CS633 Lecture 06
Linear Programming

Jyh-Ming Lien

Dept of Computer Science

George Mason University

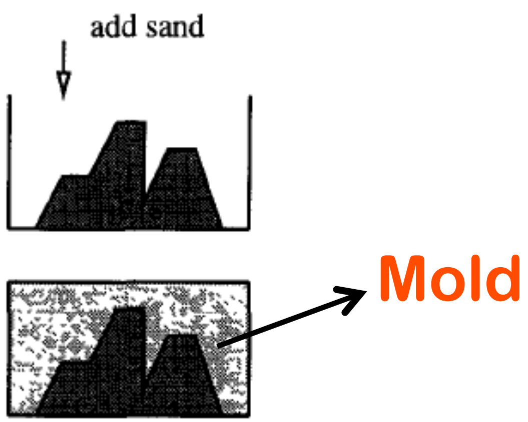
Based on Chapter 4 of the textbook
And Ming Lin's lecture note at UNC

Linear Programming

- Reading: Chapter 4 of the Textbook
- Driving Applications
  - Casting/Metal Molding
  - Collision Detection
- **Randomized Algorithms**
  - Smallest Enclosing Discs

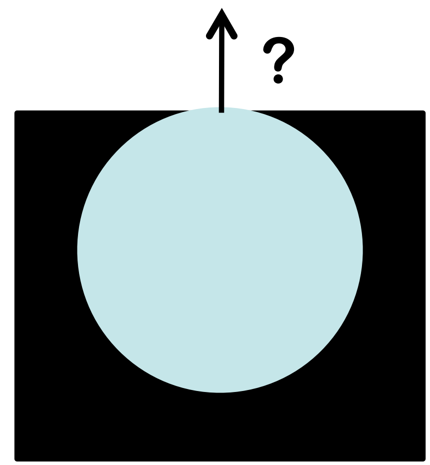
Casting

- Liquid metal is poured into a mold, it solidifies, and then the object shape is formed and the object is removed.



Casting

- Not all objects of different shapes can be removed
  - For example, a sphere



castable?

Castability

- Problems: Whether an object can be manufactured by casting; if so, find the suitable mold
  - Before you learn about this chapter
    - Build a mold $500
    - Build an object using the mold $500
    - Find out that you cannot retrieve the object from the mold
    - Repeat above until you remove the object from the mold

    Repeat 1000 times
  - After you learn about this chapter
    - Scan the object
    - Analyze the castability
    - Save $1 million

Transform to a Geometric Problem

- The shape of cavity in the mold is determined by the shape of the object, but different orientation can be crucial
  - The object must have a horizontal top facet

Transform to a Geometric Problem

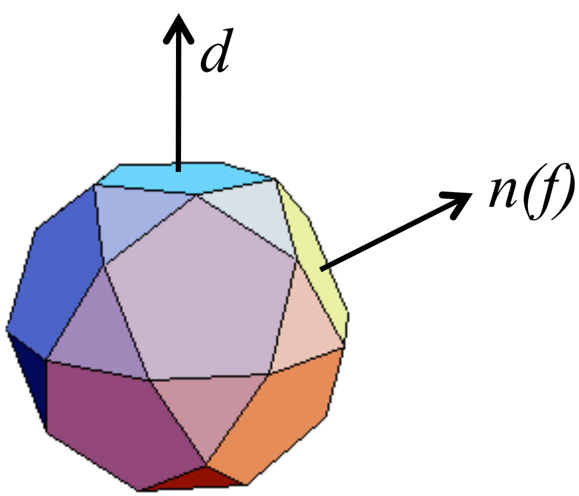
- Let *P*, object to be casted, be a 3D polyhedron bounded by planar facets with a designated top facet
  - Assume: the mold is rectangular block with a cavity that corresponds exactly to *P*.

- Problem: Decide whether a direction $\underline{d}$ exists s.t. *P* can be translated to infinity in direction $\underline{d}$ without intersecting interior of of the mold.

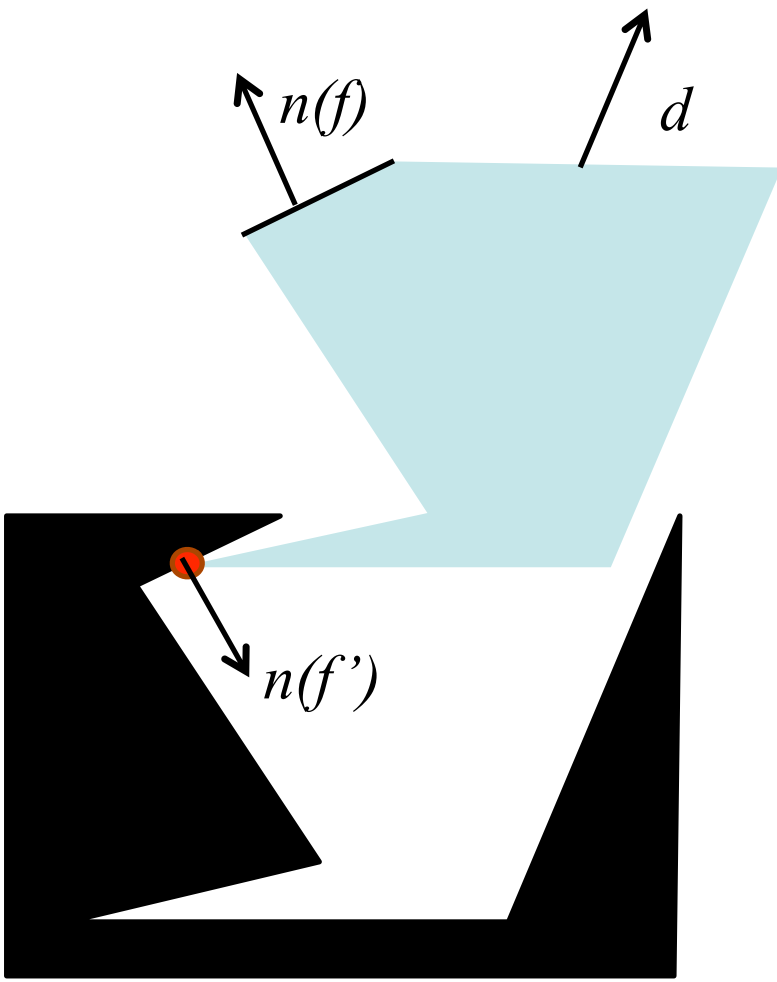
Try each top face and answer:
Can we remove the cast using this top face?
If so, what is the direction, $\underline{d}$?

Problem Analysis

- The polyhedron P can be removed from its mold by a translation in direction $\underline{d}$ if and only if $\underline{d}$ makes an angle of at least 90° with the outward normal of all ordinary facets of P.



Problem Analysis



When in collision:

The angle between $n(f')$ and d must be larger than 90°

The angle between $n(f)$ and d must be smaller than 90°

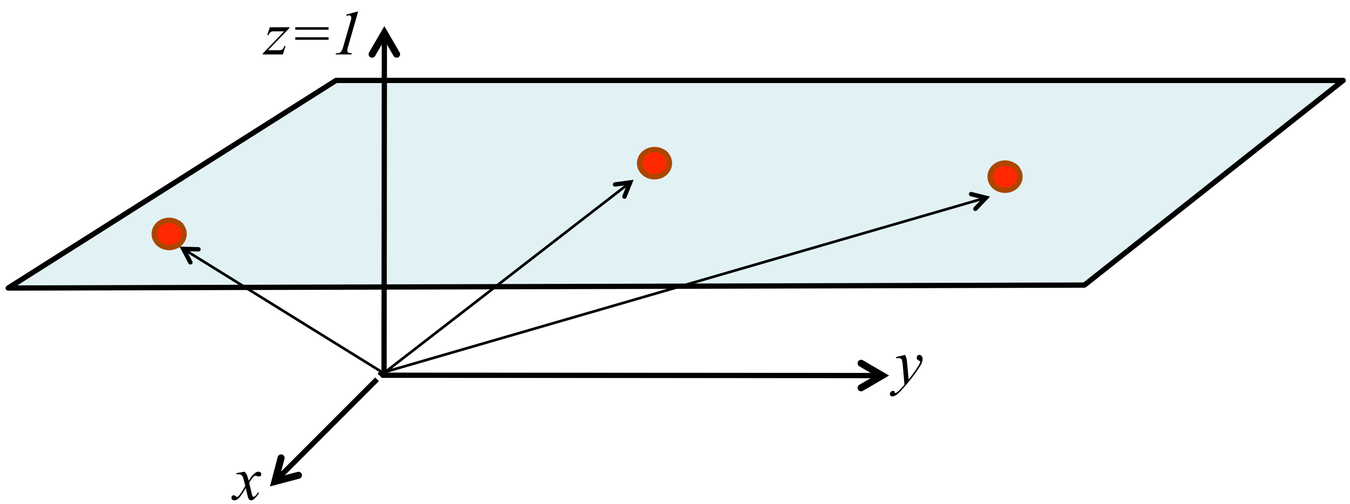
Problem Analysis

- Let $\underline{n} = (n_x, n_y, n_z)$ be the outward normal of an ordinary facet. The direction $\underline{d} = (d_x, d_y, 1)$ makes an angle at least 90° with $\underline{n}$ if and only if the dot product of $\underline{n}$ and $\underline{d}$ is non-positive:

