- So, a uniform measure over Y would assign weights $\frac{1}{|Y|}$ to each point $y \in Y$.
- Let μ be a *probability measure* on a set X , and let \mathcal{F} be a system of μ -measurable subsets of X . For ϵ in $[0, 1]$, a subset $N \subseteq X$ is called an ϵ -net for range space (X, \mathcal{F}) with respect to μ if N meets S for all $S \in \mathcal{F}$ with $\mu(S) \geq \epsilon$. Definition 10.2.2 in [1].
- The ϵ -net Theorem 10.2.4 in [1] due to Haussler and Welzl states that there is a $\frac{1}{r}$ -net for (X, \mathcal{F}) with respect to μ of size at most $Cdr \ln r$, where C is an *absolute constant*.

ϵ -NETS, INFINITE SETS, MEASURES (CONTD.) [1]

- The size of this set is independent of the sizes of X and \mathcal{F} and d is the upper bound on the VC-dimension $\dim(\mathcal{F})$.
- Firstly, we can drop all S from \mathcal{F} that satisfy $\mu(S) < \frac{1}{r}$ as these sets S do not matter. Also, the probability $(1 - \frac{1}{r})^s$ of a random sample N of at least $s = r \ln(|\mathcal{F}| + 1)$ items, missing a fixed set out of the \mathcal{F} sets is at most $\frac{1}{|\mathcal{F}|+1}$, and thus the probability of missing "some" set out of the \mathcal{F} sets is at most $\frac{|\mathcal{F}|}{|\mathcal{F}|+1} < 1$.
- So, we can have such a big $\frac{1}{r}$ -net ! However, we need a smaller one of size $s = Cdr \ln r$. So we take a random sample N of size s , where each choice is drawn independently from X using the probability distribution μ . We also take another random sample M of similar size and construction. It really does not matter whether or not μ is a *uniform measure* with each element in Y getting the weight $\frac{1}{|Y|}$.
- Event E_0 is that N misses some set in \mathcal{F} . Event E_1 is that E_0 happens with N not meeting some $S \in \mathcal{F}$ as well as S meeting M in at least $k = \frac{s}{2r}$ points.
- It suffices to show that $\text{Prob}[E_1] \leq \text{Prob}[E_0] \leq 2\text{Prob}[E_1]$, and also that $\text{Prob}[E_1] < \frac{1}{2}$. Why?

ϵ -NETS, INFINITE SETS, MEASURES (CONTD.) [1]

- For n trials, the expectation $E(X) = E(X_1 + X_2 + \dots + X_n)$ is np (say ≥ 8) and variance $npq \leq np$, where $p + q = 1$. Here, we set $n = s = Cdr \ln r$ and $p = \frac{1}{r}$. By Chebyshev's inequality we have $Prob[X < \frac{1}{2}np = \frac{1}{2}Cdr \ln r] \leq Prob[|X - E(X)| \geq \frac{1}{2}np] \leq \frac{np}{(\frac{1}{2}np)^2} = \frac{4}{np} \leq \frac{1}{2}$; consequently, $Prob[X \geq \frac{1}{2}np] \geq \frac{1}{2}$.
- $Prob[E_1] \leq Prob[E_0]$ as E_1 requires E_0 to happen.
- If N is fixed and M thus random and N is a $\frac{1}{r}$ -net then $Prob[E_1/N] = 0$ because E_1 cannot occur.
- However, if $S \in \mathcal{F}$ with $N \cap S = \phi$, fix any such S as S_N .
- So, $Prob[E_1/N] \geq Prob[|M \cap S_N| \geq k = \frac{s}{2r} = \frac{1}{2}np] \geq \frac{1}{2}$ for all such N . Each of these k matches is in a hyperedge S_N with $\mu(S_N) \geq \frac{1}{r}$, where these k matches may be viewed as successes in binomial sampling s times with k successes, each with probability at least $\frac{1}{r}$.
- Since E_0 happens for N , $Prob[E_0/N] \leq 1$ and so it is at most $2Prob[E_1/N]$ for all N , and thus $Prob[E_0] \leq 2Prob[E_1]$.
- What remains to show now is that $Prob[E_1] < \frac{1}{2}$, as this will show $Prob[E_0] < 1$.

ϵ -NETS, INFINITE SETS, MEASURES (CONTD.) [1]

- The draws M and N can be viewed alternatively as an initial random sample $A = (z_1, z_2, \dots, z_{2s})$ of $2s$ draws from X , and then fix N as one of the $\binom{2s}{s}$ combinations, and choose the favourable probability of $\frac{\binom{2s-k}{s}}{\binom{2s}{s}}$ as an upper bound for the conditional probability P_S for a specific S missed by N and M meeting at least k elements of the same set S . Note that P_S is at most the probability of the set N of s items missing S entirely, and thus at least k items in A of S , which go into the s items in M . Why? The number of ways to choose s items in N from A is $\binom{2s}{s}$, and the number of ways this choice is done avoiding k items is $\binom{2s-k}{s}$.
- So, $P_S \leq \frac{\binom{2s-k}{s}}{\binom{2s}{s}} \leq (1 - \frac{k}{2s})^s \leq e^{-\frac{k}{2s} s} = e^{-\frac{k}{2}} = e^{-\frac{Cd \ln r}{4}} = r^{-\frac{Cd}{4}}$
- This was for a fixed $S \in \mathcal{F}$ missed by N given a fixed A of $2s$ sized sample. So, releasing S to be any of $\Phi_d(2s)$ possibilities for a fixed A of (X, \mathcal{F}) , we use Sauer's lemma to show that $\text{Prob}[E1/A] \leq (\sum_{i=1}^d \binom{2s}{i}) r^{-\frac{Cd}{4}} \leq (\frac{2es}{d})^d r^{-\frac{Cd}{4}} < \frac{1}{2}$, as $C = 1$, for $d > 2$, beyond all $r > r_0$ for some r_0 [1].

RANDOM SAMPLING IN GEOMETRY [3, 7]

- We have n objects in the set N , and subsets of N can be the *defining* elements of *configurations* comprising the set $\Pi = \Pi(N)$ of configurations.
- Let $\sigma \in \Pi$ be one such configuration.
- Imagine a one-dimensional space (the real line) with n distinct points and the pairs of n points as configurations, which are actually linear intervals.
- If we fix a constant $r < n$, we may take a *random sample* R of r elements, selected out of the n elements in N .
- Sampling is done without repetitions; each time an element is selected independently and randomly.
- Here, the cardinality $d(\sigma)$ of the set $D(\sigma)$ of *triggers* or *defining elements* of any of these configurations σ is exactly 2.
- Each configuration σ may contain (intersect) a set $L(\sigma)$ of elements from N called *stoppers*. The number of stoppers is denoted by $l(\sigma)$ and is called the *conflict size*.

RANDOM SAMPLING IN GEOMETRY

- We say that $\sigma \in \Pi$ is active over a subset $R \subseteq N$ if it occurs as an interval in $H(R)$, the partition formed on the line by R .
- This occurs if and only if R contains all the points defining σ but no point in conflict with σ .
- Such configurations are called *active configurations* of Π over the random sample R .
- We show (as in Theorem 5.0.1 from [3]) that with probability at least $\frac{1}{2}$, every active configuration over the random sample R of cardinality r , would have conflict size $O(\frac{n}{r} \log r)$.
- Let $p(\sigma, r)$ denote the conditional probability that R has no point in conflict with σ , given that R contains the points defining σ

$$p(\sigma, r) \leq \left(1 - \frac{l(\sigma)}{n}\right)^{r-d(\sigma)} \quad (3)$$

RANDOM SAMPLING IN GEOMETRY

- The intuitive justification is as follows. The interval being of conflict size $l(\sigma)$, the probability of choosing a conflicting point is at least $\frac{l(\sigma)}{n}$.
- Since we select $r - d(\sigma)$ points without conflicts, the probability required is upper bounded as in Inequality 3.
- However, since $1 - x \leq \exp(-x)$ where $\exp(x) = e^x$, we have

$$p(\sigma, r) \leq \exp\left(-\frac{l(\sigma)}{n}(r - d(\sigma))\right) \quad (4)$$

- Since $d(\sigma) \leq 2$, putting $l(\sigma) \geq c(n \ln s)/(r - 2)$ for some $c > 1$ and $s \geq r$, we get

$$p(\sigma, r) \leq \exp(-c \ln s) = \frac{1}{s^c} \quad (5)$$

- Now an active configuration σ due to the random sample R must be such that all its defining points must be in R .

RANDOM SAMPLING IN GEOMETRY

- In other words, $\sigma \in \Pi(R)$.
- Let this probability be $q(\sigma, r)$.
- The probability that σ is an active configuration in the partition created by the random sample R is at most

$$p(\sigma, r)q(\sigma, r)$$

- The probability that there is some active configuration created by the partition due to the random sample R with conflict size lower bounded by $\frac{cn \ln s}{r-2}$, is upper bounded by the sum of the probabilities over all such configurations

$$\sum_{\sigma \in \Pi: l(\sigma) > \frac{cn \ln s}{r-2}} p(\sigma, r)q(\sigma, r)$$

RANDOM SAMPLING IN GEOMETRY

$$\leq \sum_{\sigma \in \Pi: l(\sigma) > \frac{cn \ln s}{r-2}} q(\sigma, r) / s^c \leq \frac{1}{s^c} \sum_{\sigma \in \Pi} q(\sigma, r)$$

- Now the last summation in the above inequality is $E(\pi(R))$ and $\pi(R) = |\Pi(R)| = O(r^2)$. So, choosing $c > 2$ we can ensure that the probability of having a “long” active configuration in $\sigma \in \Pi(R)$ is less than $\frac{1}{2}$ for a random sample R (Theorem 5.0.1 from [3]).

THE WEAK CUTTING LEMMA 4.6.1 IN [1]

- With n lines in the plane, we can have at most $O(n^2)$ cells, and a lower bound of at least $\frac{n^2}{2}$ cells. Let L be this set of n lines.
- Any triangle that cuts k lines is divided into at most $2k^2$ parts.
- Suppose we have only t (arbitrary) triangles partitioning the whole plane containing the arrangement of n lines, and each such triangle is cut by at most k of the n given lines.
- Since each of these at least $\frac{n^2}{2}$ cells has to be covered by only t triangles, each of which has at most $2k^2$ cells as stated above, we need to have $t \geq \frac{n^2}{4k^2} = \Omega(r^2)$ triangles, provided we fix $k \leq \frac{n}{r}$.
- We now show that a set of $O(r^2 \log^2 n)$ triangles can be used to ensure that less than $\frac{n}{r}$ lines of the arrangement of n lines cross each such triangle.
- Use a random sample $S \subset L$ of size $s = 6r \log n$ to create $O(s^2)$ regions as follows. If there are non-triangular regions, we triangulate them.
- A *bad* triangle T (defined by any three lines of the n lines in L) has strictly more than $k = \frac{n}{r}$ lines intersecting T .

