Instance-Optimal Geometric Algorithms

Diego Seco

Universidade da Coruña

Rules

- Questions are allowed
- Answers are appreciated
- Please... ask and answer...
- and you will be rewarded!

Outline

Evolution of the Analysis of Algorithms

- Worst-case
  - Pessimistic
- Average-case
  - No information about the performance on a specific input
- Adaptive
  - Non-comparable algorithms
- Instance-Optimal

Evolution of the Analysis of Algorithms

- Worst-case
  - Pessimistic
- Average-case
  - No information about the performance on a specific input
- Adaptive
  - Non-comparable algorithms
- Instance-Optimal

Evolution of the Analysis of Algorithms

- Worst-case
  - Pessimistic
- Average-case
  - No information about the performance on a specific input
- Adaptive
  - Non-comparable algorithms
- Instance-Optimal

Evolution of the Analysis of Algorithms

- Worst-case
  - Pessimistic
- Average-case
  - No information about the performance on a specific input
- Adaptive
  - Non-comparable algorithms
- Instance-Optimal

Instance-Optimal

- An algorithm A is instance-optimal if its cost is at most a constant factor from the cost of any other algorithm A' running on the same input, for every input instance
- But... this is to stringent!!!
  - Many linear algorithms when the input is ordered
  - Instance-optimal in the order-oblivious

Instance-Optimal

- An algorithm A is instance-optimal if its cost is at most a constant factor from the cost of any other algorithm A' running on the same input, for every input instance
- But... this is to stringent!!!
  - Many linear algorithms when the input is ordered
  - Instance-optimal in the order-oblivious

Instance-Optimal

- An algorithm A is instance-optimal if its cost is at most a constant factor from the cost of any other algorithm A' running on the same input, for every input instance
- But... this is to stringent!!!
  - Many linear algorithms when the input is ordered
  - Instance-optimal in the order-oblivious

Instance-Optimal

- An algorithm A is instance-optimal if its cost is at most a constant factor from the cost of any other algorithm A' running on the same input, for every input instance
- But... this is to stringent!!!
  - Many linear algorithms when the input is ordered
  - Instance-optimal in the order-oblivious

Instance-Optimal

Definition

A *correct* algorithm refers to an algorithm that outputs a correct answer for every possible sequence of elements in a domain $\mathcal{D}$

Definition

For a set S of n elements in $\mathcal{D}$, $T_A(S)$ denotes the maximum running time of A on input σ over all $n!$ possible permutations σ of S

Definition

$OPT(S)$ denotes the minimum of $T_{A'}(S)$ over all correct algorithms $A' \in \mathcal{A}$ ($\mathcal{A}$ is a class of algorithms)

Instance-Optimal

Definition

A *correct* algorithm refers to an algorithm that outputs a correct answer for every possible sequence of elements in a domain $\mathcal{D}$

Definition

For a set S of n elements in $\mathcal{D}$, $T_A(S)$ denotes the maximum running time of A on input σ over all $n!$ possible permutations σ of S

Definition

$OPT(S)$ denotes the minimum of $T_{A'}(S)$ over all correct algorithms $A' \in \mathcal{A}$ ($\mathcal{A}$ is a class of algorithms)

Instance-Optimal

Definition

A *correct* algorithm refers to an algorithm that outputs a correct answer for every possible sequence of elements in a domain $\mathcal{D}$

Definition

For a set S of n elements in $\mathcal{D}$, $T_A(S)$ denotes the maximum running time of A on input σ over all $n!$ possible permutations σ of S

Definition

$OPT(S)$ denotes the minimum of $T_{A'}(S)$ over all correct algorithms $A' \in \mathcal{A}$ ($\mathcal{A}$ is a class of algorithms)

Instance-Optimal

Definition

If $A \in \mathcal{A}$ is a correct algorithm such that $T_A(S) \leq O(1) \times OPT(S)$ for every set S, then we say A is instance-optimal in the order-oblivious setting

Goal

- Understand the advantages of the Instance-Optimal analysis
- Examples: 2-D maxima and convex-hull (Techniques)
- Is it the final analysis? (Drawbacks)
- Future?