$$n_x d_x + n_y d_y + n_z \leq 0$$

Problem Analysis

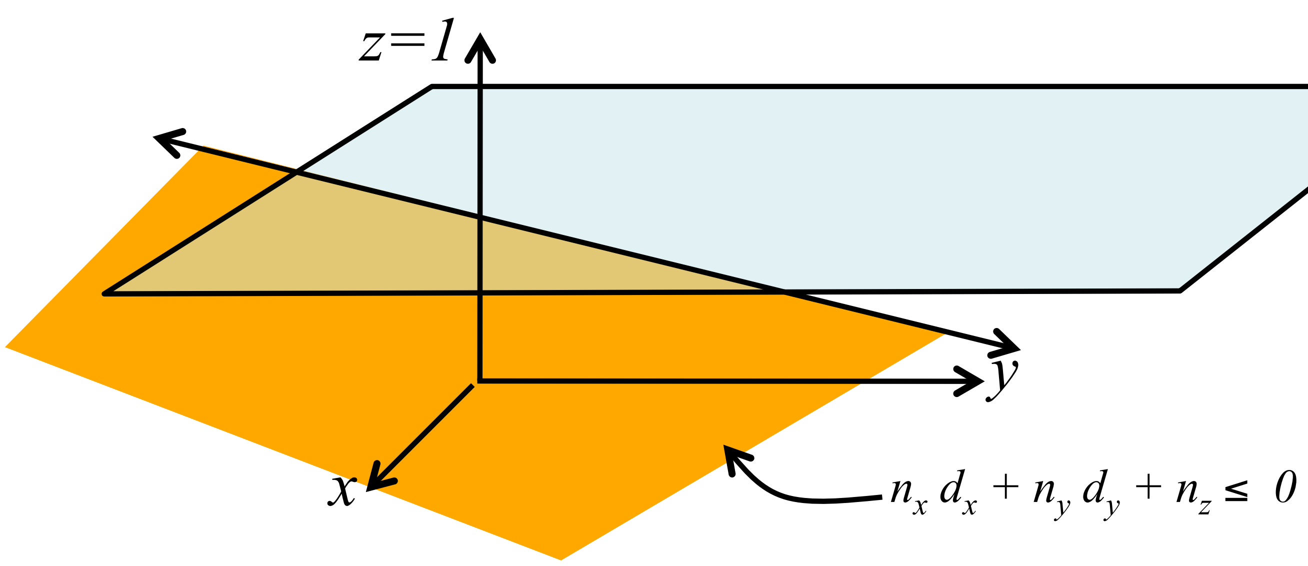
- Representing the direction as $\underline{d} = (d_x, d_y, 1)$
  - Unique upward direction
  - Using fewer variables
    - Reduce the problem from 3D to 2D



Problem Analysis

$$n_x d_x + n_y d_y + n_z \leq 0$$

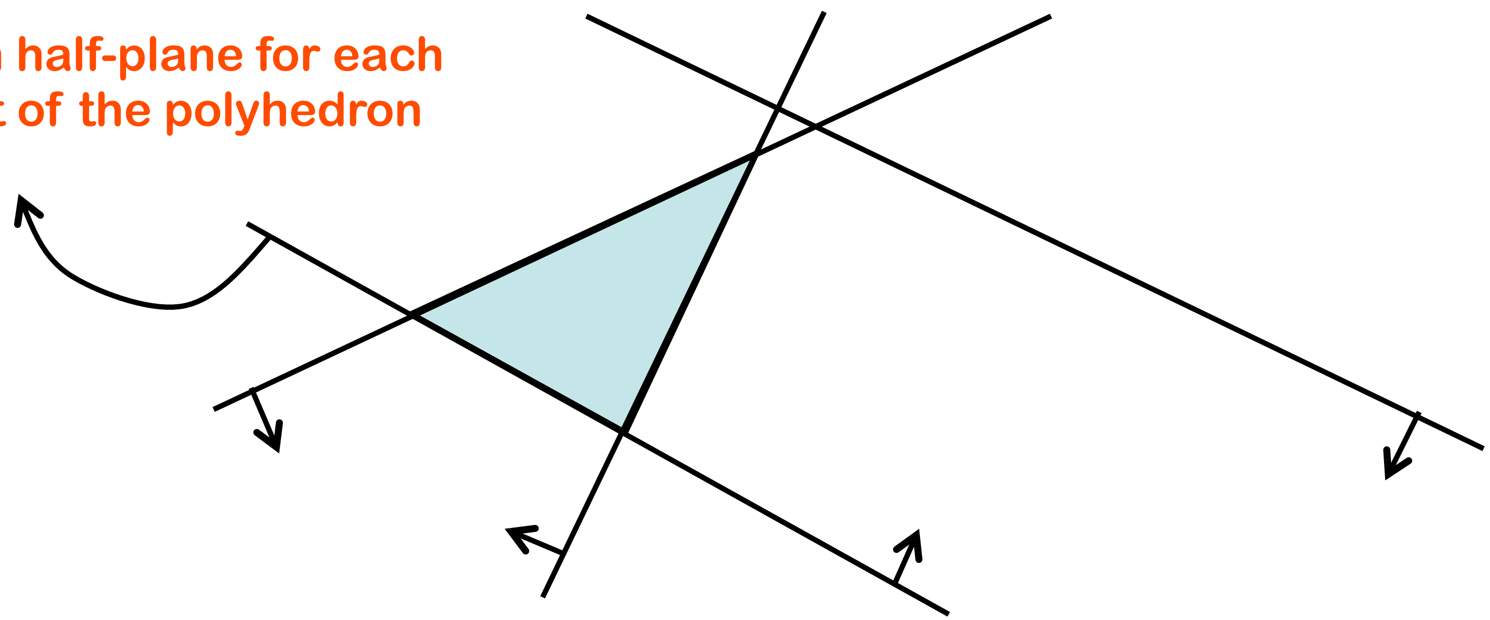
- This describes a half-plane on the plane $z=1$, i.e. the area to the left or right of a line on the plane.



Problem Analysis

- Casting problem: given a set of half-planes, find a point in their common intersection or decide if the common intersection is empty.

Each half-plane for each facet of the polyhedron



Half-Plane Intersection

- Let $H = \{h_1, h_2, \ldots, h_n\}$ be a set of linear constraints in two variables, i.e. in the form:

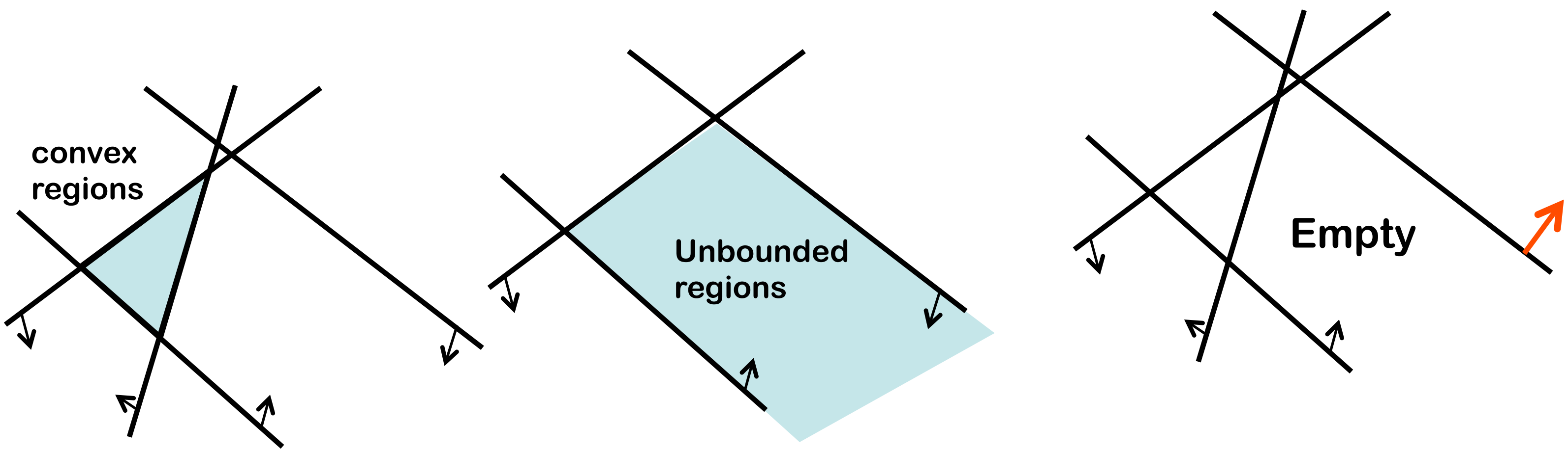
  $$a_i x + b_i y \leq c_i$$

  where a_i , b_i and c_i are constants, s.t. at least one of a_i and b_i is non-zero.

- **Problem**: Find the set of all points $(x,y) \in R^2$ that satisfy all n constraints at the same time; i.e. find *all* the points lying in the common intersection of the half-planes in H.