THE WEAK CUTTING LEMMA (CONTD.)

- There are at most n^6 interesting triangles since each triangle is defined by three of the $\binom{n}{2}$ points of intersections as vertices.
- The probability that T is a bad triangle is less than n^{-6} if we choose $s = 6r \log n$. Why?
- So, the probability that some interesting triangle is bad is *strictly less* than unity, as we show now.
- This probability is strictly less than

$$n^6 \left(1 - \frac{k}{n}\right)^s \leq n^6 \left(1 - \frac{1}{r}\right)^{6r \ln n} < n^6 e^{-6 \ln n} = n^6 n^{-6} = 1$$
- Therefore, there exists a random sample S of size $s = 6r \log n$ such that the none of the $O(s^2)$ triangles induced by S meet more than $\frac{n}{r}$ lines of L .
- This can be used to design data structures for searching in an arrangement of lines.

2D LINEAR PROGRAMMING IN EXPECTED LINEAR TIME [9]

- The “sickle” (Section 7.2.1) method or the “slab” method (Sections 2.2.2.1 and 7.2.1), recursively computes the common intersection of halfplanes in the 2d plane [8].

2D LINEAR PROGRAMMING IN EXPECTED LINEAR TIME [9]

- The “sickle” (Section 7.2.1) method or the “slab” method (Sections 2.2.2.1 and 7.2.1), recursively computes the common intersection of halfplanes in the 2d plane [8].
- We can represent the sequence of the common intersections of the constraint halfplanes as $C_0, C_1, C_2, \dots, C_n$, as we process the halfplanes h_1, h_2, \dots, h_n , even if we do not compute these common intersections.

2D LINEAR PROGRAMMING IN EXPECTED LINEAR TIME [9]

- The “sickle” (Section 7.2.1) method or the “slab” method (Sections 2.2.2.1 and 7.2.1), recursively computes the common intersection of halfplanes in the 2d plane [8].
- We can represent the sequence of the common intersections of the constraint halfplanes as $C_0, C_1, C_2, \dots, C_n$, as we process the halfplanes h_1, h_2, \dots, h_n , even if we do not compute these common intersections.
- The current optimal for the 2d linear program v_{i-1} changes giving a new optimal v_i if and only if the new halfplane h_i does not contain v_{i-1} (see Lemma 4.5 in [9]).

2D LINEAR PROGRAMMING IN EXPECTED LINEAR TIME [9]

- The “sickle” (Section 7.2.1) method or the “slab” method (Sections 2.2.2.1 and 7.2.1), recursively computes the common intersection of halfplanes in the 2d plane [8].
- We can represent the sequence of the common intersections of the constraint halfplanes as $C_0, C_1, C_2, \dots, C_n$, as we process the halfplanes h_1, h_2, \dots, h_n , even if we do not compute these common intersections.
- The current optimal for the 2d linear program v_{i-1} changes giving a new optimal v_i if and only if the new halfplane h_i does not contain v_{i-1} (see Lemma 4.5 in [9]).
- Since we randomly permute the sequence of constraints, there is a change in the current optimal in the i st step if one of the two defining lines for the new optimal is the i st constraint.

2D LINEAR PROGRAMMING IN EXPECTED LINEAR TIME [9]

- This happens only if one of the two halfplanes defining the edges incident on the current optimal v_i is h_i (which happens with probability $\frac{2}{i-2}$), provided more than two halfplane boundaries do pass through v_i (Theorem 4.8 in [9]).

2D LINEAR PROGRAMMING IN EXPECTED LINEAR TIME [9]

- This happens only if one of the two halfplanes defining the edges incident on the current optimal v_i is h_i (which happens with probability $\frac{2}{i-2}$), provided more than two halfplane boundaries do pass through v_i (Theorem 4.8 in [9]).
- The random permutation renders each of the i halfplanes to appear as the i th constraint in the processing of constraints with equal probability.

2D LINEAR PROGRAMMING IN EXPECTED LINEAR TIME [9]

- This happens only if one of the two halfplanes defining the edges incident on the current optimal v_i is h_i (which happens with probability $\frac{2}{i-2}$), provided more than two halfplane boundaries do pass through v_i (Theorem 4.8 in [9]).
- The random permutation renders each of the i halfplanes to appear as the i th constraint in the processing of constraints with equal probability.
- In cases the current optimal shifts, we need to find its location on the bounding line of h_i in $O(i)$ time, solving a 1d linear program ! See Lemma 4.6 and Figure 4.6 in [9].

2D LINEAR PROGRAMMING IN EXPECTED LINEAR TIME [9]

- This happens only if one of the two halfplanes defining the edges incident on the current optimal v_i is h_i (which happens with probability $\frac{2}{i-2}$), provided more than two halfplane boundaries do pass through v_i (Theorem 4.8 in [9]).
- The random permutation renders each of the i halfplanes to appear as the i th constraint in the processing of constraints with equal probability.
- In cases the current optimal shifts, we need to find its location on the bounding line of h_i in $O(i)$ time, solving a 1d linear program ! See Lemma 4.6 and Figure 4.6 in [9].
- Whence, the expected running time is $\sum_{i=1}^n (\frac{2}{i-2}) O(i) = O(n)$.

COMPUTING THE VORONOI DIAGRAM [8]

- Theorem 5.8 [8] says that every vertex of the Voronoi diagram must be the circumcentre of a circle with three sites defining the circle, such that the circle contains no site inside it.

COMPUTING THE VORONOI DIAGRAM [8]

- Theorem 5.8 [8] says that every vertex of the Voronoi diagram must be the circumcentre of a circle with three sites defining the circle, such that the circle contains no site inside it.
- Theorem 5.9 says that Voronoi edges are defined by neighbouring pairs of sites.

COMPUTING THE VORONOI DIAGRAM [8]

- Theorem 5.8 [8] says that every vertex of the Voronoi diagram must be the circumcentre of a circle with three sites defining the circle, such that the circle contains no site inside it.
- Theorem 5.9 says that Voronoi edges are defined by neighbouring pairs of sites.
- Theorem 5.10 says that all unbounded regions correspond to exactly the convex hull vertices.

COMPUTING THE VORONOI DIAGRAM [8]

- Theorem 5.8 [8] says that every vertex of the Voronoi diagram must be the circumcentre of a circle with three sites defining the circle, such that the circle contains no site inside it.
- Theorem 5.9 says that Voronoi edges are defined by neighbouring pairs of sites.
- Theorem 5.10 says that all unbounded regions correspond to exactly the convex hull vertices.
- See Figures 5.25 (a) and (b) of the two vertically separated subsets of roughly equal sizes, along with their respective Voronoi diagrams, overlapping in the plane.

COMPUTING THE VORONOI DIAGRAM [8]

- Theorem 5.8 [8] says that every vertex of the Voronoi diagram must be the circumcentre of a circle with three sites defining the circle, such that the circle contains no site inside it.
- Theorem 5.9 says that Voronoi edges are defined by neighbouring pairs of sites.
- Theorem 5.10 says that all unbounded regions correspond to exactly the convex hull vertices.
- See Figures 5.25 (a) and (b) of the two vertically separated subsets of roughly equal sizes, along with their respective Voronoi diagrams, overlapping in the plane.
- Figures 5.26, 5.27 and 5.28 depict the merging step that computes the convex hull of the entire set of points from the two hulls in Figure 5.25.

COMPUTING THE VORONOI DIAGRAM [8] CONTD.

- We start with the (perpendicular) bisector of the vertices of the upper common tangent and move downwards towards the edges of the two hulls along the (would-be) zig-zag polyline σ .

COMPUTING THE VORONOI DIAGRAM [8] CONTD.

- We start with the (perpendicular) bisector of the vertices of the upper common tangent and move downwards towards the edges of the two hulls along the (would-be) zig-zag polyline σ .

COMPUTING THE VORONOI DIAGRAM [8] CONTD.

- We start with the (perpendicular) bisector of the vertices of the upper common tangent and move downwards towards the edges of the two hulls along the (would-be) zig-zag polyline σ .
- This polyline σ is made up of bisectors of sites from opposite sides, thus defining the final convex hull, cutting and pruning off some edges of both previous Voronoi diagrams.

COMPUTING THE VORONOI DIAGRAM [8] CONTD.

- We start with the (perpendicular) bisector of the vertices of the upper common tangent and move downwards towards the edges of the two hulls along the (would-be) zig-zag polyline σ .
- This polyline σ is made up of bisectors of sites from opposite sides, thus defining the final convex hull, cutting and pruning off some edges of both previous Voronoi diagrams.
- The downwards ride along σ has one or more right (left) turns when it crosses edges of the right (left) hull, keeping the left (right) hull vertex unchanged for the new bisector and switching over to a neighbour of the current right (left) hull vertex of the current bisector, for the new edge (bisector) along σ .

COMPUTING THE VORONOI DIAGRAM [8] CONTD.

- A clockwise (counterclockwise) traversal of the current left (right) hull face prudently avoids repeated scans for intersection points of the changing right (left) turning edges of σ for the final exit, a left (right) turn, on the current left (right) hull face with a left (right) turn. See Figures 5.28 and 5.29.

COMPUTING THE VORONOI DIAGRAM [8] CONTD.

- A clockwise (counterclockwise) traversal of the current left (right) hull face prudently avoids repeated scans for intersection points of the changing right (left) turning edges of σ for the final exit, a left (right) turn, on the current left (right) hull face with a left (right) turn. See Figures 5.28 and 5.29.
- This kind of traversal permits linear time computation of the path σ . The DCEL data structures of both hulls are merged into a single one for the merged hull with the pruning off of right (left) hull edges on the right (left) sides of σ . The necessary boundary edges of intersected hull faces are inserted as edges, the edges of σ .

ONLINE VORONOI DIAGRAM COMPUTATION, CHAPTER 7 [9]

- We sweep a horizontal line downwards through the n sites and stop at "site" events and other events to take actions.

ONLINE VORONOI DIAGRAM COMPUTATION, CHAPTER 7 [9]

- We sweep a horizontal line downwards through the n sites and stop at "site" events and other events to take actions.
- A "beachline" lower envelope of certain vertical parabolas is maintained as an x-monotone chain.

