CPSC 420/500 Scribe Notes - Jan 18+20
Matthew Nyhus

Continuing on from Monday, Jan 16's lesson:
Lemma: Any linear decision tree that computes F has:height $\geq$ ceil($\log_3(\Sigma_{\text{output t's}} \#F_t)$)
(from previous class; $\#F_t$ is the number of connected components for the given t value)

Theorem Any linear decision tree that computes Element Uniqueness has height $\Omega(n*\log(n))$

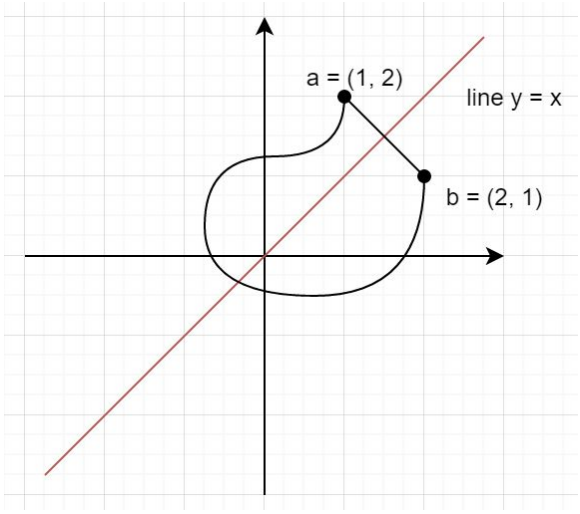
Note that for element uniqueness, the possible t's are t = Yes and t = No. For the t = No case, the solution space is along the $x_i = x_j$ planes. (So, in the 3D case, the t = No case would only be the x = y, x = z and y = z planes.) So, $\#F_{No}$ = 1 since all the planes make one connected component (they all intersect at the origin point, so a path can always be made between any 2 points).

Now we need to know something about the size of $\#F_{Yes}$

Idea:
Are a = (1, 2, 3, …, n) and
b = (2, 1, 3, …, n) in the same connected component?

The answer is No, by the Intermediate Value Theorem (IVT)

Example: consider the 2D Case where a = (1, 2) and b = (2, 1):




Note how it is impossible to get from a to b without passing through the line x = y. (A formal proof would use the IVT)

Now for the general case: let a be a vector of n **unique** numbers and b $\neq$ a be any permutation of x. (The example above is such an example of a and b.) Then, there must be indices i and j such that $a_i < a_j$ and $b_i > b_j$. (This can be seen from the contrapositive: if for every i, j, both $a_i < a_j$ and $b_i < b_j$, then a and b are the same permutation.)

Now, note that any continuous path from a to b must contain a point c such that $c_i = c_j$ (by the IVT). c is a t = NO input (since the indices in c represent a non-unique set of numbers), so a and b cannot be in the same connected component (since such a c can be found for **any** path).

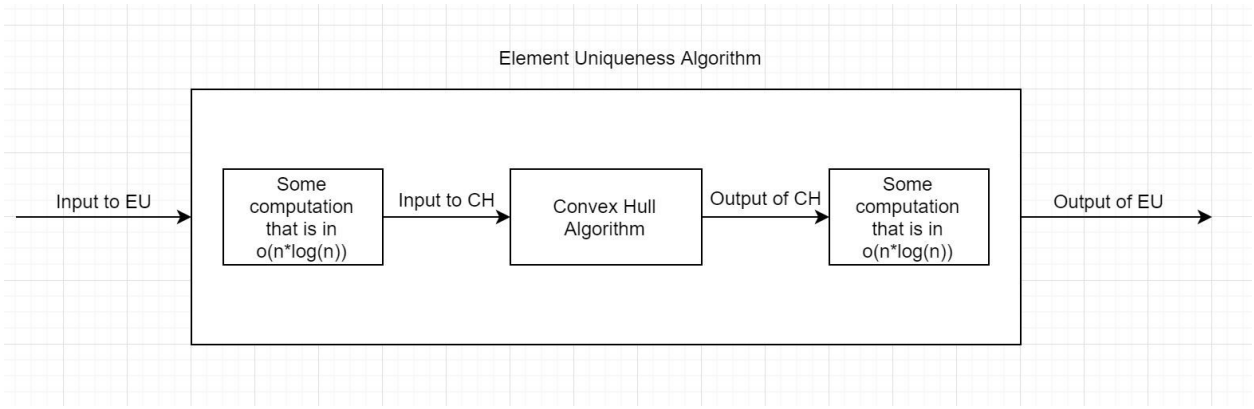
Thus, none of the n! permutations of a are in the same connected component. This implies that $\#F_{yes} \geq n!$

<u>Explanation of the use of the Intermediate Value Theorem:</u>
Let p be a path from a to be: p:[0, 1] -> $\mathbb{R}^n$ where p(0) = a and p(1) = b.
Let $q(t) = p(t)_j - p(t)_i$. Note that q(0) > 0 and q(1) < 0 (by choice of i, j above)
This implies that q(t) = 0 for some 0 < t < 1 (because p is continuous, so q must also be continuous). QED for the idea.