Type of Intersections

- Convex regions bounded by at most n edges (half-planes / lines)
  - Degenerate cases: a line or point
- Unbounded regions
- Empty



Half-Plane Intersection

A Divide-n-Conquer approach

Input: A set H of n half-planes in the plane
Output: A convex polygonal region $C := \cap_{h \in H} h$

1. if card(H) = 1 *(a plate?)*
2. then $C \leftarrow$ the unique half-plane $h \in H$
3. else Split H into sets H_1 and H_2 of the size $(n/2)$ and $(n/2)$
4. $C_1 \leftarrow$ IntersectHalfPlanes (H_1)
5. $C_2 \leftarrow$ IntersectHalfPlanes (H_2)
6. $C \leftarrow$ IntersectConvexRegions(C_1, C_2)

Intersection of Two Polygons

- How to compute intersection of two polygons?
  - *Using line-segment intersection*
  - *Using doubly-connected edge list*
  - *Updating the facets*

Run Time Analysis

- Computing intersections of two overlays takes $O((n+k) \log n)$ time, where k is the number of intersection points between edges of C_1 and edges of C_2 and $k \leq n$

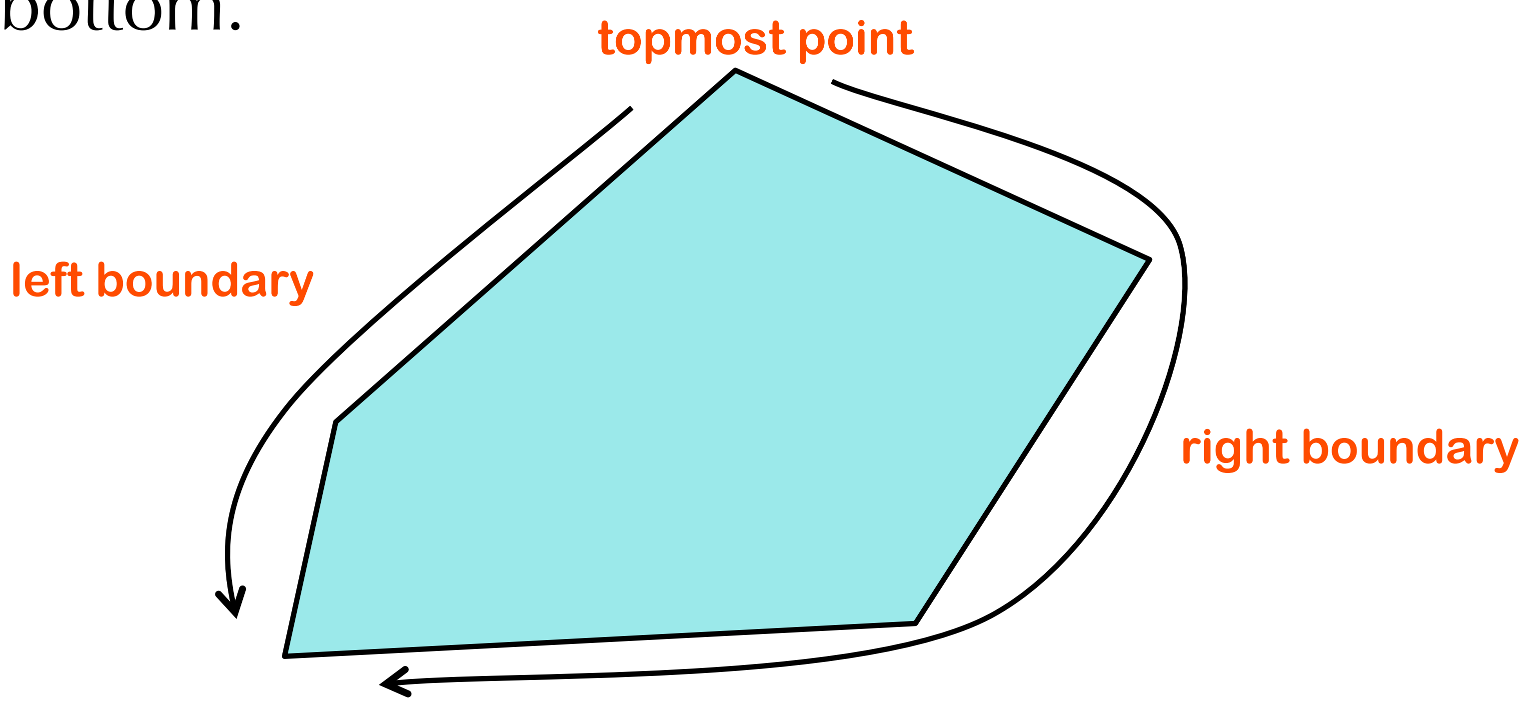
- $T(n) = O(1)$, if $n = 1$
- $T(n) = O(n \log n) + 2\, T(n/2)$, if $n > 1$

$\Rightarrow T(n) = O(n \log^2 n)$

Can we do better?

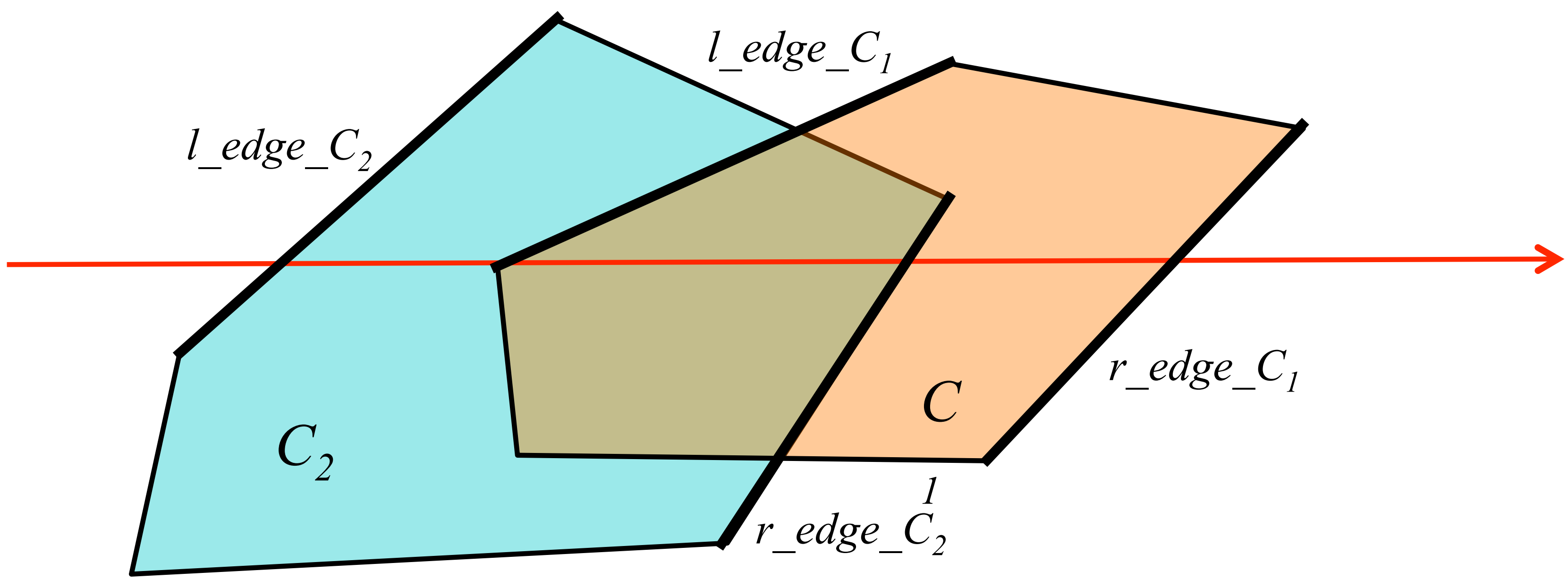
Another Plane-Sweep

- Store left/right boundary of C as sorted lists of half-planes, $L_{left}(C)$ & $L_{right}(C)$, in order from top to bottom.



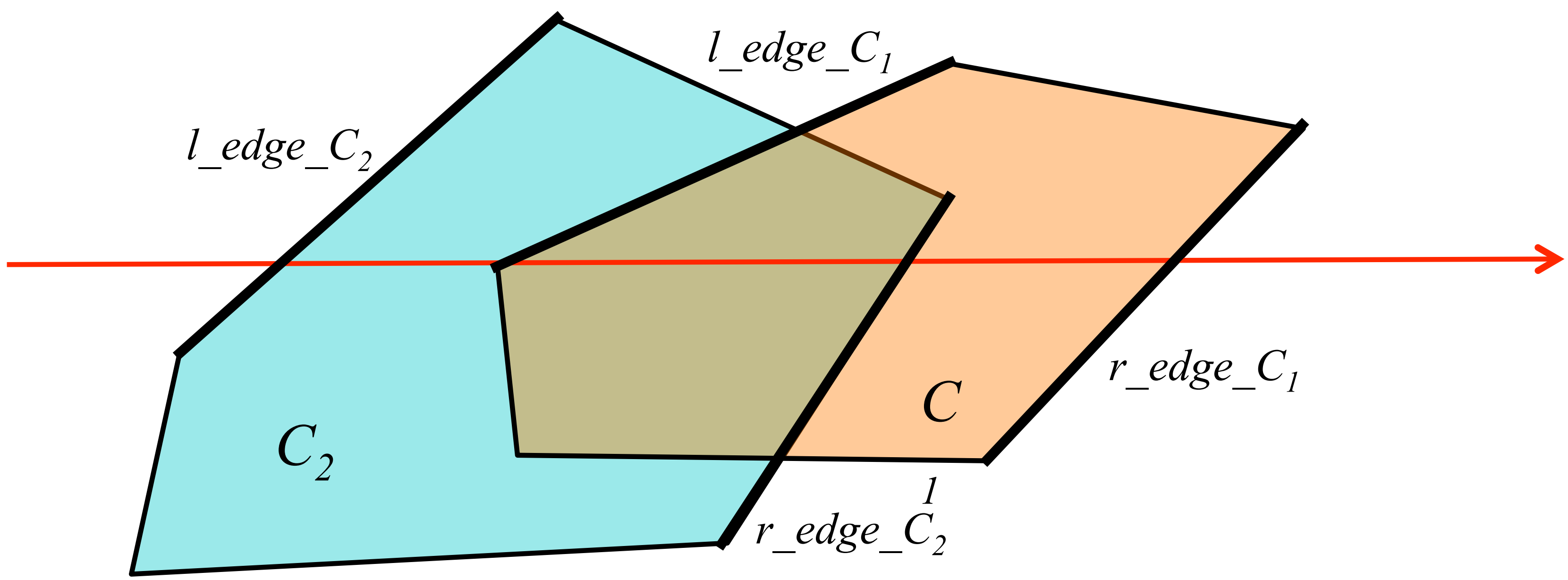
Another Plane-Sweep

- Plane-Sweep: maintain edges of C_1 & C_2 intersecting the sweep line. There are at most four. Use pointers: *l_edge_*C_1, *r_edge_*C_1, *l_edge_*C_2, *r_edge_*C_2.