ONLINE VORONOI DIAGRAM COMPUTATION, CHAPTER 7 [9]

- We sweep a horizontal line downwards through the n sites and stop at "site" events and other events to take actions.
- A "beachline" lower envelope of certain vertical parabolas is maintained as an x -monotone chain.
- The parabolas grow in width from a start that is just a thin segment when the "site" event triggers the respective parabola on.

ONLINE VORONOI DIAGRAM COMPUTATION, CHAPTER 7 [9]

- We sweep a horizontal line downwards through the n sites and stop at "site" events and other events to take actions.
- A "beachline" lower envelope of certain vertical parabolas is maintained as an x -monotone chain.
- The parabolas grow in width from a start that is just a thin segment when the "site" event triggers the respective parabola on.
- The lower envelope of the parabolas has precisely those points p which have minimal vertical distance from the sweepline, from amongst the distances of p from all the sites; the focus of the parabola is a site that is of the same distance from the lower envelope point p , as its vertical distance from the sweepline. The algorithm maintains a portion of the Voronoi diagram which does not change due to the appearance of further new sites below sweep line.

ONLINE VORONOI DIAGRAM COMPUTATION [9]

(CONTD.)

- Consequently, every instant as we sweep, any two neighbouring parabolas must be meeting at a point p of a Voronoi edge; the point p is at minimal distance from the sweep line and also equidistant from two (closest) sites, and thus a point on a Voronoi edge.

ONLINE VORONOI DIAGRAM COMPUTATION [9]

(CONTD.)

- Consequently, every instant as we sweep, any two neighbouring parabolas must be meeting at a point p of a Voronoi edge; the point p is at minimal distance from the sweep line and also equidistant from two (closest) sites, and thus a point on a Voronoi edge.
- If a new arc (parabola) appears on the beach line through an event other than through a site event (like if a new parabola breaches through the beachline, overtaking the beachline), then we can derive a contradiction.

ONLINE VORONOI DIAGRAM COMPUTATION [9]

(CONTD.)

- Consequently, every instant as we sweep, any two neighbouring parabolas must be meeting at a point p of a Voronoi edge; the point p is at minimal distance from the sweep line and also equidistant from two (closest) sites, and thus a point on a Voronoi edge.
- If a new arc (parabola) appears on the beach line through an event other than through a site event (like if a new parabola breaches through the beachline, overtaking the beachline), then we can derive a contradiction.
- There can therefore be at most $2n - 1$ parabolic arcs on the beachline. How?

ONLINE VORONOI DIAGRAM COMPUTATION [9]

(CONTD.)

- Consequently, every instant as we sweep, any two neighbouring parabolas must be meeting at a point p of a Voronoi edge; the point p is at minimal distance from the sweep line and also equidistant from two (closest) sites, and thus a point on a Voronoi edge.
- If a new arc (parabola) appears on the beach line through an event other than through a site event (like if a new parabola breaches through the beachline, overtaking the beachline), then we can derive a contradiction.
- There can therefore be at most $2n - 1$ parabolic arcs on the beachline. How?
- A shrinking arc that corresponds to a site which has distinct neighbours, whose respective flanking sites define converging Voronoi edges (edges reaching out to meet at a Voronoi vertex), vanishes. This is called a "circle" event.

ONLINE VORONOI DIAGRAM COMPUTATION [9]

(CONTD.)

- These three sites define their circumcentre (a Voronoi vertex), of an empty circle.

ONLINE VORONOI DIAGRAM COMPUTATION [9]

(CONTD.)

- These three sites define their circumcentre (a Voronoi vertex), of an empty circle.
- Now we will see how the steps for incrementally updating the Voronoi diagram on each event is aided by three different kinds of data structures- the DCEL, the dynamically balanced binary search tree, and a priority queue.

ONLINE VORONOI DIAGRAM COMPUTATION [9]

(CONTD.)

- These three sites define their circumcentre (a Voronoi vertex), of an empty circle.
- Now we will see how the steps for incrementally updating the Voronoi diagram on each event is aided by three different kinds of data structures- the DCEL, the dynamically balanced binary search tree, and a priority queue.
- All Voronoi vertices are detected at circle events; a circle event created may however be a false alarm and in that case will be destroyed subsequent to its creation. Why?

ONLINE VORONOI DIAGRAM COMPUTATION [9]

(CONTD.)

- These three sites define their circumcentre (a Voronoi vertex), of an empty circle.
- Now we will see how the steps for incrementally updating the Voronoi diagram on each event is aided by three different kinds of data structures- the DCEL, the dynamically balanced binary search tree, and a priority queue.
- All Voronoi vertices are detected at circle events; a circle event created may however be a false alarm and in that case will be destroyed subsequent to its creation. Why?
- The tree that stores the beach line permits insertion/deletion/splitting of parabolas in logarithmic time, where the parabolas are stored symbolically, not explicitly. Why?

ONLINE VORONOI DIAGRAM COMPUTATION [9]

(CONTD.)

- These three sites define their circumcentre (a Voronoi vertex), of an empty circle.
- Now we will see how the steps for incrementally updating the Voronoi diagram on each event is aided by three different kinds of data structures- the DCEL, the dynamically balanced binary search tree, and a priority queue.
- All Voronoi vertices are detected at circle events; a circle event created may however be a false alarm and in that case will be destroyed subsequent to its creation. Why?
- The tree that stores the beach line permits insertion/deletion/splitting of parabolas in logarithmic time, where the parabolas are stored symbolically, not explicitly. Why?
- After all site events and circle events are done, the beachline may continue with the infinite edges of open cells of the Voronoi diagram. The total cost is $O(n \log n)$

HALFPLANE INTERSECTIONS AND VORONOI DIAGRAMS [9] CHAPTER 11

- If all points q closest to a site p (and no other points) can be lifted to a halfplane $h(p)$ corresponding to the point p , then that part of $h(p)$ can be projected back to the Voronoi region for p .
- So, all such halfplanes can be used to compute their upper halfplanes' intersections.
- Points of other Voronoi cells would land up in other similar halfplanes.
- This is brought about by Theorem 11.8 in [9] by showing first that $q(r)$ lies lower than $q(p)$ for all sites $r \neq p$ by using the fact that $q(p)q$ distance is the square of the distance pq .
- Let $z = x^2 + y^2$ denote the unit paraboloid U . Let $p = (p_x, p_y, 0)$. Consider the vertical line through p intersecting U in the point $p' = (p_x, p_y, p_x^2 + p_y^2)$. Let $h(p)$ be the non-vertical plane $z = 2p_x x + 2p_y y - (p_x^2 + p_y^2)$. Notice that $h(p)$ contains the point p' .

HALFPLANE INTERSECTIONS AND VORONOI DIAGRAMS

- Like p and p' we define for any point q in the Voronoi cell of p , the points q' on the paraboloid and a point $q(p)$ on the plane $h(p)$, and show that $q'q(p) = \text{dist}^2(p, q)$.
- We know that $\text{dist}(q, p) < \text{dist}(q, r)$ for all $r \in P$ with $r \neq p$. We show that the vertical line through q intersects the upper envelope of the planes at a point lying on $h(p)$.
- Recall that for a point $r \in P$, the plane $h(r)$ is intersected by the vertical line through q at the point $q(r) = (q_x, q_y, q_x^2 + q_y^2 - \text{dist}^2(q, r))$.
- Of all points in P , the point p has the smallest distance to q , so $q(p)$ is the highest intersection point.
- Hence, the vertical line through q intersects the upper envelope at a point lying on $h(p)$.

ONLINE CONVEX HULL COMPUTATION [8]

We need to maintain the convex hull as we add the new point p_i to the convex hull $CH(S_{i-1})$, yielding $CH(S_i)$, where $S_i = S_{i-1} \cup \{p_i\}$. (Pages 119-124, Theorem 3.11)

The new point p_i can be anywhere in the plane.

One way is to compute the two tangents from p_i to $CH(S_{i-1})$.

We show how we can use *concatenable queues* for storing convex hulls to achieve this goal in $O(\log n)$ time where n is the total number of points processed.

Let m be the left end, M be the median or the middle point and α be the counterclockwise angle $\angle mp_iM$.

The first four cases have α convex ($\leq \pi$) and the last four have it reflex ($> \pi$) (See Figure 3.16 [8]).

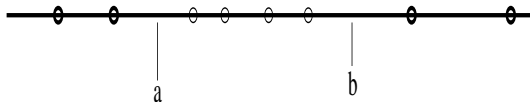
ONLINE CONVEX HULL COMPUTATION (CONTD.) [8]

If m and M are both concave (or nonconcave-reflex) incidences, then the same subtree $R(M)$ ($L(M)$), will receive both tangents (Cases (1) and (3), Figure 3.16).

Cases 1 and 2 are complementary about M , with M turning nonconcave from concave, so the right tangent remains on $R(M)$ (as in Case 1) but the left tangent goes to $T - R(M)$, including M . Case 4 changes m from concave in Case 1 to nonconcave, making the right tangent land on $L(M)$. Also M becomes nonreflex, that is, either supporting or reflex, making the left tangent land on $T - L(M)$. Cases 5-8 are just replicas of Case 1-4 with roles of m and M exchanged and the $\angle mp_i M$ being reflex.

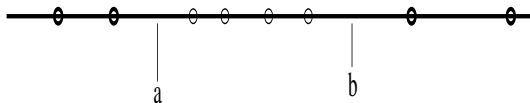
Note that in all the eight cases we can discard search on one half of the concatenable queue for either tangent, making the search for both tangents logarithmic.

1-DIMENSIONAL RANGE SEARCHING



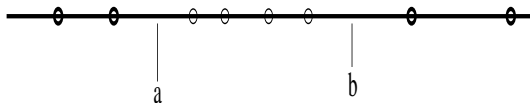
- Problem: Given a set P of n points $\{p_1, p_2, \dots, p_n\}$ on the real line, report points of P that lie in the range $[a, b]$, $a \leq b$.

1-DIMENSIONAL RANGE SEARCHING



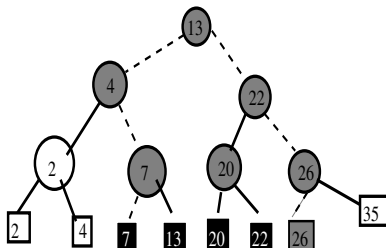
- Problem: Given a set P of n points $\{p_1, p_2, \dots, p_n\}$ on the real line, report points of P that lie in the range $[a, b]$, $a \leq b$.
- Using binary search on an array we can answer such a query in $O(\log n + k)$ time where k is the number of points of P in $[a, b]$.