Thus, $\#F_{yes} \geq n!$ => $\Sigma\#F_t \in \Omega(n!)$ => height $\in \Omega(n*\log(n))$
(Note: "=>" is "implies")

Thus, we have proved that any linear decision tree that computes Element Uniqueness has height $\Omega(n*\log(n))$. QED for the theorem.

Using this, we want this reduction for the Convex Hull:




Note, the computations that are in o(n*log(n)) are in ***little-oh*** of n*log(n). Recall that if $f(n) \in o(g(n))$, then $\forall\, c > 0,\ \exists\, n_0$ so that $\forall\, n \geq n_0$, $f(n) < c*g(n_0)$.

This would prove that $\Omega(\text{CH}) \in \Omega(\text{Elem. Un})$
Since we know that Elem Un runs in $\Omega(n*\log(n))$ (from above), this would prove:
$\Omega(\text{CH}) \in \Omega(n*\log(n))$

(We didn't find this reduction in class - try it yourself!)

**General Case:**
Linear decision trees are 1st order Algebraic decision trees (ADT). $d^{th}$-order ADT are like LDT but the functions in the internal nodes are $d^{th}$ order polynomials.

Ex: $3x_1^2 - 2x_1x_2 - 7x_1x_3$ … is a 2nd order polynomial

The number of connected components of the input space that can reach the same leaf is $\leq d(2d-1)^{n+h-1}$ where h is the height of the decision tree. (A proof of this statement was not given - it is beyond the scope of the course.)

We know that the number of leaves in a degree d Algebraic Decision Tree of height h is at most $3^h$. By the fact cited above, the set of inputs that reach a leaf has at most $d*(2d-1)^{n+h-1}$ connected components. Since a decision tree of height h has at most $3^h$ leaves, the number of connected components represented at the leaves of the tree is at most $3^h*d*(2d-1)^{n+h-1}$. For a function F with $C(F) = \Sigma_{\text{outputs } t} \#F_t$, the number of of components represented at the leaves must be at least C(F). So $3^h*d*(2d-1)^{n+h+1} \geq C(F)$. This is equivalent to $(6d-3)^h \geq C(F) / (d(2d-1)^{n-1})$ which is equivalent to $h \geq \log_{6d-3}(C(F) / (d(2d-2)^{n-1}))$. Since C(F) has size n!, the right-hand side is asymptotically $\Omega(\log_d(n!) - n) = \Omega(n*\log_d(n))$.

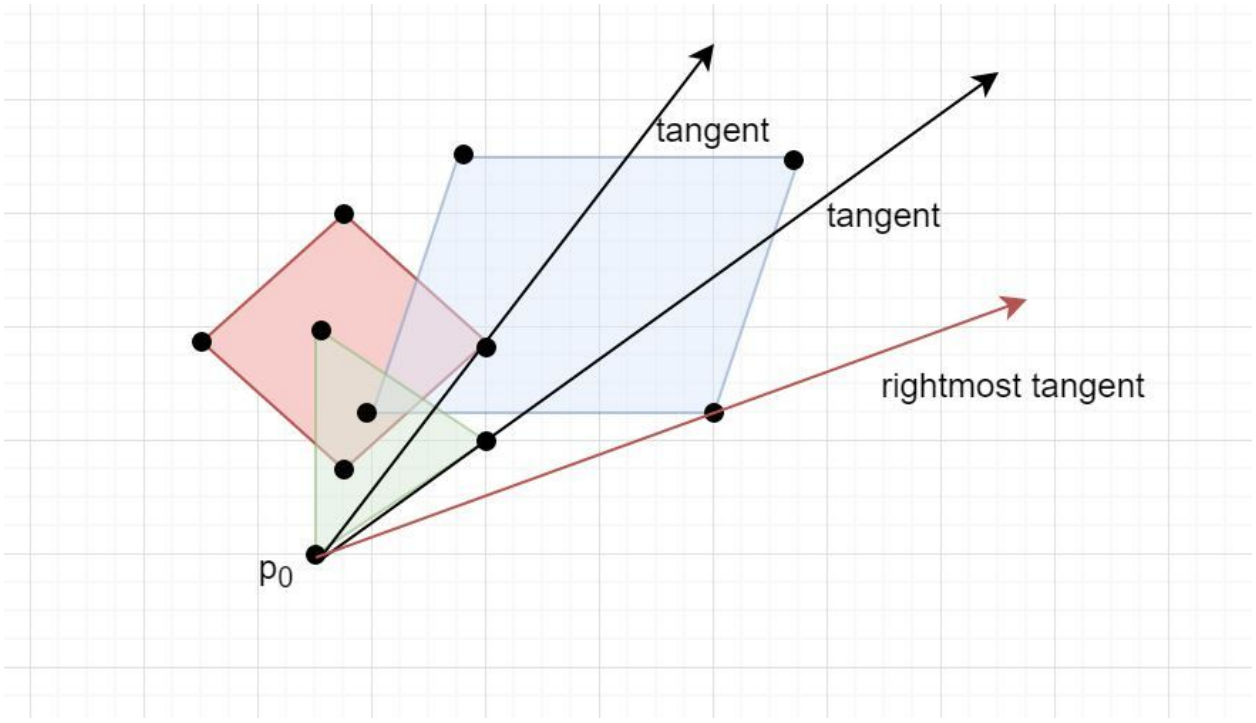
**<u>Chan's Algorithm</u>**
Finding the convex hull in O(n*log(h)) where h is the size of the convex hull