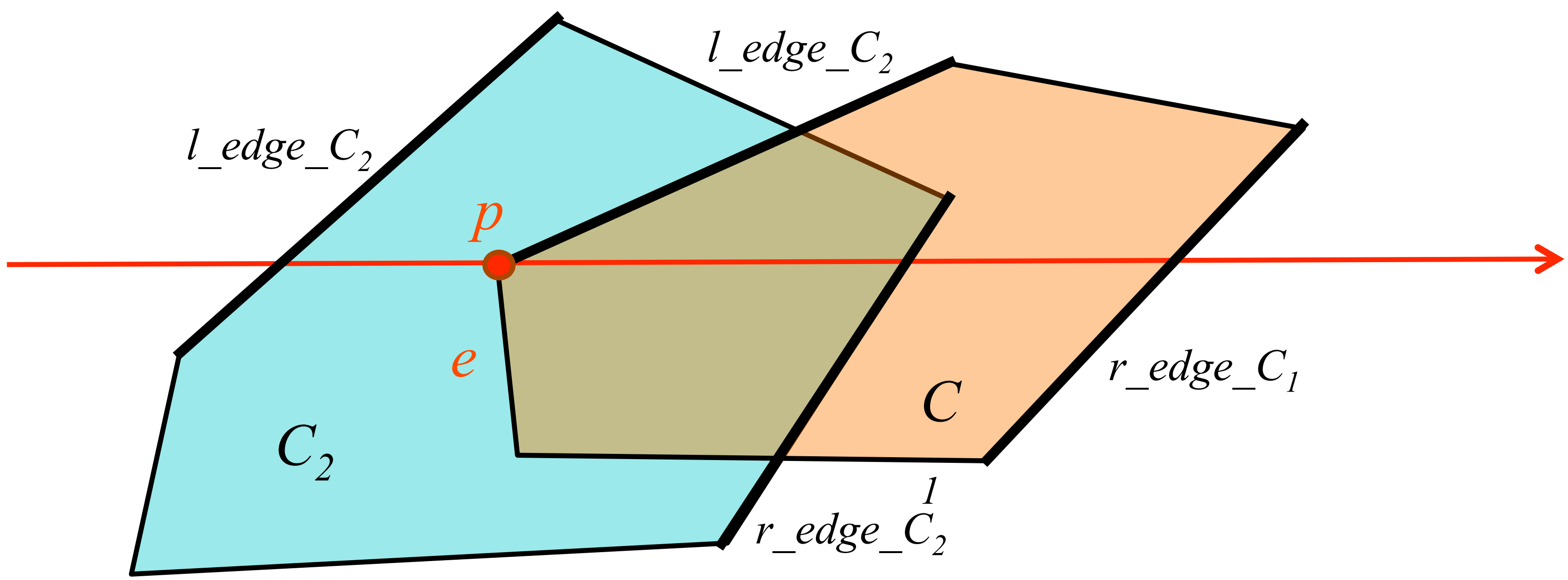
Another Plane-Sweep

- Plane-Sweep: Events are the vertices of the convex polygon points



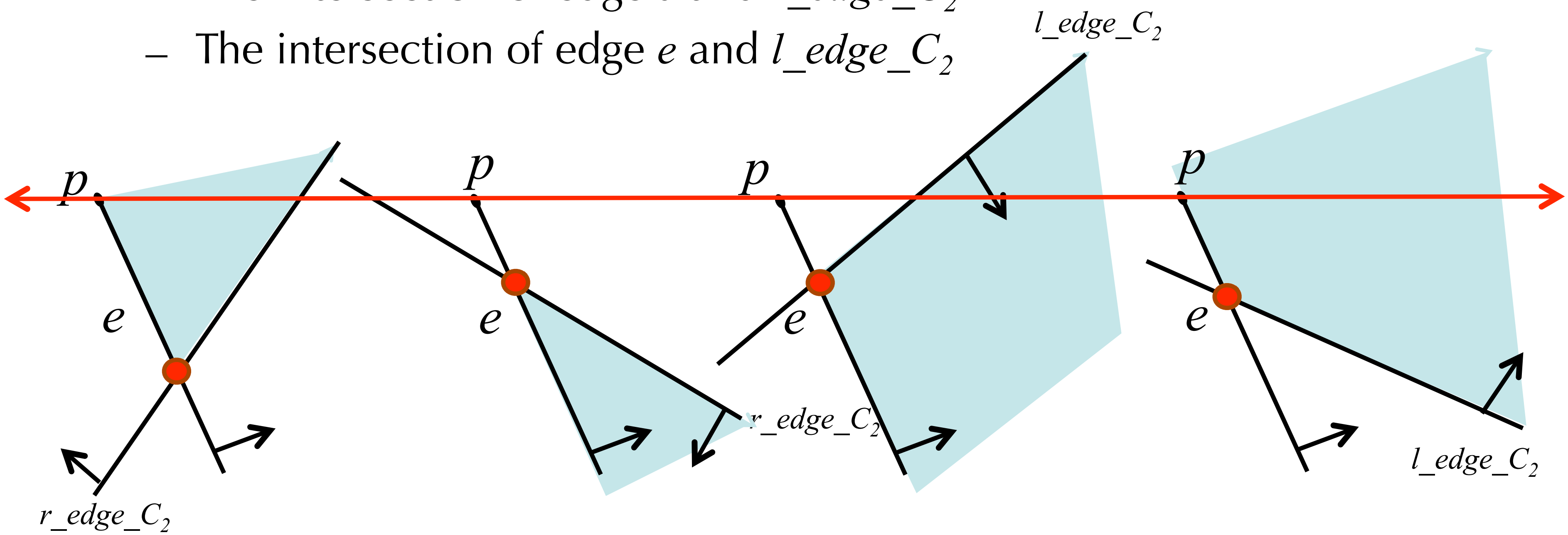
Another Plane-Sweep

- Plane-Sweep: Assume *l* is at the upper endpoint *p* of an edge *e* of $l\_edge\_C_1$
  - *p* will be a vertex of the new convex object



Another Plane-Sweep

- Plane-Sweep: Assume *l* is at the upper endpoint *p* of an edge *e* of $l\_edge\_C_1$
  - *p* will be a vertex of the new convex object
  - The intersection of edge *e* and $r\_edge\_C_2$
  - The intersection of edge *e* and $l\_edge\_C_2$



Half-Plane Intersection

- At each event point, some new edge *e* appears on the boundary. To handle edge *e*, we first check whether *e* belongs to C_1 or C_2, and whether it is on the left or right boundary, and then call appropriate procedure

- According to the handling of each case, we add the appropriate half-planes to the intersection of C_1 & C_2. All cases can be decided in constant time
  - Keep track of left boundary and right boundary in the new convex region

Algorithm Analysis

- It takes constant time to handle an edge. So, the intersection of two convex polygonal regions in the plane can be computed in $O(n)$ time. So, now ……

  $T(n) = O(n) + 2\,T(n/2)$, if $n > 1$

  $\Rightarrow T(n) = O(n \log n)$

- The common intersection of a set of n half-planes in the plane can be computed in $O(n \log n)$ time and linear storage.

Casting Problem: Summary

- Let P be a polyhedron with n facets. In $O(n^2 \log n)$ time and using $O(n)$ storage it can be decided whether P is castable.

- Moreover, if P is castable, a mold and a valid direction for removing P from it can be computed in the same amount of time.

Break time

- Take a 10 min break.

Quiz time

Casting Problem: Summary

- Let P be a polyhedron with n facets. In $O(n^2 \log n)$ time and using $O(n)$ storage it can be decided whether P is castable.

- Moreover, if P is castable, a mold and a valid direction for removing P from it can be computed in the same amount of time.

Algorithm Analysis

- Can we do better?
  - Using convex objects intersection, we find all possible answers
  - But we only need one answer (one remove direction)!