Problem Statement

Definition

For two points p and q, p dominates q if each coordinate of p is greater than that the corresponding coordinate of q

Definition

Given a set S of n points in $\mathbb{R}^d$, a point p is maximal if $p \in S$ and p is not dominated by any other point in S

Definition

The maxima problem is to report all maximal points from left to right

Problem Statement

Definition

For two points p and q, p dominates q if each coordinate of p is greater than that the corresponding coordinate of q

Definition

Given a set S of n points in $\mathbb{R}^d$, a point p is maximal if $p \in S$ and p is not dominated by any other point in S

Definition

The maxima problem is to report all maximal points from left to right

Problem Statement

Definition

For two points p and q, p dominates q if each coordinate of p is greater than that the corresponding coordinate of q

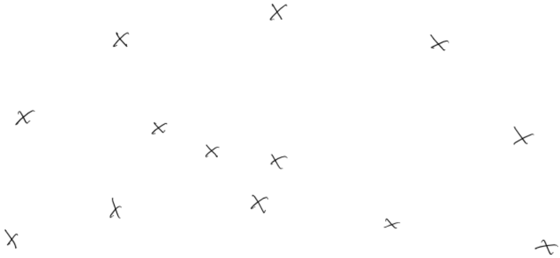
Definition

Given a set S of n points in $\mathbb{R}^d$, a point p is maximal if $p \in S$ and p is not dominated by any other point in S

Definition

The maxima problem is to report all maximal points from left to right

Example



Algorithm

maxima(Q):

1. if $|Q| = 1$ then return Q
2. divide Q into the left and right halves Q_ℓ and Q_r by the median x-coordinate
3. *discover* the point q with the maximum y-coordinate in Q_r (computable in linear time)
4. *prune* all points in Q_ℓ and Q_r dominated by q
5. return the concatenation of maxima(Q_ℓ) and maxima(Q_r)

Example

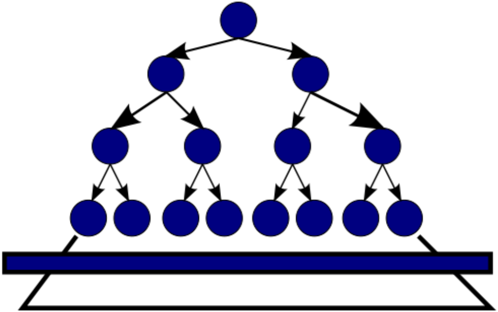
maxima(Q):

1. if $|Q| = 1$ then return Q
2. divide Q into the left and right halves Q_ℓ and Q_r by the median x-coordinate
3. *discover* the point q with the maximum y-coordinate in Q_r (computable in linear time)
4. *prune* all points in Q_ℓ and Q_r dominated by q
5. return the concatenation of maxima(Q_ℓ) and maxima(Q_r)

Upper bound

- Kirkpatrick and Seidel: $O(n \log h)$
- $O(n(\mathcal{H}(\Pi_{vert}) + 1))$
- Afshani, Barbay, and Chan: $O(n(\mathcal{H}(S) + 1))$

Execution tree [$O(n \log h)$]



Upper bound

Definition

Consider a partition Π of the input set S into disjoint subsets $S_1, S_2, \ldots, S_t$. Π is respectful if each subset S_k is either a singleton or can be enclosed by an axis-aligned box B_k whose interior is completely below the staircase of S

Definition

$H(\Pi) = \sum_{k=1}^{t}(|S_k|/n)\log(n/|S_k|)$

Definition

$H(S)$ is the minimum $H(\Pi)$

Upper bound

Definition

Consider a partition Π of the input set S into disjoint subsets $S_1, S_2, \ldots, S_t$. Π is respectful if each subset S_k is either a singleton or can be enclosed by an axis-aligned box B_k whose interior is completely below the staircase of S

Definition

$H(\Pi) = \sum_{k=1}^{t}(|S_k|/n)\log(n/|S_k|)$

Definition

$H(S)$ is the minimum $H(\Pi)$

Upper bound

Definition

Consider a partition Π of the input set S into disjoint subsets $S_1, S_2, \ldots, S_t$. Π is respectful if each subset S_k is either a singleton or can be enclosed by an axis-aligned box B_k whose interior is completely below the staircase of S