This algorithm is a combination of Jarvis' Mach and Graham's Scan.
Given n points in the set P <u>and</u> a guess h for the number of hull vertices (assume this guess exists; we'll deal with it later):

(1) Divide points into ceil(n/h) groups of size h (except the last group which can be smaller)
(2) Use Graham's scan on each group to find the convex hull of the group
(3) Find the point in P with the smallest y-coordinate, $p_0$; i = 0
(4) Do gift wrapping (Jarvis March) for h steps, but cleverly:
    (a) Find the right tangent from $p_i$ to each of the n/h convex hulls
    (b) $p_{i+1}$ = rightmost of these tangent points
    (c) if $p_{i+1} = p_0$ output HULL (the set of $\{p_0, \ldots, p_i\}$)
(5) Output "h is too small!"

A quick note on 4a), since it contains the clever use of Jarvis' march. Unlike the typical Jarvis' march, Chan's algorithm uses the information from the n/h convex hulls to make a faster choice

of the next hull point. Here's a diagram:




In the diagram, the red, blue, and green highlighted areas represent the n/h convex hulls. Note that n = 11, and the guess for h is 4. So, there are 3 convex hulls, with the last one (the green one) containing fewer points. Note also that some points in P may be excluded in this step (in the case where the convex hull of a subset does not contain all the points. (This hasn't happened in this example, but it doesn't affect the algorithm; note that any point that is not in some sub-hull will never be in the final hull).

The right tangent of each of the n/h sub-hulls is found. This can be done, for each hull, in O(log(h)) time using binary search, since they are sorted. (The sorting is not straightforward; going around the convex hulls in order will result in some form of an increase in angle, then a decrease, then an increase, like a loop. However, since it only loops once, this is still sorted enough to allow for binary search.)

Finding the right tangent needs to be done for each convex hull, so in total, this will take O((n/h)*log(n)) time. There will then be n/h tangents to choose from; we can select the rightmost in n/h time for 4b). Thus, the total runtime of one iteration of the loop in 4) takes O((n/h)*log(h)) time, and from it we get the next hull point of the entire set.

Runtime for one pass:

(1) runs in O(n); (2) runs in O(n*log(h)); (3) runs in O(n); (4) runs for h iterations, where each iteration runs in O((n/h)*log(h)), for a total runtime of O(n*log(h)).

So, one pass of Chan's algorithm runs in O(n*log(h)).

Now to deal with the guess for h:
Generate guesses for h* (the true number of hull vertices)
Start with h = 4, 16, 256, …, 2^(2^t) for the $t^{th}$ try.
Runtime:

$$\Sigma_{h = 2^{(2^t)} \text{ until } h > h^*} O(n*\log(h)) = \Sigma_{\text{from } t = 1 \text{ to ceil}(\log(\log(h^*)))} O(n*2^t)$$
$$= O(n * \Sigma_{\text{from } t = 1 \text{ to ceil}(\log(\log(h^*)))} 2*t)$$
$$= O(n*2^{(\text{ceil}(\log(\log(h^*))) - 1)}$$
$$= O(n*\log(h^*))$$

Note that the ceil can be ignored, since it adds at most 2^1 which is just a constant factor.

Thus, the runtime of Chan's algorithm, overall for all the passes, is still O(n*log(h*))