Linear Programming

- Linear Programming/Optimization: finding a solution to a set of linear constraints

Maximize $c_1 x_1 + c_2 x_2 + \ldots + c_d x_d \leq c_i$

Subject to

$$a_{11} x_1 + a_{12} x_2 + \ldots + a_{1d} x_d \leq b_1$$
$$a_{21} x_1 + a_{22} x_2 + \ldots + a_{2d} x_d \leq b_2$$
$$\vdots$$
$$a_{n1} x_1 + a_{n2} x_2 + \ldots + a_{nd} x_d \leq b_n$$

where a_{ij}, b_i and c_i are real numbers and inputs

LP Terminology

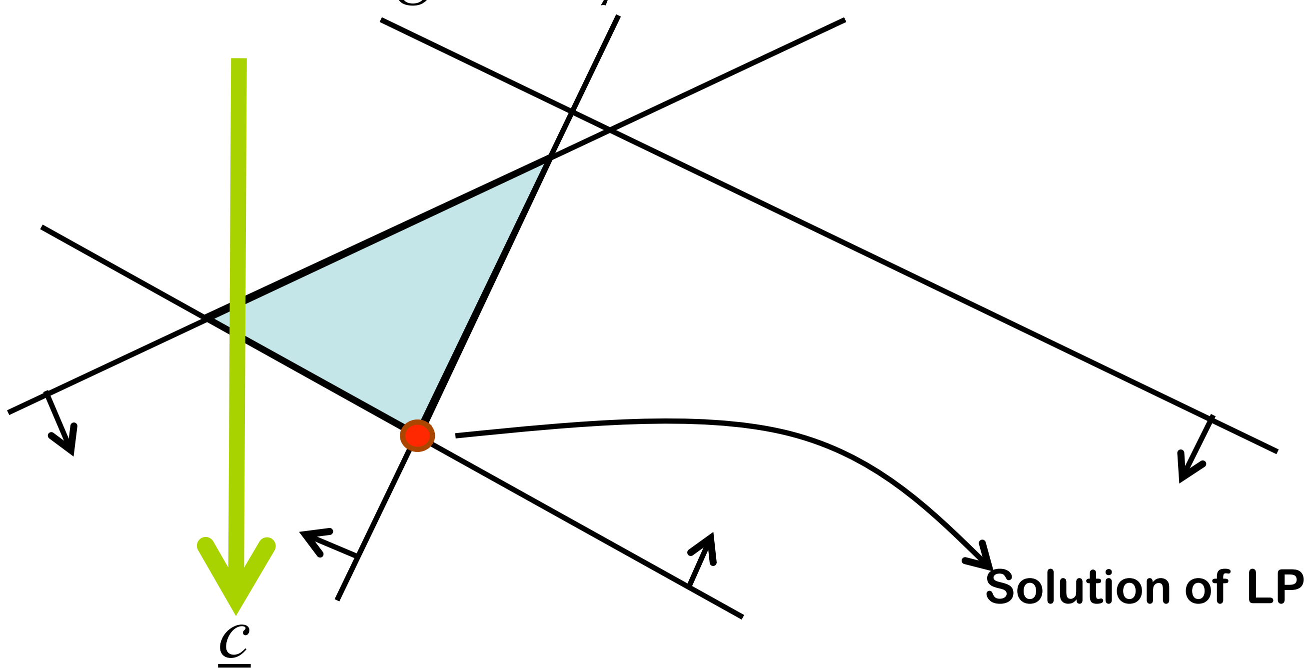
- *Objective Function*: the funct. to be maximized
- *Linear Program:* the objective functions and the set of constraints together
- *Dimension*: the number of variables, d
- *Feasible Regions*: the set of points satisfying all constraints. Points in this region are called "*feasible*" & points outside "*infeasible*".

2D Linear Programming

- Let H be a set of n linear constraints
- The vector defining the obj. func. is $\underline{c} = (c_x, c_y)$
- The objective function is $f_{\underline{c}}(p) = c_x p_x + c_y p_y$

2D Linear Programming

- Problem: find a point $p \in R^2$ s.t. $p \in \cap H$ and $f_{\underline{c}}(p)$ is maximized. Denote the LP by $(H, \underline{c})$ and its feasible region by C.



Types of Solutions to 2D-LP

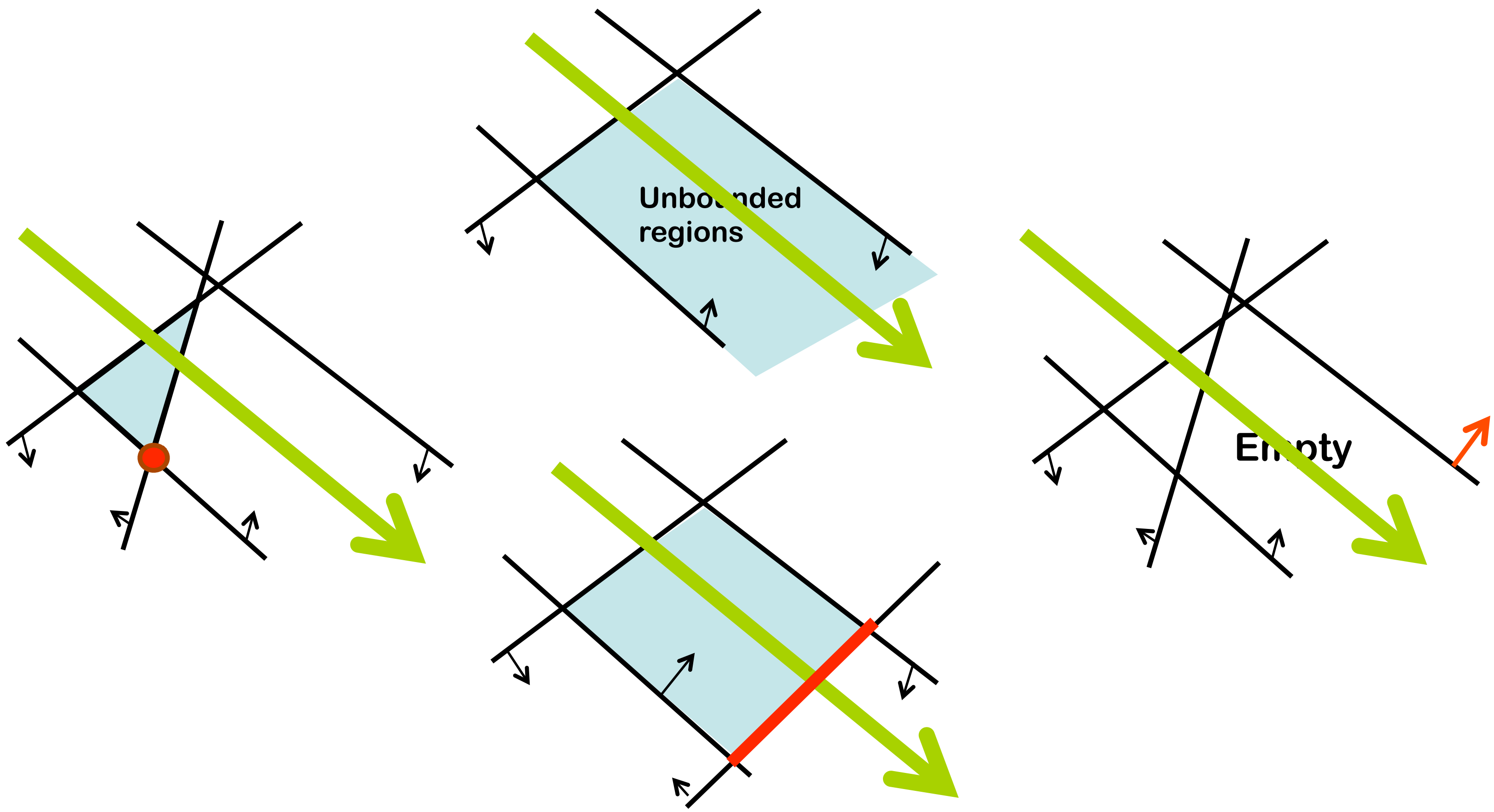
1. LP is infeasible, i.e. no solution

2. The feasible region is unbounded in direction $\underline{c}$. There is a ray p completely contained in feasible region C, s.t. $f_{\underline{c}}$ takes arbitrary large value along p

3. The feasible region has an edge e whose outward normal points in the direction $\underline{c}$. The solution to LP is not unique: any point on e

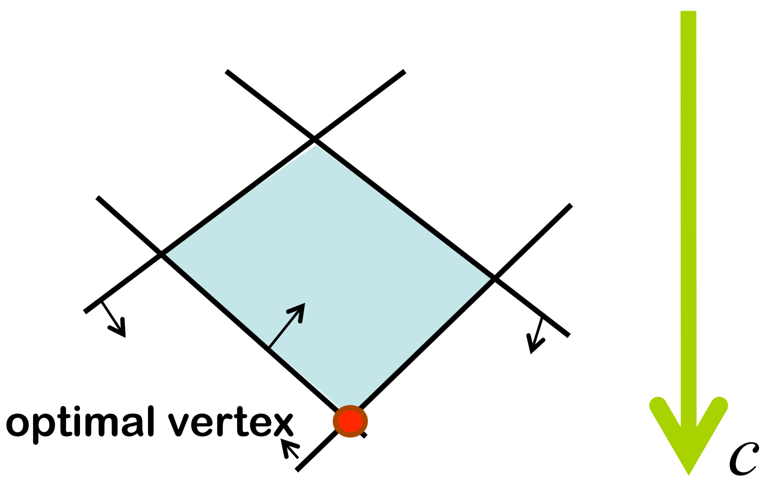
4. There is a unique solution: a vertex v of C that is extreme in the direction $\underline{c}$

Types of Solutions to 2D-LP



Optimal Vertex

- In the case where an edge is a solution, there is an unique solution: lexicographically smallest one
- With this convention, we define "*optimal vertex*" as a vertex of the feasible region



