"ULTIMATE PLANAR C.H. ALGORITHM?"
KIRKPATRICK-SEIDEL

It's a divide & conquer algorithm

[Diagram of upper hull only]
"ULTIMATE PLANAR C.H. ALGORITHM?"
KIRKPATRICK-SEIDEL

It's a divide & conquer algorithm

divide-conquer-merge

Upper hull only
"ULTIMATE PLANAR C.H. ALGORITHM?"
KIRKPATRICK-SEIDEL

It's a divide & conquer algorithm

divide-conquer-merge
divide-merge-conquer!

Upper hull only
"ULTIMATE PLANAR C.H. ALGORITHM?"
KIRKPATRICK-SEIDEL

It's a divide & conquer algorithm

divide-merge-conquer

Upper hull only
Goal: Get $O(n \log h)$

- We can't sort anything
  - How do we split? Median $O(n)$

- We can also afford $O(n)$ to merge
  - I.e. to find a bridge
Finding a Bridge in Linear Time

Let the bridge have slope $k^*$

Suppose we guess slope $k$.

Sweep $k$

Guess $K < K^*$
\[ \Rightarrow \text{sweep stops on blue} \]

Guess $K > K^*$
\[ \Rightarrow \text{sweep stops on red} \]

\[ \{ \text{Guess } K = K^* \] \Rightarrow \text{confirm bridge} \]

$O(n)$ time to guess & verify
- Arbitrarily pair up points
- Arbitrarily pair up points
- Find median slope
- Guess $K=\text{median}$
Case 1: $k > k^*$

Half of the pairs have slope $k' > k$, so $k' > k^*$
Case 1

$K > K^*$

\[ K^* \]

Half of the pairs have slope $K' > K$, so $K' > K^*$

$K^*$ can't sweep below $b$

@ can't be on bridge (it could be on C.H.)
Case 2:

\[ K < K^* \]

Half of the pairs have slope \[ K' < K \], so
\[ K' < K^* \]

\[ a_x < b_x \]}

\[ K^* \] can't sweep below \( a \) and \( b \) can't be on bridge (it could be on C.H.)
THROW AWAY ONE POINT (a or b) FROM HALF THE PAIRS

Case 1:

\[ K > K^* \]

Half of the pairs have slope \( K' > K \), so \( K' > K^* \)

\[ \text{can't be on bridge (it could be on C.H.)} \]

Case 2:

\[ K < K^* \]

Half of the pairs have slope \( K' < K \), so \( K' < K^* \)

\[ \text{can't be on bridge (it could be on C.H.)} \]
If we guess wrong:

THROW AWAY
ONE POINT \((a \lor b)\)
FROM HALF THE PAIRS

Then arbitrarily pair remaining points & "guess" again

Time: \(c\cdot n\) for first wrong guess

\[c \cdot \frac{3n}{4}\] for second "" ""

\[c \cdot \frac{3}{4} \cdot \frac{3n}{4}\] for third.

etc

total: \(O(n)\)
"Prune & Search"

If you can throw out a constant fraction of your input whenever you fail, then you will still have a good algorithm.

\[ T(n) = F(n) + T(\frac{n}{c}) \quad [c > 1] \]

\[ O(\log n) \quad O(1) : \text{binary search} \quad [c = 2] \]

\[ O(n) \quad O(n) : \text{finding a bridge} \quad [c = \frac{4}{3}] \]

\[ O(n^k) \quad O(n^k) : \frac{n^k + \frac{n^k}{2^k} + \frac{n^k}{4^k} + \cdots + \frac{n^k}{2^{i_k}}}{2} \quad [c = 2] \]

\[ O(2^n) \quad O(2^n) : 2^n + 2^{n-1} + 2^{n-2} + \cdots + 2 \quad [c = 2] \]
Example of linear-time bridge finding

- Unknown bridge
- Upper hull only
- Median separator
- $X_{\text{min}}$
- $X_{\text{max}}$
Find median slope
Test slope:
- too steep
- only left side is extremal
Because slope is too steep: discard left endpoints of steeper pairs.
Subset: \( \leq \frac{3}{4} \cdot \text{original} \)
Discard left endpoints of steeper pairs.
New subset \( \leq \frac{3}{4} \cdot \frac{3}{4} \cdot \text{original} \)
New random pairs and median slope
Too steep yet again
Discard...
This time:
Too shallow
Discard right endpoints of shallower pairs.
Find median slope. Extreme finds one point on each side. ➔ DONE
We know how to find a bridge in linear time.