1-DIMENSIONAL RANGE SEARCHING



- Problem: Given a set P of n points $\{p_1, p_2, \dots, p_n\}$ on the real line, report points of P that lie in the range $[a, b]$, $a \leq b$.
- Using binary search on an array we can answer such a query in $O(\log n + k)$ time where k is the number of points of P in $[a, b]$.
- However, when we permit insertion or deletion of points, we cannot use an array answering queries so efficiently.

1-DIMENSIONAL RANGE SEARCHING

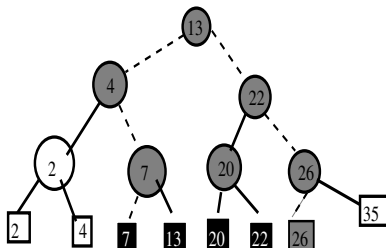


Search range [6,25]

Report 7,13,20,22

- We use a *binary leaf search tree* where leaf nodes store the points on the line, sorted by x-coordinates.

1-DIMENSIONAL RANGE SEARCHING

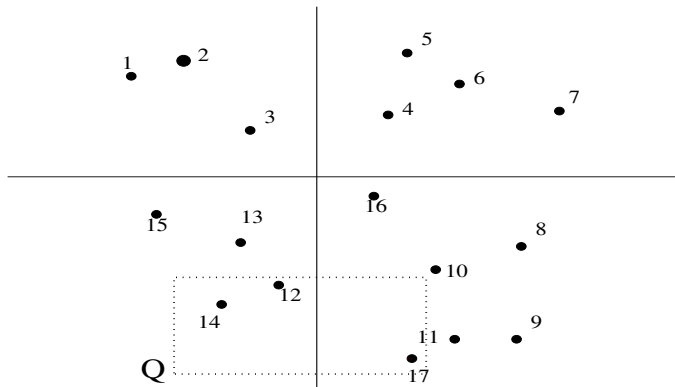


Search range [6,25]

Report 7,13,20,22

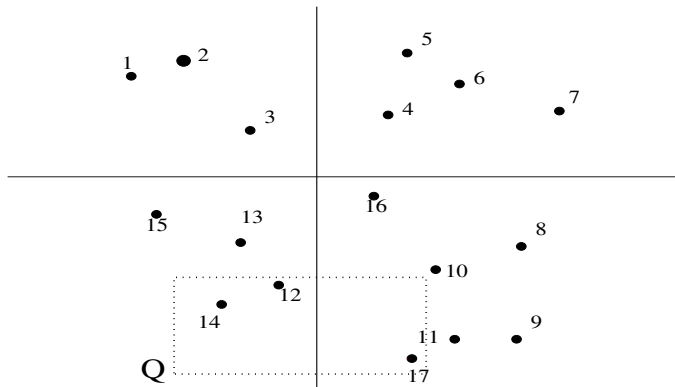
- We use a *binary leaf search tree* where leaf nodes store the points on the line, sorted by x-coordinates.
- Each internal node stores the x-coordinate of the rightmost point in its left subtree for guiding search.

2-DIMENSIONAL RANGE SEARCHING



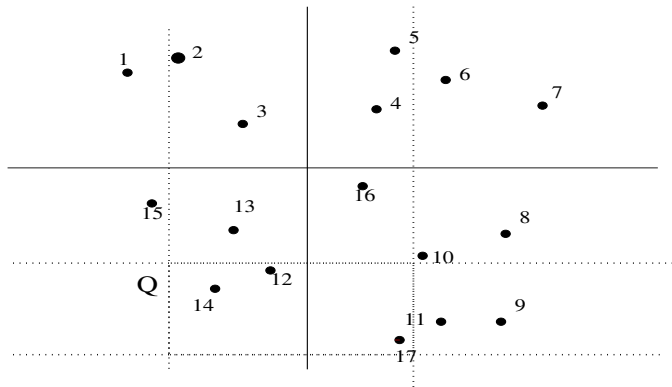
- Problem: Given a set P of n points in the plane, report points inside a query rectangle Q whose sides are parallel to the axes.

2-DIMENSIONAL RANGE SEARCHING



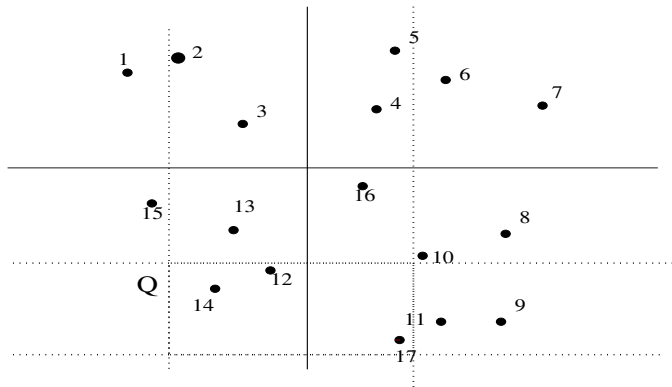
- Problem: Given a set P of n points in the plane, report points inside a query rectangle Q whose sides are parallel to the axes.
- Here, the points inside R are 14, 12 and 17.

2-DIMENSIONAL RANGE SEARCHING



- Using two 1-d range queries, one along each axis, solves the 2-d range query.

2-DIMENSIONAL RANGE SEARCHING



- Using two 1-d range queries, one along each axis, solves the 2-d range query.
- The cost incurred may exceed the actual output size of the 2-d range query.

RANGE SEARCHING WITH RANGE TREES AND KD-TREES

- Given a set S of n points in the plane, we can construct a $2d$ -range tree in $O(n \log n)$ time and space, so that rectangle queries can be executed in $O(\log^2 n + k)$ time.

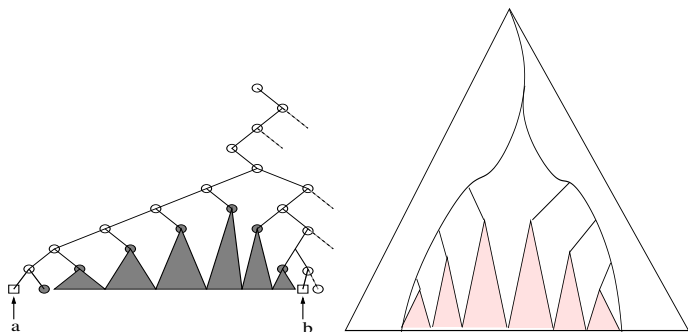
RANGE SEARCHING WITH RANGE TREES AND KD-TREES

- Given a set S of n points in the plane, we can construct a $2d$ -range tree in $O(n \log n)$ time and space, so that rectangle queries can be executed in $O(\log^2 n + k)$ time.
- The query time can be improved to $O(\log n + k)$ using the technique of *fractional cascading*.

RANGE SEARCHING WITH RANGE TREES AND KD-TREES

- Given a set S of n points in the plane, we can construct a $2d$ -range tree in $O(n \log n)$ time and space, so that rectangle queries can be executed in $O(\log^2 n + k)$ time.
- The query time can be improved to $O(\log n + k)$ using the technique of *fractional cascading*.
- Given a set S of n points in the plane, we can construct a Kd-tree in $O(n \log n)$ time and $O(n)$ space, so that *rectangle queries* can be executed in $O(\sqrt{n} + k)$ time. Here, the number of points in the query rectangle is k .

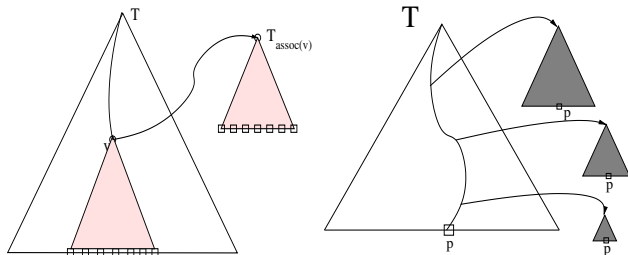
RANGE SEARCHING IN THE PLANE USING RANGE TREES



Given a 2-d rectangle query $[a, b] \times [c, d]$, we can identify subtrees whose leaf nodes are in the range $[a, b]$ along the X-direction.

Only a subset of these leaf nodes lie in the range $[c, d]$ along the

RANGE SEARCHING IN THE PLANE USING RANGE TREES

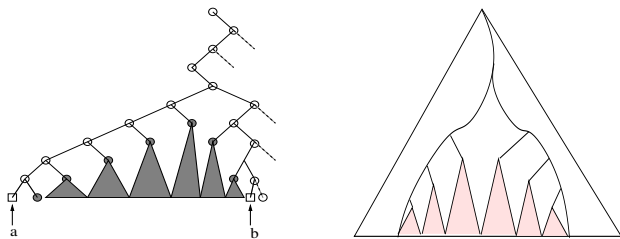


$T_{assoc(v)}$ is a binary search tree on y -coordinates for points in the leaf nodes of the subtree rooted at v in the tree T .

The point p is duplicated in $T_{assoc(v)}$ for each v on the search path for p in tree T .

The total space requirement is therefore $O(n \log n)$.

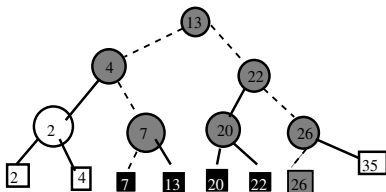
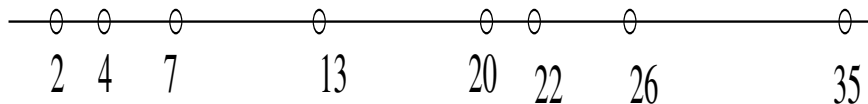
RANGE SEARCHING IN THE PLANE USING RANGE TREES



We perform 1-d range queries with the y-range $[c, d]$ in each of the subtrees adjacent to the left and right search paths within the x-range $[a, b]$ in the tree T .

Since the search path is $O(\log n)$ in size, and each y-range query requires $O(\log n)$ time, the total cost of searching is $O(\log^2 n)$. The reporting cost is $O(k)$ where k points lie in the query rectangle.