Incremental LP

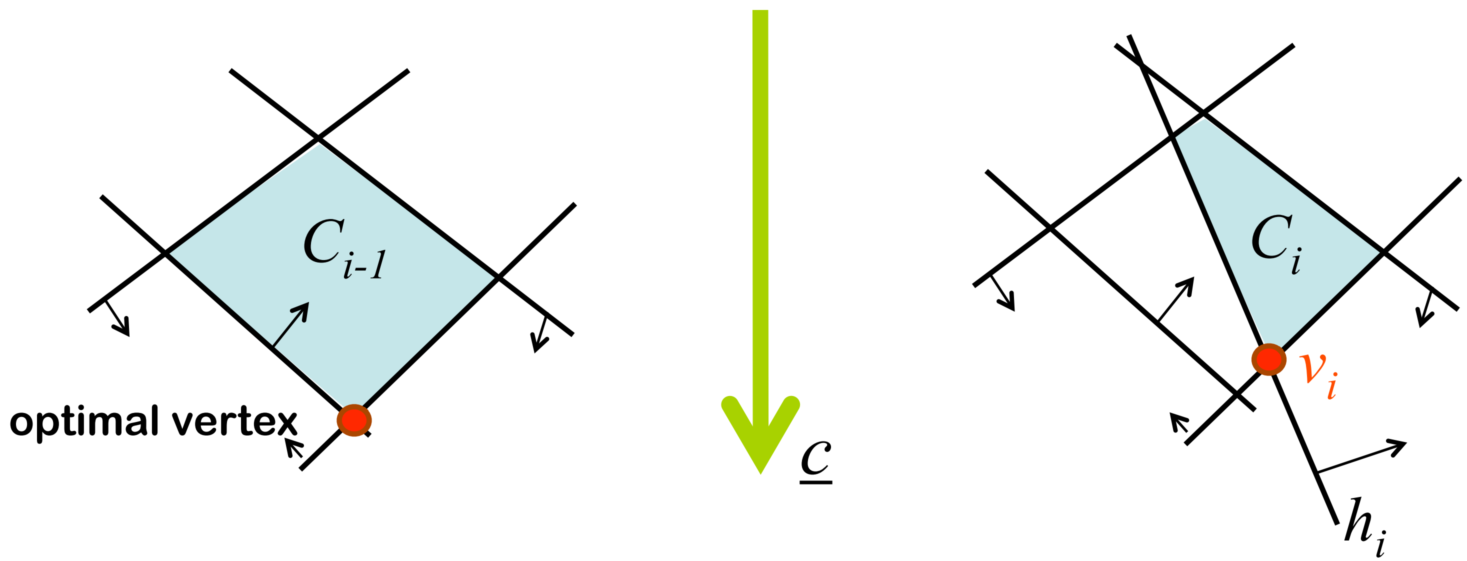
- We can incrementally add one constraint (half-plan or a facet) at a time
- Maintain the optimal vertex of the intermediate feasible regions, except in the case of unbounded LP.

- Constraints considered so far-- $H_i = \{h_1, h_2, \ldots, h_i\}$

- Feasible region so far-- $C_i := h_1 \cap h_2 \cap \ldots \cap h_i$

Incremental LP

- Denote the optimal vertex of C_i by v_i, clearly we have:
  $$C_2 \supseteq C_3 \supseteq C_4 \ \dots \supseteq C_n = C$$
- If $C_i = \phi$ for some i, then $C_j = 0$ for $\forall\, j \geq i$ & LP infeasible

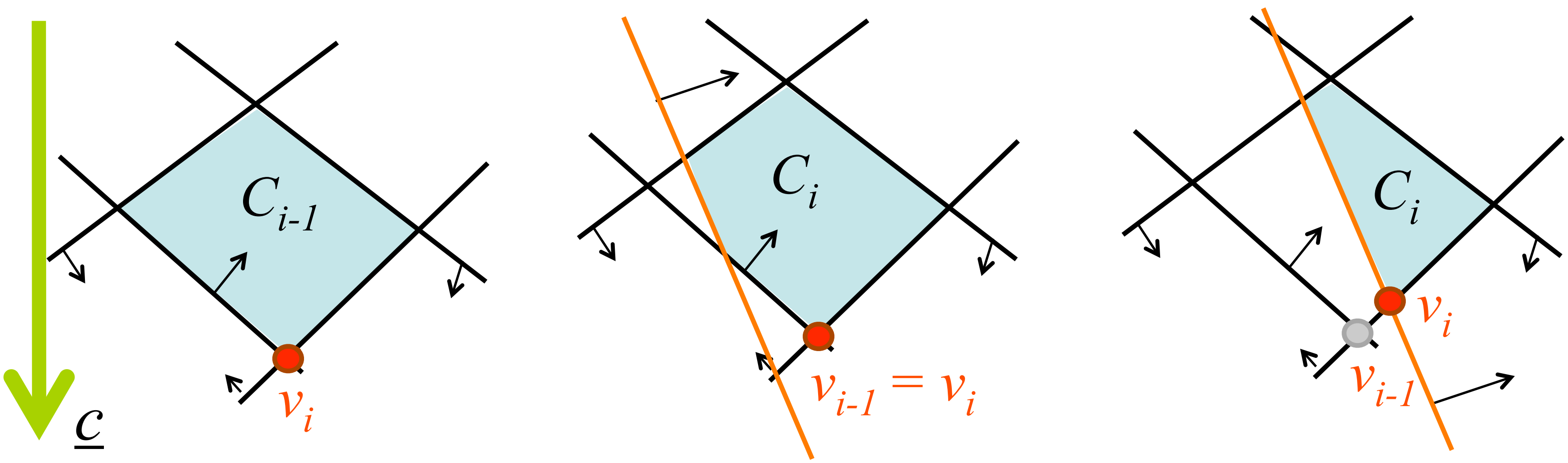


Incremental LP

- We keep 2 half-planes, h_1 and h_2 from H, s.t. ($\{h_1, h_2\}$, $\underline{c}$) is bounded. These half-planes are called certificates that proves (H, $\underline{c}$) is really bounded.

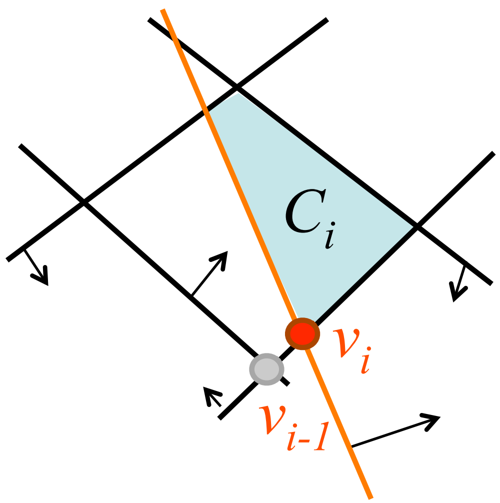
How Optimal Vertex Changes

- Let $2 < i \leq n$, let C_i and v_i be defined as above.
  - If $v_{i-1} \in h_i$, then $v_i = v_{i-1}$
  - If $v_{i-1} \notin h_i$, then either $C_i = 0$ or $v_i \in l_i$ where l_i is the line bounding h_i



How Optimal Vertex Changes

- How do we find the new optimal vertex?
  - Such a point must be on l_i



New 1D LP:

Find the point p on l_i that maximizes $f_{\underline{c}}(p)$, subject to the constraints $p \in h_j$, for $1 \leq j \leq I$

Simplifying to 1D-LP

Maximize $f_{\underline{c}}((x,0))$

Subject to $x \geq x_j$, $1 \leq j < i$ *and* $l_i \cap h_j$ *bounded to left*

$x \leq x_k$, $1 \leq k < i$ *and* $l_i \cap h_k$ *bounded to right*

- This is a 1D-LP. Let

  $x_{left} = max\ 1 \leq j < i$ $\{x_j : l_i \cap h_j$ is bounded to the left$\}$ and

  $x_{right} = min\ 1 \leq k < i$ $\{x_k : l_i \cap h_k$ is bounded to the right$\}$

  The interval $[x_{left} : x_{right}]$ is the feasible region of the 1D-LP. Hence, the LP is infeasible if $x_{left} > x_{right}$, and otherwise the optimal point is either x_{left} or x_{right}, depending on the objective function.

2D-LP

Input: A line program $(H, \underline{c})$ where H is set of n half-planes, $\underline{c} \in R^2$

1. Let h_1, $h_2 \in H$ be the 2 certificate half-planes
2. Let v_2 be the intersection point of l_1 & l_2
3. Let h_3 , ... , h_n be the remaining half-planes of H
4. for $i \leftarrow 3$ to n do
5. if $v_{i-1} \in h_i$
6. then $v_i \leftarrow v_{i-1}$
7. else $v_i \leftarrow$ the point p on l_i that maximizes $f_{\underline{c}}(p)$, subject to the constraints, h_1 , ... , h_{i-1}
8. if p does not exist
9. then Report that LP is infeasible & quit.
10. Return v_n