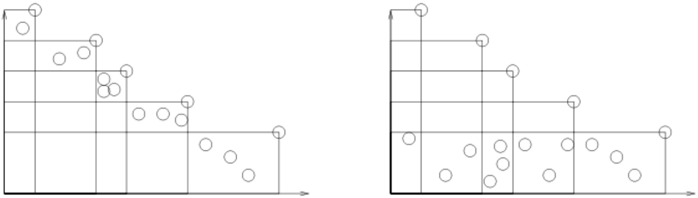
Definition

$H(\Pi) = \sum_{k=1}^{t} (|S_k|/n) \log(n/|S_k|)$

Definition

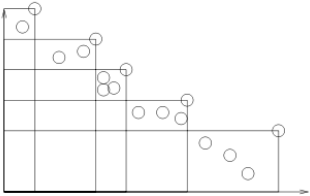
$H(S)$ is the minimum $H(\Pi)$

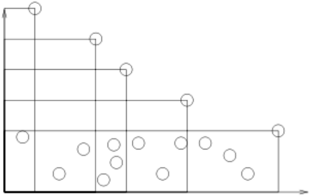
Example



$H(\Pi) = \sum_{k=1}^{t}(|S_k|/n)\log(n/|S_k|)$ is at maximum $\log h \ldots$ when?
$|S_k| = n/h$

Example

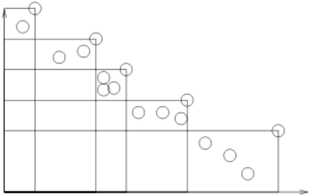


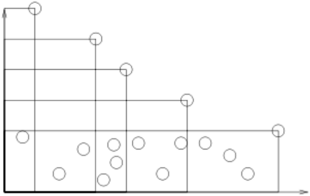


$H(\Pi) = \sum_{k=1}^{t} (|S_k|/n) \log(n/|S_k|)$ is at maximum $\log h \ldots$ when?

$|S_k| = n/h$

Example

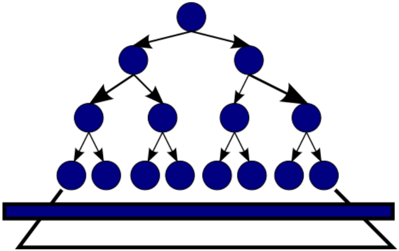




$H(\Pi) = \sum_{k=1}^{t} (|S_k|/n) \log(n/|S_k|)$ is at maximum $\log h \ldots$ when?
$|S_k| = n/h$

Execution tree $[O(n(\mathcal{H}(S)+1))]$

- X_j sublist of maximal points discovered during the first j levels
- $S^{(j)}$ subset of S that survives recursion level j and $n_j = |S^{(j)}|$
- There can be at most $\lceil n/2^j \rceil$ points of $S^{(j)}$ with x-coordinates between any two consecutive points in X_j
- All points of S that are strictly below the staircase of X_j have been pruned during levels $0, \ldots, j$ of the recursion



Lower bound

- $O(n \log n)$, $O(n \log h)$, $O(n(\mathcal{H}(\Pi_{vert}) + 1))$. All of them are optimum!!!

Theorem

$OPT(S) = \Omega(n(\mathcal{H}(S) + 1))$ for the 2-d maxima problem in the comparison model

- Use a K-d-tree ($\mathcal{T}$) to construct a partition
- Use an adversary to construct a *bad* permutation

Lower bound

- $O(n \log n)$, $O(n \log h)$, $O(n(\mathcal{H}(\Pi_{vert}) + 1))$. All of them are optimum!!!

Theorem

$OPT(S) = \Omega(n(\mathcal{H}(S) + 1))$ for the 2-d maxima problem in the comparison model

- Use a K-d-tree ($\mathcal{T}$) to construct a partition
- Use an adversary to construct a *bad* permutation

Lower bound

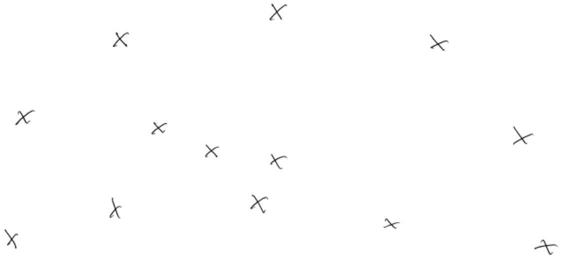
- $O(n \log n)$, $O(n \log h)$, $O(n(\mathcal{H}(\Pi_{vert}) + 1))$. All of them are optimum!!!