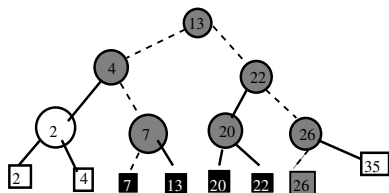
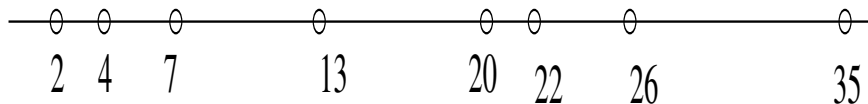
2-RANGE TREE SEARCHING



Search range [6,25]

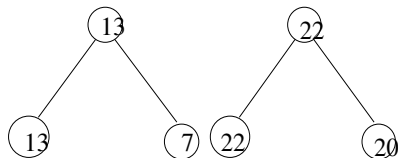
Report 7,13,20,22

2-RANGE TREE SEARCHING

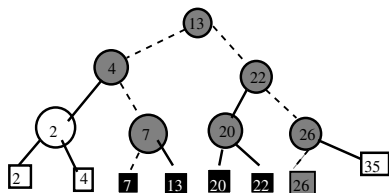
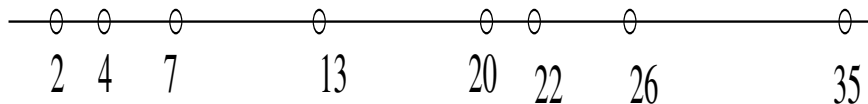


Search range [6,25]

Report 7,13,20,22

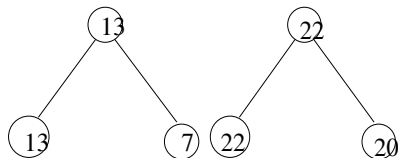


2-RANGE TREE SEARCHING

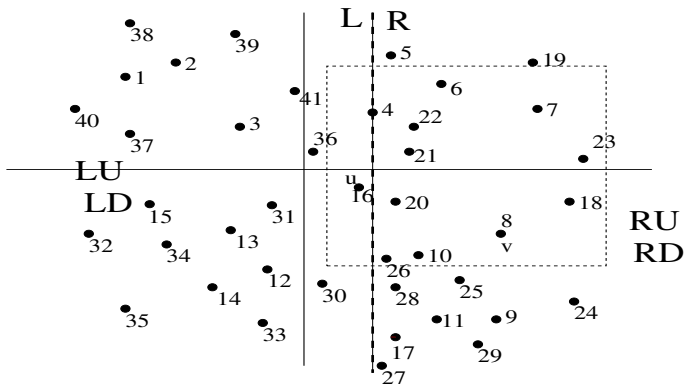


Search range [6,25]

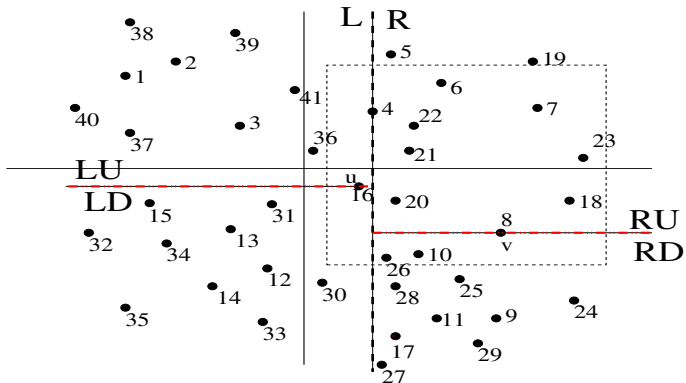
Report 7,13,20,22



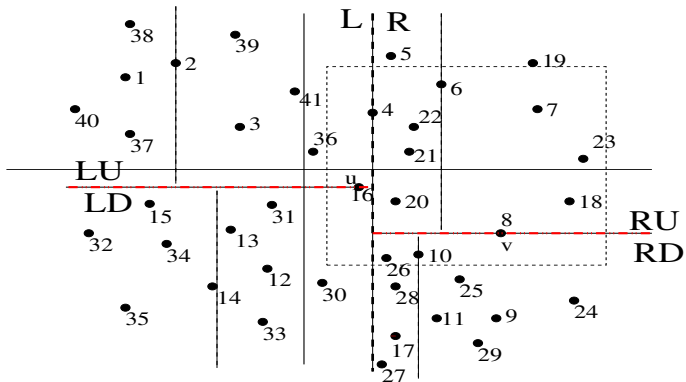
PARTITION BY THE MEDIAN OF X-COORDINATES



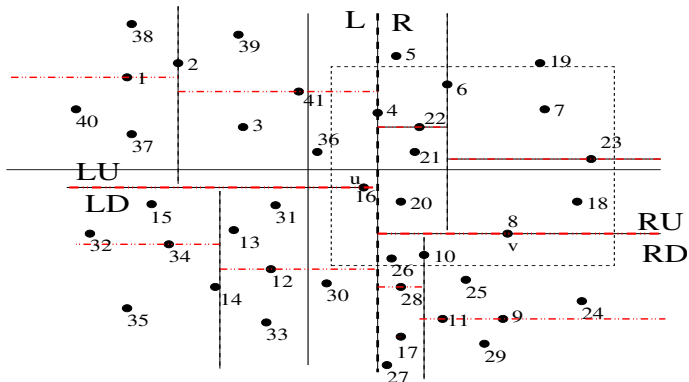
PARTITION BY THE MEDIAN OF Y-COORDINATES



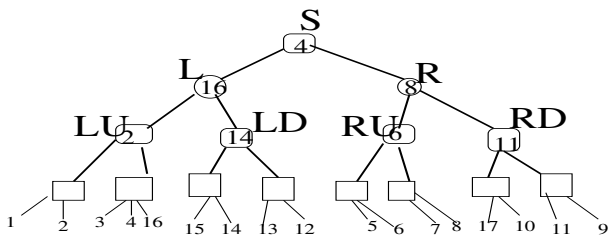
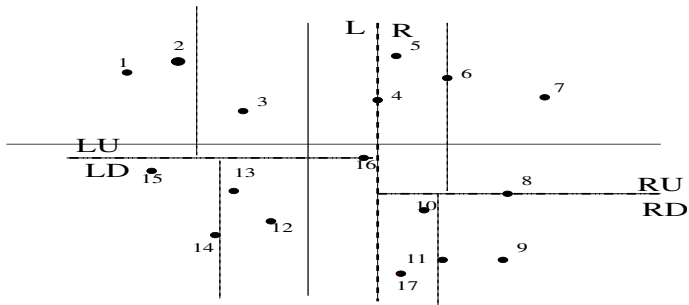
PARTITION BY THE MEDIAN OF X-COORDINATES



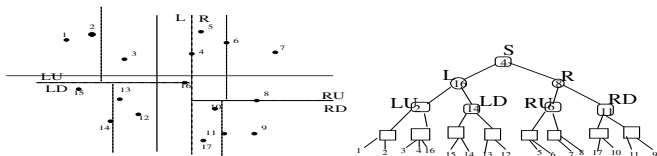
PARTITION BY THE MEDIAN OF Y-COORDINATES



2-DIMENSIONAL RANGE SEARCHING USING KD-TREES

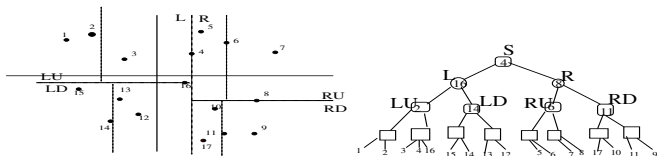


DESCRIPTION OF THE KD-TREE



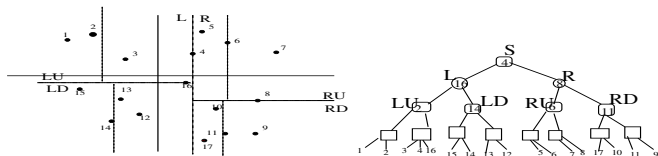
- The tree T is a perfectly height-balanced binary search tree with alternate layers of nodes spitting subsets of points in P using x - and y - coordinates, respectively as follows.

DESCRIPTION OF THE KD-TREE



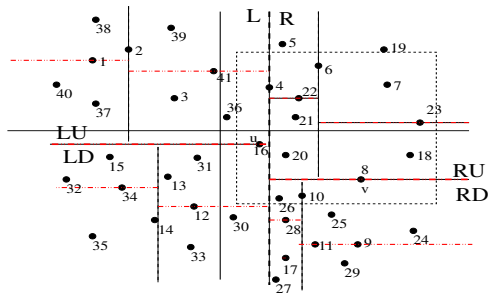
- The tree T is a perfectly height-balanced binary search tree with alternate layers of nodes spitting subsets of points in P using x - and y - coordinates, respectively as follows.
- The point r stored in the root vertex T splits the set S into two roughly equal sized sets L and R using the median x -coordinate $x_{median}(S)$ of points in S , so that all points in L (R) have abscissae less than or equal to (strictly greater than) $x_{median}(S)$.

DESCRIPTION OF THE KD-TREE



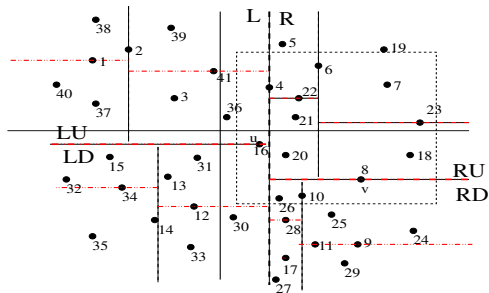
- The tree T is a perfectly height-balanced binary search tree with alternate layers of nodes spitting subsets of points in P using x - and y - coordinates, respectively as follows.
- The point r stored in the root vertex T splits the set S into two roughly equal sized sets L and R using the median x -coordinate $x_{median}(S)$ of points in S , so that all points in L (R) have abscissae less than or equal to (strictly greater than) $x_{median}(S)$.
- The entire plane is called the *region*(r).

ANSWERING RECTANGLE QUERIES



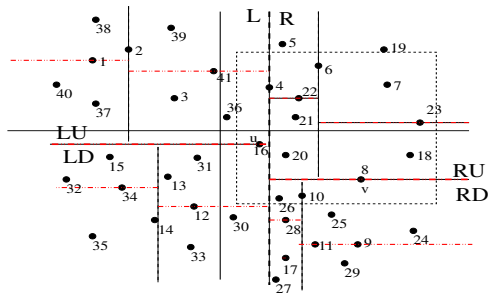
- A query rectangle Q may (i) overlap a region, (ii) completely contain a region, or (iii) completely miss a region.

ANSWERING RECTANGLE QUERIES



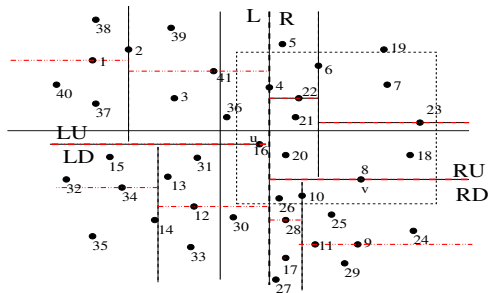
- A query rectangle Q may (i) overlap a region, (ii) completely contain a region, or (iii) completely miss a region.
- If R contains the entire bounded $region(p)$ of a point p defining a node N of T then report all points in $region(p)$.