2D-LP Algorithm Analysis

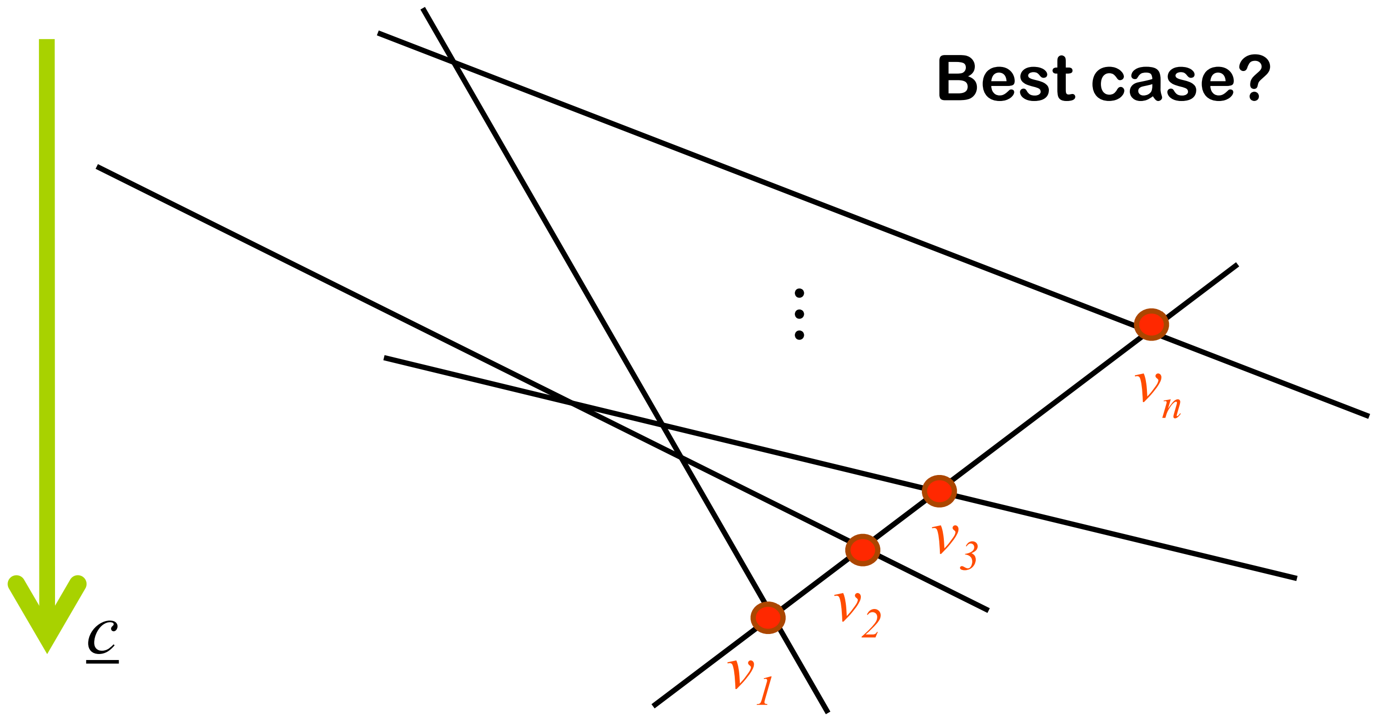
- This algorithm computes the solution to a linear program with n constraints and two variables in $O(n^2)$ time and linear storage
  - The time spent in Stage i is dominated by solving 1D-LP in Step 8, which takes $O(i)$ time

$$\sum_{3 \leq i \leq n} O(i) = O(n^2)$$

- Observation: if we could bound the number of times the optimal vertex changes, then we may be able to get better running time.

Algorithm Analysis

- Worst case for LP: $O(n^2)$



Randomized LP

- For any set of *H* of half-planes, there is a good order to treat them.

- There is no good way to determining an ordering of *H*, so simply pick a random ordering.

- Worst case for RLP: $O(n^2)$, but maybe better!

Random Permutation

Input: An array A[1…n]

Output: The array A[1…n] with the same elements, but rearranged into a random permutation.

1. for $k \leftarrow n$ downto 2
2. do *rndindex* ← RANDOM(k)
3. Exchange $A[k]$ and $A[rndindex]$

- RANDOM() takes k as input & generate a random integer btw *1* and k in constant time

R-LP Algorithm Analysis

- The *average expected* running time over all possibilities is:

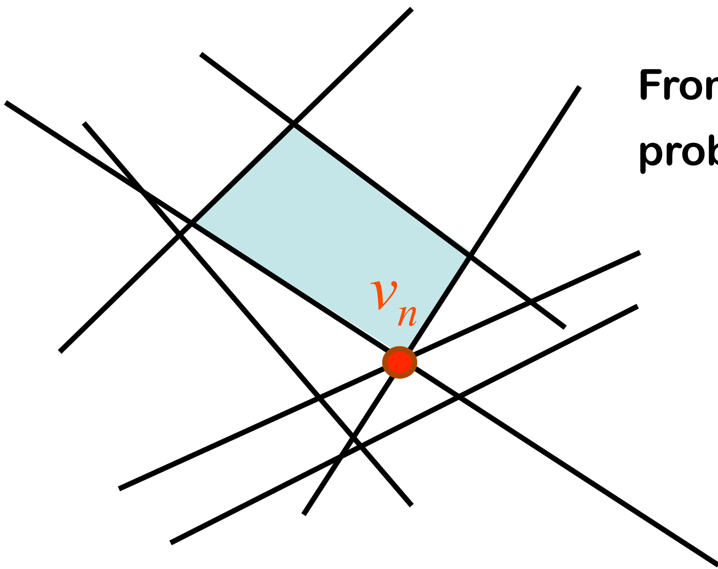
$$E[\sum_{1\le i\le n} O(i)\cdot X_i] = \sum_{1\le i\le n} O(i)\cdot E[X_i]$$
$$= \sum_{1\le i\le n} O(i)\cdot[2/i] = O(n)$$

- X_i is a random variable.
  - $X_i = 1$ if new optimal vertex is needed
  - Otherwise: $X_i = 0$

⇒The 2D-LP problem with n constraints can be solved in $O(n)$ randomized expected time using worst-case linear storage.

What is $\mathbf{E}[X_i]$?

- Backward analysis
  - Instead of analyze how vertices are created
  - We analyze how vertices can be destroyed!
  - These two have same probability



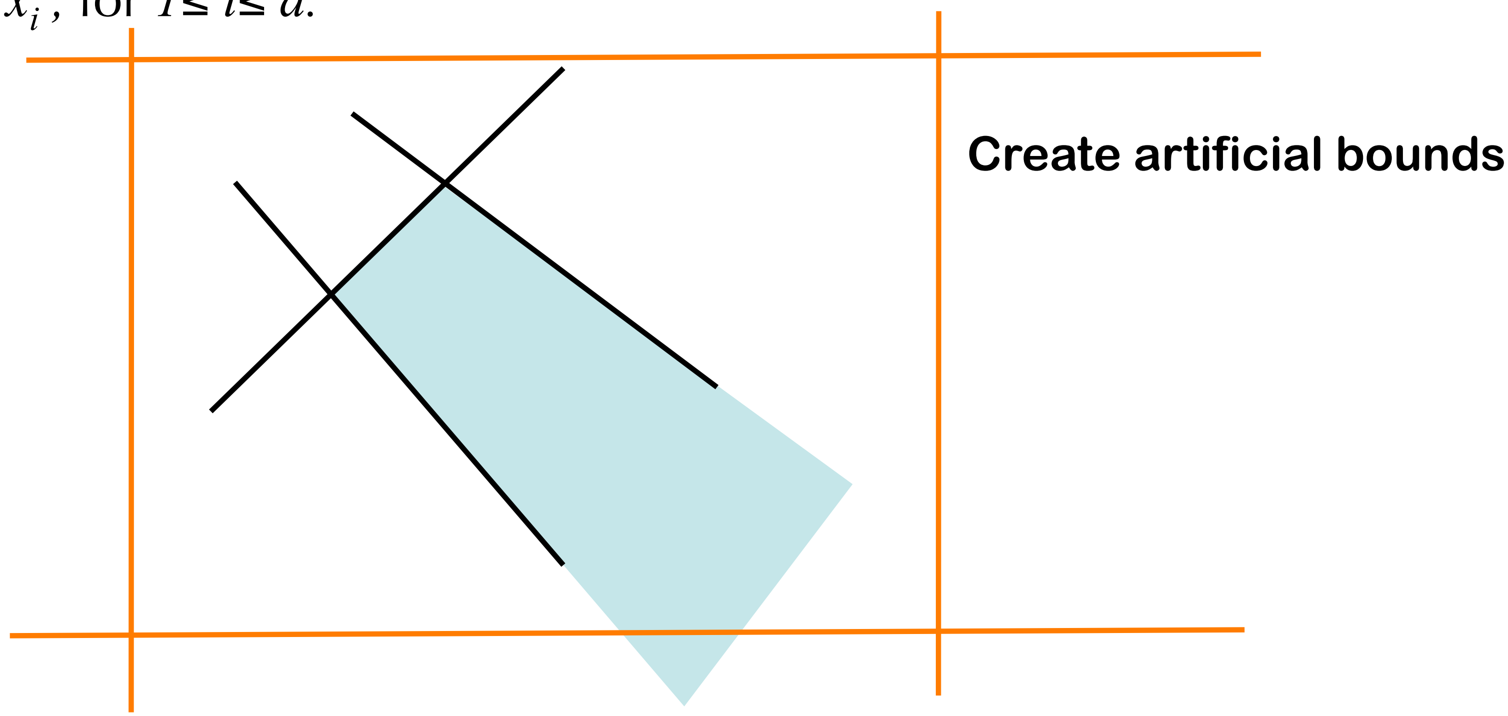
From n constraints, pick one constraint, what is the probability that v_n pick will be destroyed?