Theorem

$OPT(S) = \Omega(n(\mathcal{H}(S) + 1))$ for the 2-d maxima problem in the comparison model

- Use a K-d-tree $(\mathcal{T})$ to construct a partition
- Use an adversary to construct a *bad* permutation

K-d-tree



Adversary argument

- What is an adversary?
  - A second algorithm which intercepts access to data structures
  - Constructs the input data only as needed
  - Attempts to make original algorithm work as hard as possible
  - ... he is the Devil!!!
- How does he construct the permutation?
  - Maintain a box B_p in $\mathcal{T}$ (p fixed just in leaves)
  - For each box B in $\mathcal{T}$, $n(B)$ denotes the number of points p with B_p contained in B
  - Invariant: $n(B) \leq |S \bigcap B|$
  - If $n(B) = |S \bigcap B|$, B is full

Adversary argument

- What is an adversary?
  - A second algorithm which intercepts access to data structures
  - Constructs the input data only as needed
  - Attempts to make original algorithm work as hard as possible
  - ... he is the Devil!!!
- How does he construct the permutation?
  - Maintain a box B_p in $\mathcal{T}$ (p fixed just in leaves)
  - For each box B in $\mathcal{T}$, $n(B)$ denotes the number of points p with B_p contained in B
  - Invariant: $n(B) \leq |S \bigcap B|$
  - If $n(B) = |S \bigcap B|$, B is full

Adversary argument

- What is an adversary?
  - A second algorithm which intercepts access to data structures
  - Constructs the input data only as needed
  - Attempts to make original algorithm work as hard as possible
  - ... he is the Devil!!!
- How does he construct the permutation?
  - Maintain a box B_p in $\mathcal{T}$ (p fixed just in leaves)
  - For each box B in $\mathcal{T}$, $n(B)$ denotes the number of points p with B_p contained in B
  - Invariant: $n(B) \leq |S \bigcap B|$
  - If $n(B) = |S \bigcap B|$, B is full

Adversary argument

- Solving the comparisons (x-coordinates between two points p and q):
  1. If B_p (resp. B_q) at even depth
     - Reset B_p to one of its children (resp. B_q)
     - Both at odd depths
     - Median x-coordinate of B_p less than median x-coordinate of B_q
     - Reset B_p to the left child (B'_p) and B_q to the right child (B'_q)
     - Comparison solved
  2. If a child B'_p of B_p is full
     - Reset B_p to the sibling B''_p of B'_p
     - Go back to step 1 to solve the comparison
  3. When B_p (sim. B_q) is a leaf we use the actual x-coordinate

Adversary argument

- Solving the comparisons (x-coordinates between two points p and q):
  1. If B_p (resp. B_q) at even depth
     - Reset B_p to one of its children (resp. B_q)
     - Both at odd depths
     - Median x-coordinate of B_p less than median x-coordinate of B_q
     - Reset B_p to the left child (B'_p) and B_q to the right child (B'_q)
     - Comparison solved
  2. If a child B'_p of B_p is full
     - Reset B_p to the sibling B''_p of B'_p
     - Go back to step 1 to solve the comparison
  3. When B_p (sim. B_q) is a leaf we use the actual x-coordinate

Adversary argument

- Solving the comparisons (x-coordinates between two points p and q):
  1. If B_p (resp. B_q) at even depth
     - Reset B_p to one of its children (resp. B_q)
     - Both at odd depths
     - Median x-coordinate of B_p less than median x-coordinate of B_q
     - Reset B_p to the left child (B'_p) and B_q to the right child (B'_q)
     - Comparison solved
  2. If a child B'_p of B_p is full
     - Reset B_p to the sibling B''_p of B'_p
     - Go back to step 1 to solve the comparison
  3. When B_p (sim. B_q) is a leaf we use the actual x-coordinate