ANSWERING RECTANGLE QUERIES



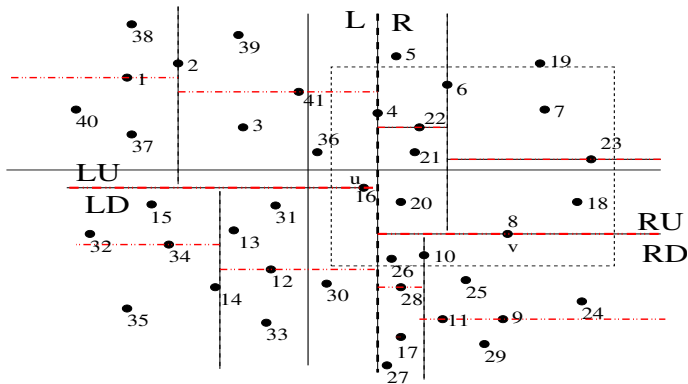
- A query rectangle Q may (i) overlap a region, (ii) completely contain a region, or (iii) completely miss a region.
- If R contains the entire bounded $region(p)$ of a point p defining a node N of T then report all points in $region(p)$.
- If R misses the $region(p)$ then we do not traverse the subtree rooted at this node.

ANSWERING RECTANGLE QUERIES



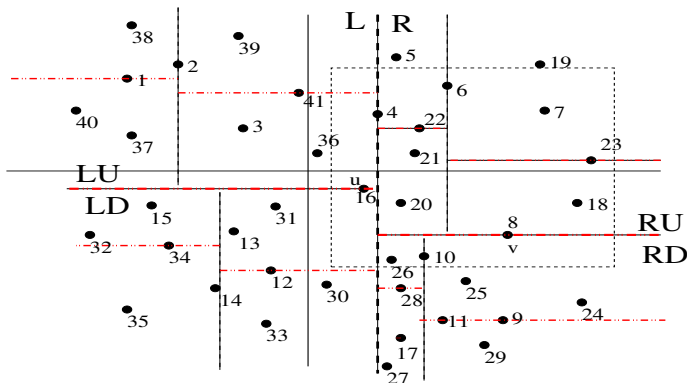
- A query rectangle Q may (i) overlap a region, (ii) completely contain a region, or (iii) completely miss a region.
- If R contains the entire bounded $region(p)$ of a point p defining a node N of T then report all points in $region(p)$.
- If R misses the $region(p)$ then we do not traverse the subtree rooted at this node.
- If R overlaps $region(p)$ then we check whether R also overlaps the

2-DIMENSIONAL RANGE SEARCHING: KD-TREES



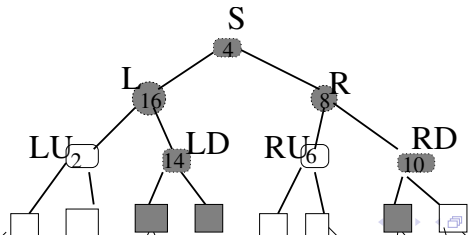
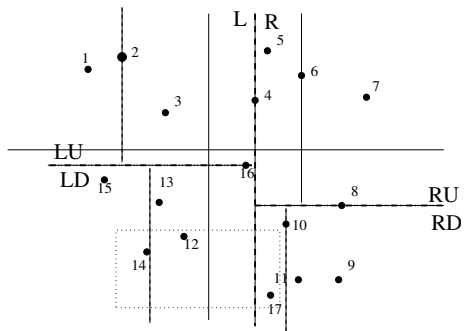
- The set L (R) is split into two roughly equal sized subsets LU and LD (RU and RD), using point u (v) that has the median y -coordinate in the set L (R), and including u in LU (RU).

2-DIMENSIONAL RANGE SEARCHING: KD-TREES

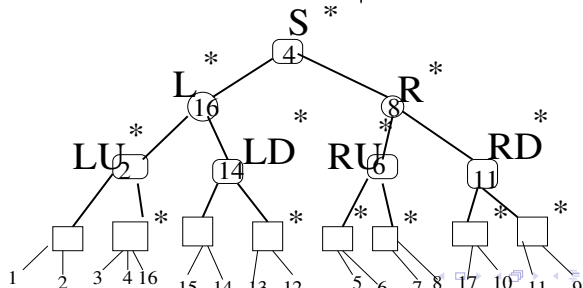
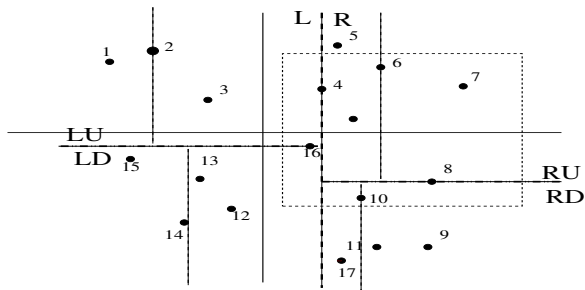


- The set L (R) is split into two roughly equal sized subsets LU and LD (RU and RD), using point u (v) that has the median y -coordinate in the set L (R), and including u in LU (RU).
- The entire halfplane containing set L (R) is called the $region(u)$ ($region(v)$).

NODES TRAVERSED IN THE KD-TREE



NODES TRAVERSED IN THE KD-TREE



TIME COMPLEXITY OF OUTPUT POINT REPORTING

- Reporting points within R contributes to the output size k for the query.

TIME COMPLEXITY OF OUTPUT POINT REPORTING

- Reporting points within R contributes to the output size k for the query.
- No leaf level region in T has more than 2 points.

TIME COMPLEXITY OF OUTPUT POINT REPORTING

- Reporting points within R contributes to the output size k for the query.
- No leaf level region in T has more than 2 points.
- So, the cost of inspecting points outside R but within the intersection of leaf level regions of T can be charged to the internal nodes traversed in T .

TIME COMPLEXITY OF OUTPUT POINT REPORTING

- Reporting points within R contributes to the output size k for the query.
- No leaf level region in T has more than 2 points.
- So, the cost of inspecting points outside R but within the intersection of leaf level regions of T can be charged to the internal nodes traversed in T .
- This cost is borne for all leaf level regions intersected by R .

WORST-CASE COST OF TRAVERSAL

- It is sufficient to determine the upper bound on the number of (internal) nodes whose regions are intersected by a single vertical (horizontal) line.

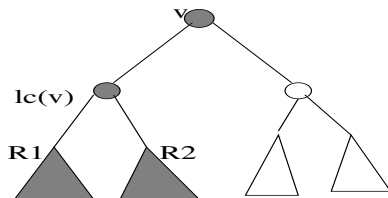
WORST-CASE COST OF TRAVERSAL

- It is sufficient to determine the upper bound on the number of (internal) nodes whose regions are intersected by a single vertical (horizontal) line.
- Any vertical line intersecting S can intersect either L or R but not both, but it can meet both RU and RD (LU and LD).

WORST-CASE COST OF TRAVERSAL

- It is sufficient to determine the upper bound on the number of (internal) nodes whose regions are intersected by a single vertical (horizontal) line.
- Any vertical line intersecting S can intersect either L or R but not both, but it can meet both RU and RD (LU and LD).
- Any horizontal line intersecting R can intersect either RU or RD but not both, but it can meet both children of RU (RD).

TIME COMPLEXITY OF RECTANGLE QUERIES FOR KD-TREES

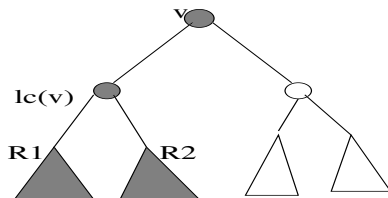


- Therefore, the time complexity $T(n)$ for an n -vertex Kd-tree obeys the recurrence relation

$$T(n) = 2 + 2T\left(\frac{n}{4}\right)$$

$$T(1) = 1$$

TIME COMPLEXITY OF RECTANGLE QUERIES FOR KD-TREES



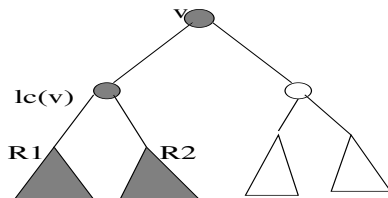
- Therefore, the time complexity $T(n)$ for an n -vertex Kd-tree obeys the recurrence relation

$$T(n) = 2 + 2T\left(\frac{n}{4}\right)$$

$$T(1) = 1$$

- The solution for $T(n) = O(\sqrt{(n)})$.

TIME COMPLEXITY OF RECTANGLE QUERIES FOR KD-TREES



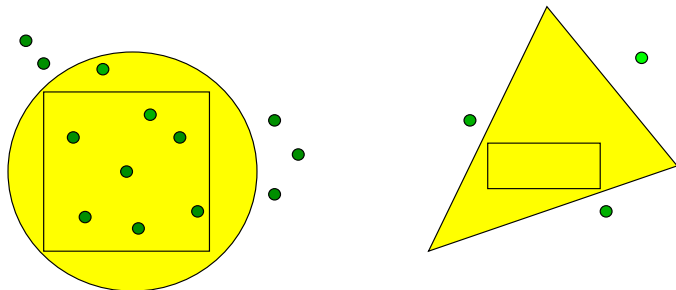
- Therefore, the time complexity $T(n)$ for an n -vertex Kd-tree obeys the recurrence relation

$$T(n) = 2 + 2T\left(\frac{n}{4}\right)$$

$$T(1) = 1$$

- The solution for $T(n) = O(\sqrt{(n)})$.
- The total cost of reporting k points in R is therefore $O(\sqrt{(n)} + k)$.

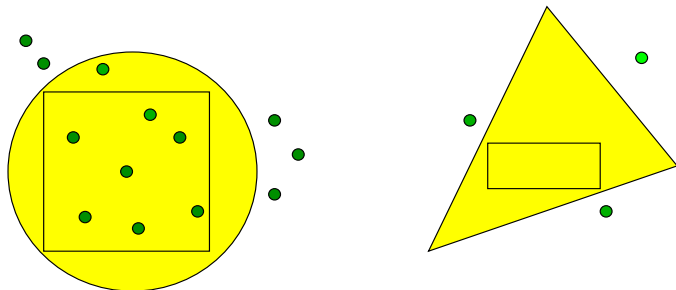
MORE GENERAL QUERIES



General Queries:

- Triangles can be used to simulate polygonal shapes with straight edges.

MORE GENERAL QUERIES



General Queries:

- Triangles can be used to simulate polygonal shapes with straight edges.
- Circles cannot be simulated by triangles either.

TRIANGLE QUERIES

- Using $O(n^2)$ space and time for preprocessing, triangle queries can be reported in $O(\log^2 n + k)$ time, where k is the number of points inside the query triangle.