Might as well throw out potential non-C.H. pts inside ... it's "free"
We know how to find a bridge in linear time.

Might as well throw out potential non-C.H. pts inside ... it's "free".

Of course we might not throw anything out.
We know how to find a bridge in linear time.

Solve 2 smaller problems with $n$ half points each.

That still only gives us $O(n \log n)$. Do we have to find a bridge that "splits" the hull evenly?

If we at least find one new bridge on both sides then we get $O(\log h)$ depth.

If we don't find a bridge on one side, we must have thrown out $\frac{n}{2}$ pts.
Cost tree

Example

\[ \text{c.n} \rightarrow \text{first bridge} \]

\[ \text{c.n/2} \rightarrow 2 \text{ more bridges} \]

\[ \text{c.n/4} \rightarrow \text{"only" 3 bridges} \]

Tree must have exactly \( h \) nodes

\[ \text{c.n/4} \rightarrow \text{actually a good thing} \]
cost tree

\[ \begin{align*}
&c \cdot n \\
&c \cdot \frac{n}{2} \\
&c \cdot \frac{n}{4} \\
&c \cdot \frac{n}{4} \\
&\times \\
&c \cdot \frac{n}{4} \\
\end{align*} \]

first bridge

2 more bridges

"only" 3 bridges

exactly \( h \) nodes

balanced case:

\[ \begin{align*}
&c n \\
&c n \\
&c n \\
&c n \\
\end{align*} \]

\( \text{depth } O(\log h) \)

\( \text{work } O(n \log h) \)

\( h \) can be \( O(n) \)
cost tree

- first bridge
- 2 more bridges
- "only" 3 bridges
- exactly $h$ nodes

unbalanced case

$O(\log n)$ depth

$O(n)$ work

if you keep getting "unbalanced" hull edges, you will run out of points quickly (i.e., it can't keep happening!)

in this case $h$ cannot be $O(n)$! (unlike, say, quick-hull)
Cost tree

- First bridge: $C \cdot \frac{n}{2}$
- 2 more bridges: $C \cdot \frac{n}{4}$
- "Only" 3 bridges: $C \cdot \frac{n}{4}$

\[ \text{Swap nodes: } A < C \cdot \frac{n}{4} \]

(\& recursively the same)
Every node ascends only. Weight per level: less than full tree case.
We get a full tree: depth $\log h$.
$O(n \cdot \log h)$.

See web notes for analysis (which isn't hard).
Decomposing into convex pieces

$r$ reflex vertices
$r$ rays (bisectors)
$r+1$ convex pieces

Best possible:
resolve 2 reflex vertices with 1 diagonal

$$\frac{r-1}{2} + 1$$ pieces
\[ \leq r + 1 \quad \text{with arbitrary borders (rays)} \]
\[ \geq \frac{r^2}{2} + 1 \quad \text{with diagonals} \]

In fact with diagonals we can always get \( \leq 2r + 1 \)

- Start with any convex partition w/ diagonals
- Process diagonals incrementally
  - For every diagonal:
    - Remove it if convexity is preserved at its 2 endpoints
\[ \leq r + 1 \quad \text{with arbitrary borders (rays)} \]
\[ \geq \frac{r^2}{2} + 1 \quad \text{with diagonals} \]

In fact with diagonals we can always get \( \leq 2r + 1 \)

- Start with any convex partition w/ diagonals
- Process diagonals incrementally
  - For every diagonal: remove it if convexity is preserved at its 2 endpoints
\[ \leq r + 1 \quad \text{with arbitrary borders (rays)} \]
\[ \geq \frac{r^2}{2} + 1 \quad \text{with diagonals} \]

In fact, with diagonals we can always get \( \leq 2r + 1 \)

- Start with any convex partition w/ diagonals
- Process diagonals incrementally
  - For every diagonal: remove it if convexity is preserved at its 2 endpoints
\leq r+1 \quad \text{with arbitrary borders (rays)}
\geq \frac{r^2}{2} + 1 \quad \text{with diagonals}