Adversary argument

- At the end of the simulation every B_p is already a leaf (if the algorithm is correct)... why?
  - Because in other case the evil adversary can modify the input and obtain a partition consistent with the comparison made and a different set of maximal points
  - Note that B_p contains at least two points and is not completely underneath the staircase of S
- The sum of the depth of B_p, D, provides a lower bound for the number of comparisons T that the algorithm makes
  - Each comparison $O(1)$ ordinary increments (step 1)
  - The total number of exceptional increments (step 2) is asymptotically at most the total number of ordinary increments ($O(T)$). (Amortization argument)
  - $D = O(T)$ (i.e. $T = \Omega(D)$)

Adversary argument

- At the end of the simulation every B_p is already a leaf (if the algorithm is correct)...why?
  - Because in other case the evil adversary can modify the input and obtain a partition consistent with the comparison made and a different set of maximal points
  - Note that B_p contains at least two points and is not completely underneath the staircase of S
- The sum of the depth of B_p, D, provides a lower bound for the number of comparisons T that the algorithm makes
  - Each comparison $O(1)$ ordinary increments (step 1)
  - The total number of exceptional increments (step 2) is asymptotically at most the total number of ordinary increments ($O(T)$). (Amortization argument)
  - $D = O(T)$ (i.e. $T = \Omega(D)$)

Adversary argument

- At the end of the simulation every B_p is already a leaf (if the algorithm is correct)... why?
  - Because in other case the evil adversary can modify the input and obtain a partition consistent with the comparison made and a different set of maximal points
  - Note that B_p contains at least two points and is not completely underneath the staircase of S
- The sum of the depth of B_p, D, provides a lower bound for the number of comparisons T that the algorithm makes
  - Each comparison $O(1)$ ordinary increments (step 1)
  - The total number of exceptional increments (step 2) is asymptotically at most the total number of ordinary increments ($O(T)$). (Amortization argument)
  - $D = O(T)$ (i.e. $T = \Omega(D)$)

Adversary argument

- At the end of the simulation each B_p has depth $\Theta(\log(n/|S \bigcap B_p|))$
- $T = \Omega(D) = \Omega(\sum_{leafB} |S \bigcap B| \log(n/|S \bigcap B_p|)) = \Omega(n\mathcal{H}(\Pi_{kd-tree})) = \Omega(n\mathcal{H}(S))$

Adversary argument

- At the end of the simulation each B_p has depth $\Theta(\log(n/|S \bigcap B_p|))$
- $T = \Omega(D) = \Omega(\sum_{leafB} |S \bigcap B| \log(n/|S \bigcap B_p|)) = \Omega(n\mathcal{H}(\Pi_{kd-tree})) = \Omega(n\mathcal{H}(S))$

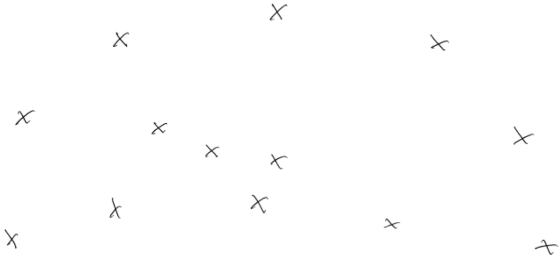
Problem Statement

Definition

Given a set S of n points in $\mathbb{R}^d$, the convex hull is the minimal convex set containing S

- It can be computed by running the *upper hull* algorithm on S and its reflection

Example



Algorithm

hull(Q):

1. if $|Q| = 2$ then return Q
2. *prune* all points from Q strictly below the line through the leftmost and rightmost point of Q
3. divide Q into the left and right halves Q_ℓ and Q_r by the median x-coordinate p_m
4. *discover* points q, q' that define the upper-hull edge $\overline{qq'}$ intersecting the vertical line at p_m
5. *prune* all points from Q_ℓ and Q_r that are strictly underneath the line segment $\overline{qq'}$
6. return the concatenation of hull(Q_ℓ) and hull(Q_r)

Example

hull(Q):