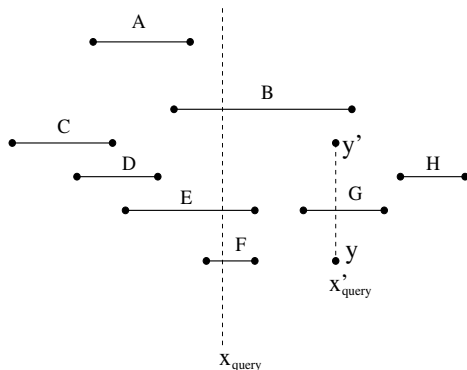
Goswami, Das and Nandy: Comput. Geom. Th. and Appl. 29 (2004) pp. 163-175.

TRIANGLE QUERIES

- Using $O(n^2)$ space and time for preprocessing, triangle queries can be reported in $O(\log^2 n + k)$ time, where k is the number of points inside the query triangle.
- For counting the number k of points inside a query triangle, worst-case optimal $O(\log n)$ time suffices.

Goswami, Das and Nandy: Comput. Geom. Th. and Appl. 29 (2004) pp. 163-175.

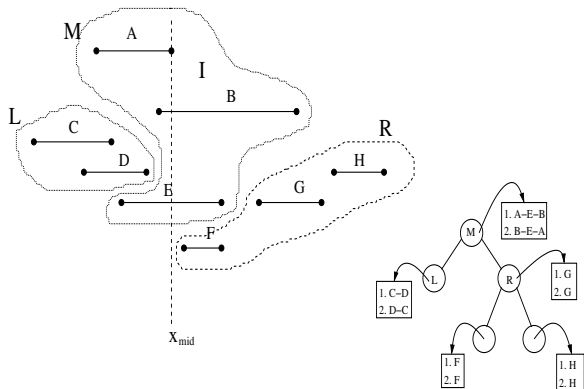
FINDING INTERVALS CONTAINING A VERTICAL QUERY LINE/SEGMENT



Simpler queries ask for reporting all intervals intersecting the vertical line $X = x_{query}$.

More difficult queries ask for reporting all intervals intersecting a vertical segment joining (x', y') and (x', y)

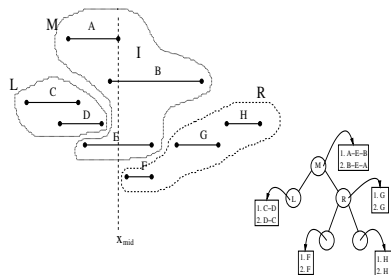
CONSTRUCTING THE INTERVAL TREE



The set M has intervals intersecting the vertical line $X = x_{mid}$, where x_{mid} is the median of the x -coordinates of the $2n$ endpoints.

The root node has intervals M sorted in two independent orders (i) by right end points ($B-E-A$), and (ii) left end points ($A-E-B$).

ANSWERING QUERIES USING AN INTERVAL TREE



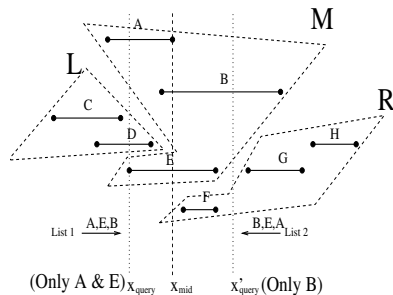
The set L and R have at most n endpoints each.

So they have at most $\frac{n}{2}$ intervals each.

Clearly, the cost of (recursively) building the interval tree is $O(n \log n)$.

The space required is linear.

ANSWERING QUERIES USING AN INTERVAL TREE

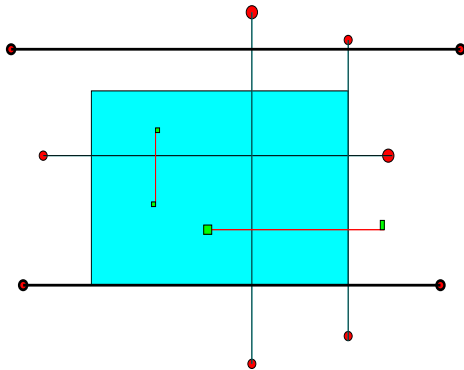


For $x_{query} < x_{mid}$, we do not traverse subtree for subset R .

For $x'_{query} > x_{mid}$, we do not traverse subtree for subset L .

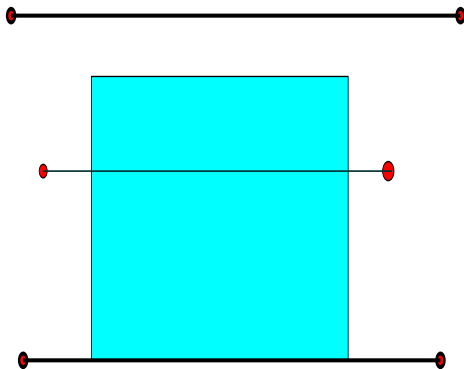
Clearly, the cost of reporting the k intervals is $O(\log n + k)$.

REPORTING (PORTIONS OF) ALL RECTILINEAR SEGMENTS INSIDE A QUERY RECTANGLE



For detecting segments with one (or both) ends inside the rectangle, it is sufficient to maintain rectangular range query apparatus for output-sensitive query processing.

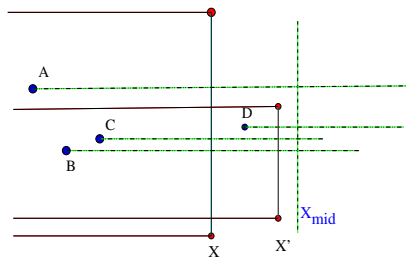
REPORTING SEGMENTS WITH NO ENDPOINTS INSIDE THE QUERY RECTANGLE



Report all (horizontal) segments that cut across the query rectangle or include an entire (top/bottom) bounding edge.

Use either the right (or left) edge, and the top (or bottom) edge of the query rectangle.

RIGHT EDGES X AND X' OF TWO QUERY RECTANGLES

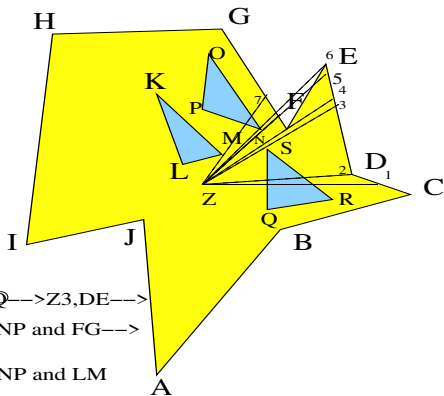


Use an interval tree of all the horizontal segments and the right bounding edge of the query rectangle like X or X' .

This helps reporting all segments cutting the right edge of the query rectangle. Use the rectangle query for vertical segment X and find points A , B and C in the rectangle with left edge at minus infinity. For X' , report B , C and D , similarly.

COMPUTING THE VISIBLE REGION IN A POLYGON WITH OPAQUE OBSTACLES

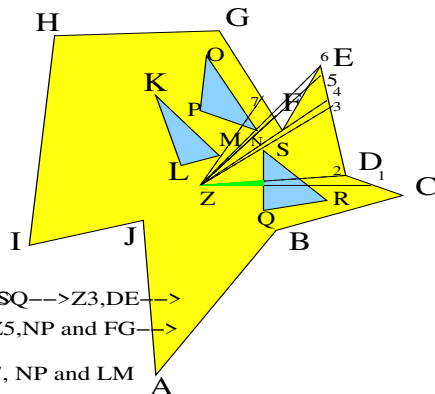
SQ,SR,DC,1-->SQ,SR,DE,2-->DE,3--
 FG,FE,DE,4-->NP,NO,FG,FE,DE,5-->
 NP,NO,FG,FE,DE,6-->LM,MK,NP,NO,FG,7



Z1 ,SQ-->Z2,SQ-->Z3,DE-->
 Z4,FG and DE-->Z5,NP and FG-->
 Z6,NP-->Z7, NP and LM

COMPUTING THE VISIBLE REGION IN A POLYGON WITH OPAQUE OBSTACLES

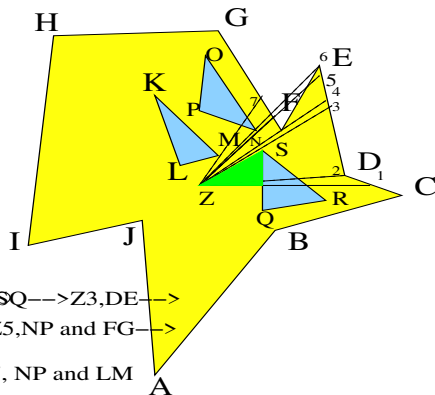
SQ,SR,DC,1-->SQ,SR,DE,2-->DE,3
 FG,FE,DE,4-->NP,NO,FG,FE,DE,5-->
 NP,NO,FG,FE,DE,6-->LM,MK,NP,NO,FG,7



Z1 ,SQ-->Z2,SQ-->Z3,DE-->
 Z4,FG and DE-->Z5,NP and FG-->
 Z6,NP-->Z7, NP and LM

COMPUTING THE VISIBLE REGION IN A POLYGON WITH OPAQUE OBSTACLES

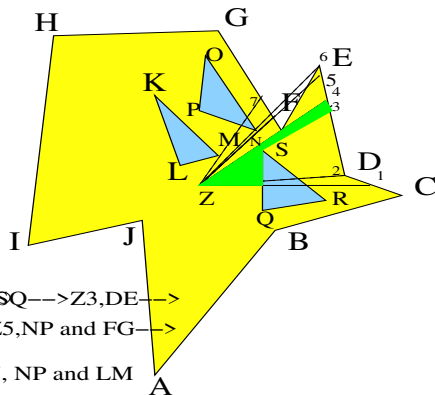
SQ,SR,DC,1-->SQ,SR,DE,2-->DE,3
 FG,FE,DE,4-->NP,NO,FG,FE,DE,5-->
 NP,NO,FG,FE,DE,6-->LM,MK,NP,NO,FG,7