In fact with diagonals we can always get \leq 2r+1

- Start with any convex partition w/ diagonals
- Process diagonals incrementally
  - For every diagonal: remove it if convexity is preserved at its 2 endpoints
\[ \leq r + 1 \quad \text{with arbitrary borders (rays)} \]
\[ \geq \frac{r^2}{2} + 1 \quad \text{with diagonals} \]

In fact with diagonals we can always get \( \leq 2r + 1 \)

- Start with any convex partition w/ diagonals
- Process diagonals incrementally
  - For every diagonal: remove it if convexity is preserved at its 2 endpoints
\[ \leq r + 1 \quad \text{with arbitrary borders (rays)} \]
\[ \geq \frac{r^2}{2} + 1 \quad \text{with diagonals} \]

In fact with diagonals we can always get \( \leq 2r + 1 \)

- Start with any convex partition w/ diagonals
- Process diagonals incrementally
  - For every diagonal:
    - Remove it if convexity is preserved at its 2 endpoints
\[ \leq r + 1 \quad \text{with arbitrary borders (rays)} \]
\[ \geq \frac{r^2}{2} + 1 \quad \text{with diagonals} \]

In fact with diagonals we can always get \( \leq 2r + 1 \)

- Start with any convex partition w/ diagonals
- Process diagonals incrementally
  - For every diagonal: remove it if convexity is preserved at its 2 endpoints
with arbitrary borders (rays)

$$\leq r+1$$

with diagonals

$$\geq \frac{r^2}{2}+1$$

In fact with diagonals we can always get \( \leq 2r+1 \)

- Start with any convex partition w/ diagonals
- Process diagonals incrementally
  - For every diagonal:
    - Remove it if convexity is preserved at its 2 endpoints
  * Any endpoint can only "complain" for \( \leq 2 \) diagonals
  * So we will remove all but \( \leq 2r \) diagonals
OPTIMALITY

- Find min number of convex pieces
  - with diagonals: \( O(r^2 n^2) \) .... \( O(r^2 n \log n) \) .... ? project
  - dynamic programming
  - without diagonals: \( O(n + r^3) \) (Chazelle)

- Find min number of guards for given polygon
  - need not use convex decomposition
  - will discuss at some point
How would you verify that some given guards suffice?

4 start by computing visibility polygon of one

How?
Visibility in a cone

Polygon: 🔄
enter on right, going

while angle goes
push edges on stack
\[ e_1, e_2, e_3 \]

if this continues until \( e_k \) great.
Non-trivial case: somewhere angle backtracks.
Suppose "upward".

starts getting complicated.
Non-trivial case: somewhere angle backtracks.

\[ \text{Suppose "upward"} \]

Ignore until path reappears at same angle.

\[ e_1 \, e_2 \, e_3 \, e_m \]

[partial edge]
So far we can handle any upward backtrack.

Eventually the chain shows up again.

\[
\begin{array}{cccccccc}
e_1 & e_2 & w_1 & e_x & w_2 & e_y & e_{y+1} & w_3 & e_z \\
\end{array}
\]

\[
\text{implicit}
\]
So far we can handle any upward backtrack.

If downward backtrack...

Just start popping from stack
Why is $e_Y$ not in stack?  

→ it will get covered eventually

notice all edges in stack point

added a prime just to show it's $\neq 0$ $e_Y$
How can we continue?

1) Forward: push
2) Backtrack: pop
3) Invisible: ignore

\[ e_1, e_2, w_1, e_x, w'_2 \]
How can we continue?

1) Forward: push
2) Backtrack: pop
3) Invisible: ignore

until emerge from window

\{ if you go inside you will create a new sub-window and you will backtrack out anyway \}
every step is local (perhaps via sequence of local pops)

all you need to check:
- am I covering an edge when backtracking?
- am I going through a window?
- Moving forward: easy
- Backtrack up: wait till you return, to move forward
- Backtrack down: pop while backtracking unless you're in your own pocket

$\downarrow$ could resume forward motion
$\downarrow$ could jump to "backtrack up" (Going directly from $\bullet$ to $\bullet$ would have been the same)