1. if $|Q| = 2$ then return Q
2. *prune* all points from Q strictly below the line through the leftmost and rightmost point of Q
3. divide Q into the left and right halves Q_ℓ and Q_r by the median x-coordinate p_m
4. *discover* points q, q' that define the upper-hull edge $\overline{qq'}$ intersecting the vertical line at p_m
5. *prune* all points from Q_ℓ and Q_r that are strictly underneath the line segment $\overline{qq'}$
6. return the concatenation of hull(Q_ℓ) and hull(Q_r)

Upper bound

- A partition Π is respectful if each subset S_k in Π is either a singleton or can be enclosed by a simples $\triangle_k$ whose interior is completely below the upper hull of S

Theorem

This 2-d upper hull algorithm runs in $O(n(\mathcal{H}(S)+1))$

- Substitute B_k with $\triangle_k$ in the proof

Upper bound

- A partition Π is respectful if each subset S_k in Π is either a singleton or can be enclosed by a simples $\triangle_k$ whose interior is completely below the upper hull of S

Theorem

This 2-d upper hull algorithm runs in $O(n(\mathcal{H}(S)+1))$

- Substitute B_k with $\triangle_k$ in the proof

Upper bound

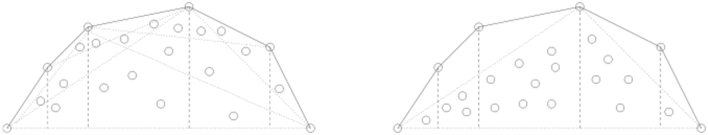
- A partition Π is respectful if each subset S_k in Π is either a singleton or can be enclosed by a simples $\triangle_k$ whose interior is completely below the upper hull of S

Theorem

This 2-d upper hull algorithm runs in $O(n(\mathcal{H}(S)+1))$

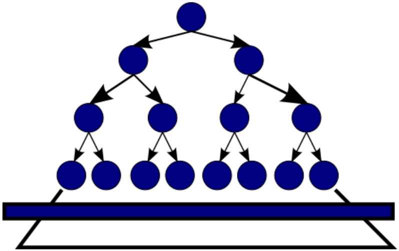
- Substitute B_k with $\triangle_k$ in the proof

Example



Execution tree $[O(n(\mathcal{H}(S)+1))]$

- X_j sublist of maximal points discovered during the first j levels
- $S^{(j)}$ subset of S that survives recursion level j and $n_j = |S^{(j)}|$
- There can be at most $\lceil n/2^j \rceil$ points of $S^{(j)}$ with x-coordinates between any two consecutive points in X_j
- All points of S that are strictly below the upper hull of X_j have been pruned during levels $0, \ldots, j$ of the recursion



Lower bound

Theorem

$OPT(S) = \Omega(n(\mathcal{H}(S)+1))$ for the upper hull problem in the multilinear decision tree model

- The proof is based on partition trees

Lower bound

Theorem

$OPT(S) = \Omega(n(\mathcal{H}(S) + 1))$ for the upper hull problem in the multilinear decision tree model

- The proof is based on partition trees

Present Work

- Instance-Optimal in the random-order setting
  - $T_A(S) \leq O(1) \times OPT^{avg}(S)$ ($OPT^{avg}(S)$ is the min $T_{A'}^{avg}(S)$)
  - Competitive against *randomized* algorithms
  - Subsume $average - case$ algorithms
- Instance-optimal algorithm for 3-d convex hull
- Other instance-optimal algorithms: orthogonal line segment intersection in 2-d, off-line orthogonal range searching in 2-d, off-line point location in 2-d, etc.

Future Work

- Other instance-optimal algorithm
  - Reporting all intersections between a set of disjoint red line segments and a set of disjoint blue line segments in 2-d
  - Computing the L_2- or $L_\infty-$ closest pair between a set of red points and a set of blue points in 2-d
  - Computing the diameter or the width of a 2-d point set
- Instance-optimal order-conscious (order-aware) algorithms?

References

- P. Afshani, J. Barbay, T. Chan *Instance-Optimal Geometric Algorithms*. FOCS'09.
- D. G. Kirkpatrick, R. Seidel *The ultimate planar convex hull algorithm*. SIAM'86.
- D. G. Kirkpatrick, R. Seidel *Output-size sensitive algorithm for finding maximal vectors*. ACM Symp. on Computational Geometry (1985).