Z1 ,SQ-->Z2,SQ-->Z3,DE-->
 Z4,FG and DE-->Z5,NP and FG-->
 Z6,NP-->Z7, NP and LM

COMPUTING THE VISIBLE REGION IN A POLYGON WITH OPAQUE OBSTACLES

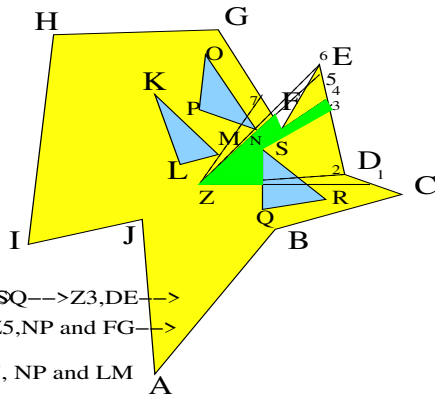
SQ,SR,DC,1-->SQ,SR,DE,2-->DE,3
 FG,FE,DE,4-->NP,NO,FG,FE,DE,5-->
 NP,NO,FG,FE,DE,6-->LM,MK,NP,NO,FG,7



Z1 ,SQ-->Z2,SQ-->Z3,DE-->
 Z4,FG and DE-->Z5,NP and FG-->
 Z6,NP-->Z7, NP and LM

COMPUTING THE VISIBLE REGION IN A POLYGON WITH OPAQUE OBSTACLES

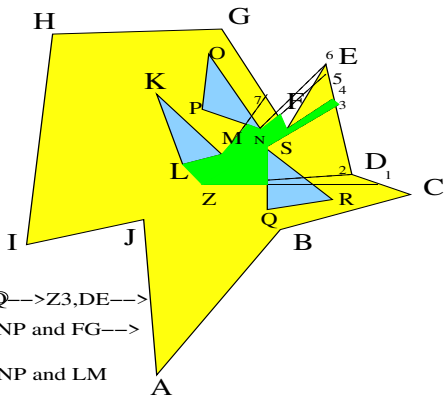
SQ,SR,DC,1-->SQ,SR,DE,2-->DE,3
 FG,FE,DE,4-->NP,NO,FG,FE,DE,5-->
 NP,NO,FG,FE,DE,6-->LM,MK,NP,NO,FG,7



Z1 ,SQ-->Z2,SQ-->Z3,DE-->
 Z4,FG and DE-->Z5,NP and FG-->
 Z6,NP-->Z7, NP and LM

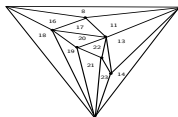
COMPUTING THE VISIBLE REGION IN A POLYGON WITH OPAQUE OBSTACLES

SQ,SR,DC,1-->SQ,SR,DE,2-->DE,3--
 FG,FE,DE,4-->NP,NO,FG,FE,DE,5-->
 NP,NO,FG,FE,DE,6-->LM,MK,NP,NO,FG,7

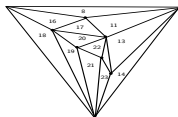


Z1 ,SQ-->Z2,SQ-->Z3,DE-->
 Z4,FG and DE-->Z5,NP and FG-->
 Z6,NP-->Z7, NP and LM

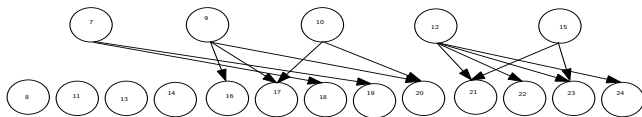
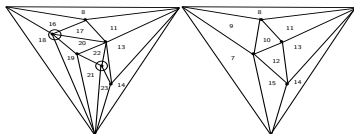
PLANAR POINT LOCATION BY TRIANGULATION REFINEMENT



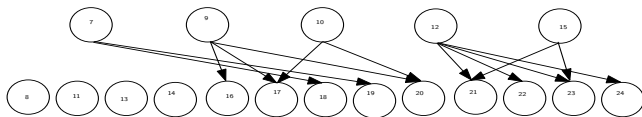
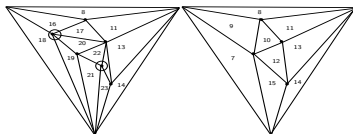
PLANAR POINT LOCATION BY TRIANGULATION REFINEMENT

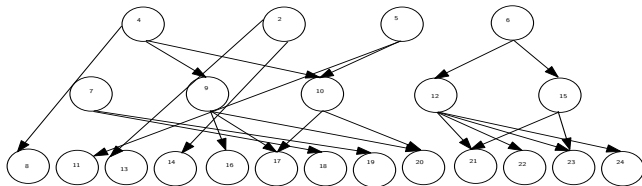
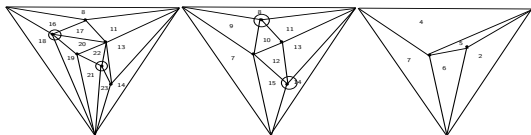


PLANAR POINT LOCATION BY TRIANGULATION REFINEMENT

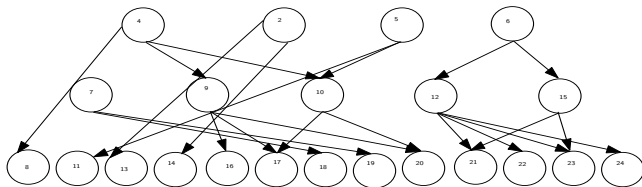
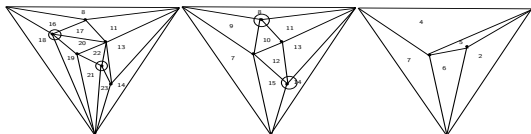


PLANAR POINT LOCATION BY TRIANGULATION REFINEMENT

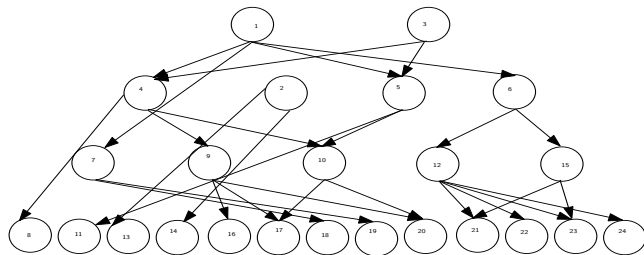
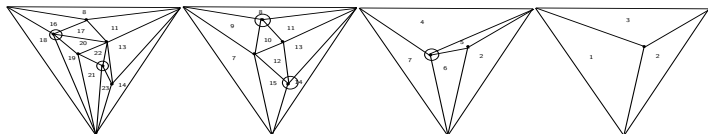




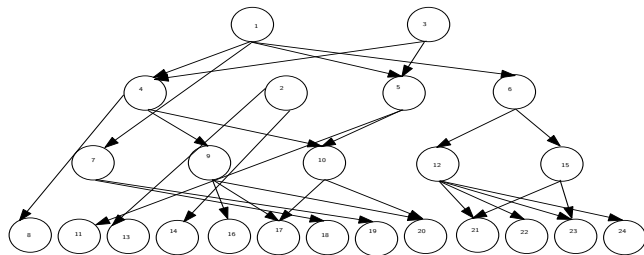
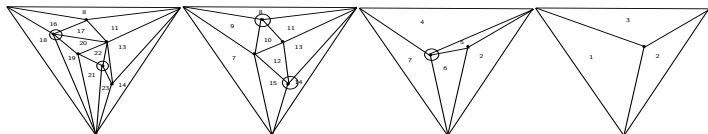
PLANAR POINT LOCATION BY TRIANGULATION REFINEMENT



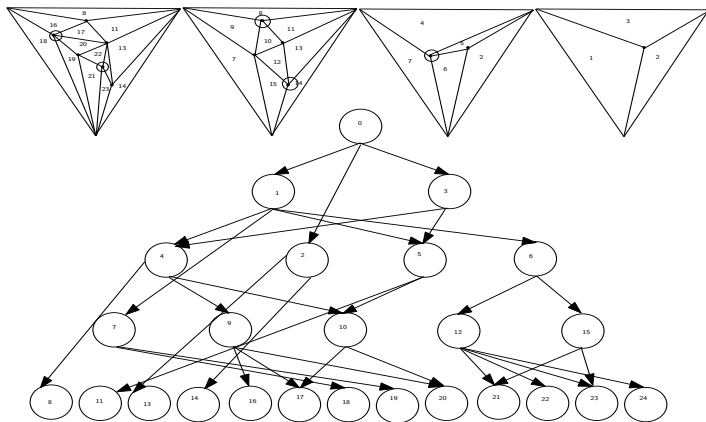
PLANAR POINT LOCATION BY TRIANGULATION REFINEMENT



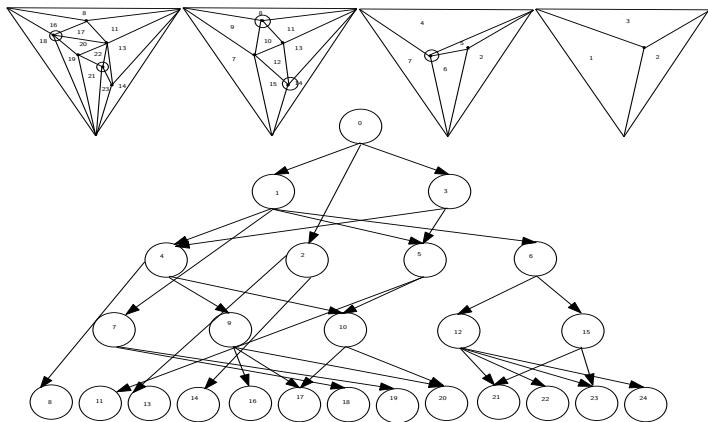
PLANAR POINT LOCATION BY TRIANGULATION REFINEMENT














PLANAR POINT LOCATION BY TRIANGULATION REFINEMENT



PLANAR POINT LOCATION BY TRIANGULATION REFINEMENT



-  Jiri Matousek, *Lectures on Discrete Geometry*, Springer.
-  David Mount's lecture notes— CMSC 754 Computational Geometry, David M. Mount Department of Computer Science, University of Maryland, Fall 2016.
-  Ketan Mulmuley, *Computational Geometry: An Introduction Through Randomized Algorithms*, Prentice Hall, 1994.
-  Janos Pach and Pankaj Agarwal, *Combinatorial Geometry*, Wiley-Interscience Series in Discrete Mathematics and Optimization, 1995.
-  B. Chazelle, *The discrepancy method: Randomness and complexity*, Cambridge University Press, 2000.
-  T. H. Cormen, C. E. Leiserson, R. L. Rivest, *Introduction to algorithms*, Second Edition, Prentice-Hall India, 2003.
-  R. Motwani and P. Raghavan, *Randomized algorithms*, Cambridge University Press, 1995.
-  F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, Springer, 1985.

-  Mark de Berg, Otfried Schwarzkopf, Marc van Kreveld and Mark Overmars, Computational Geometry: Algorithms and Applications, Springer.
-  S. K. Ghosh, Visibility Algorithms in the Plane, Cambridge University Press, Cambridge, UK, 2007.
-  Kurt Mehlhorn, Data Structures and Algorithms, Vol. 3, Springer.