$$\mathbf{E}[X_n] \leq 2/n$$

Similarly to destroy v_n $\mathbf{E}[X_i] \leq 2/i$

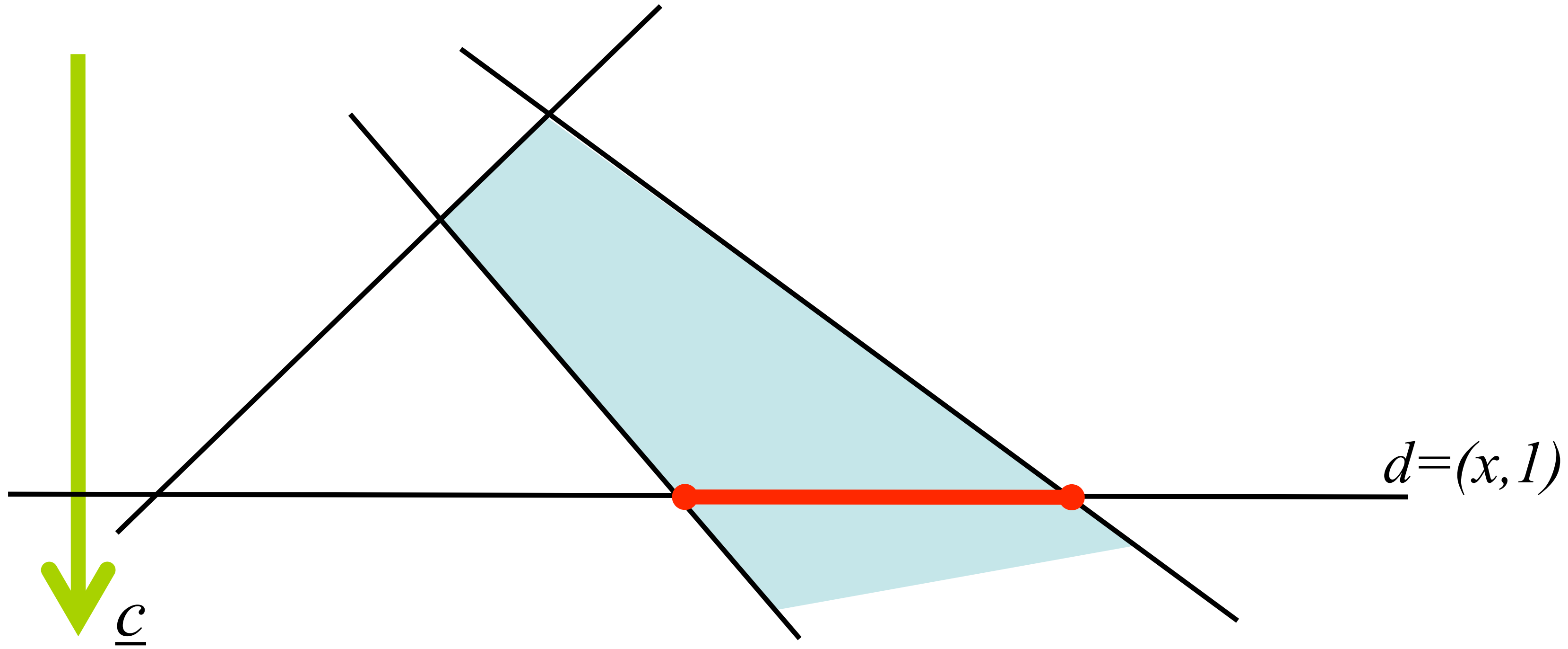
Unbounded LP

- When we only want to know if there is a feasible point
- We have a priori bound *A* on the absolute value of the solution. In such a case, we can add $2d$ constraints of the form $x_i \leq A$, and $-A \leq x_i$, for $1 \leq i \leq d$.



Unbounded LP

- How to find out the unbounded direction *d*?
  - Convert the problem to 1D LP



Casting Problem Summary

- Let P be a polyhedron with n facets. In $O(n^2)$ expected time and using $O(n)$ storage it can be decided whether P is castable.
- Moreover, if P is castable, a mold and a valid direction for removing P from it can be computed in the same amount of time.

LP in Higher Dimensions

- The 3D-LP w/ n constraints can be solved in $O(n)$ expected time using linear storage.
- The d-dimensional LP problem with n constraints can be solved in $O(d!n)$ expected time using linear storage.

⇒ RLP is only useful for lower dimensional problems. Other LP techniques, such as the simplex algorithm, are preferred for higher dimensions.

Collision Detection of Convex Polyhedra

- The problem in 3D can be posed as:

$$
\begin{aligned}
&\textit{Maximize} && 1 \\
&\textit{Subject to} && a_{11}x + a_{12}y + a_{13}z \leq b_1 \\
& && \vdots \\
& && a_{m1}x + a_{m2}y + a_{m3}z \leq b_m \\
& && c_{11}x + c_{12}y + c_{13}z \leq d_1 \\
& && \vdots \\
& && c_{n1}x + c_{n2}y + c_{n3}z \leq d_n
\end{aligned}
$$

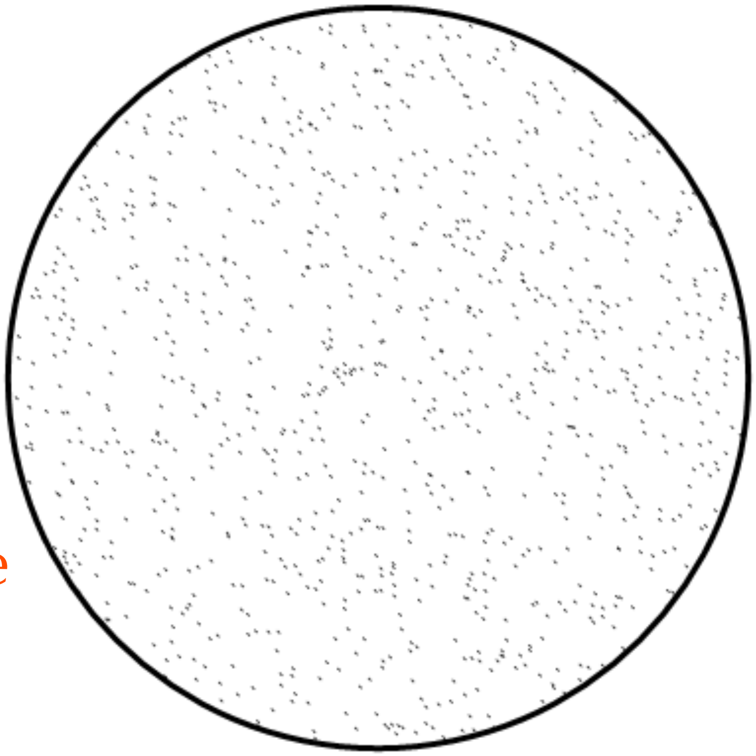
where $(a_{i1}, a_{i2}, a_{i3}, b_i)$ and $(c_{k1}, c_{k2}, c_{k3}, d_k)$ represent the outward normals of the faces of convex polyhedra A & B. If the LP is feasible, then A&B've collided.

Smallest Enclosing Discs

- A robot arm whose base is fixed and has to pick up items at various points and locate them at other points.

- Problem: what would be a good position for the base of the arm?

⇒ A good position is at the center of the smallest disc that encloses all the points.



Transform to a Randomized Algorithm

- Generate a random permutation $p_1 , \ldots , p_n$ of P
- Let $P_i = \{p_1 , \ldots , p_i\}$. We add points one by one, while maintaining D_i, the smallest enclosing disc of P_i.
- Let $2 < i < n$, P_i and D_i be defined as above, we have:
  - if $p_i \in D_{i-1}$, then $D_i = D_{i-1}$
  - if $p_i \notin D_{i-1}$, else p_i lies on the boundary of D_i

MiniDisc(*P*)

Input: A set P of n points in the plane

Output: The smallest enclosing disc for P

1. Compute a random permutation $p_1 , \ldots , p_n$ of P
2. Let D_2 be the smallest enclosing disc for $\{p_1 , p_2\}$
3. for $i \leftarrow 3$ to n
4. do if $p_i \in D_{i-1}$
5. then $D_i \leftarrow D_{i-1}$
6. else $D_i \leftarrow$ MiniDiscWithPoint $(\{p_1 , \ldots , p_{i-1} \}, p_i)$
7. return D_n

MiniDiscWithPoint(*P,q*)

Input: A set P of n points in the plane, and a point q s.t. there exists an enclosing disc for P with q on its boundary

Output: The smallest enclosing disc for P with q on its boundary

1. Compute a random permutation $p_1, \ldots, p_n$ of P
2. Let D_1 be smallest enclosing disc w/ q & p_1 on its boundary
3. for $j \leftarrow 2$ to n
4. do if $p_j \in D_{j-1}$
5. then $D_j \leftarrow D_{j-1}$
6. else $D_j \leftarrow$ MiniDiscWith2Point($\{p_1, \ldots, p_{j-1}\}, p_j, q$)
7. return D_n

MiniDiscWith2DPoint(P,q_1,q_2)

Input: A set P of n points in the plane, and two points q_1 & q_2 s.t. there exists an enclosing disc for P with q_1 & q_2 on boundary

Output: The smallest enclosing disc for P with q_1 & q_2 on bndary

1. Let D_0 be smallest enclosing disc w/ q_1 & q_2 on boundary
2. for $k \leftarrow 1$ to n
3. do if $p_k \in D_{k-1}$
4. then $D_k \leftarrow D_{k-1}$
5. else $D_k \leftarrow$ the disc w/ q_1, q_2 and p_k on its boundary
6. return D_n

Algorithm Analysis

- The running time of *MiniDiscWithPoint* is $O(n)$ without call to *MiniDiscWith2Points*. The probability of making such a call is $2/i$. The total expect run time of *MiniDiscWithPoint* is

$$O(n) + \sum_{2 \le i \le n} O(i) \cdot (2/i) = O(n)$$

- Applying the same argument once more. We have: the smallest enclosing disc for a set of n points in the plane can be computed in $O(n)$ expected time using worst-case $O(n)$ storage

Conclusion

- Castability problem
  - Compute remove direction
    - Half-plane intersection problem
    - Convex intersection problem
    - Linear programming
    - Randomized linear programming
- Collision detection of two convex polyhedra
- Smallest enclosing disc of points

Next time

- Range search
  - Database query