



# Randomized algorithms 5

## Linear programming

CS240

Spring 2022

*Rui Fan*

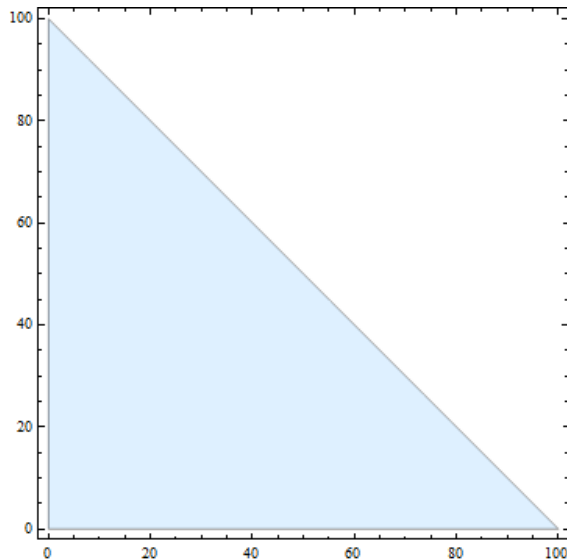


# Linear programming

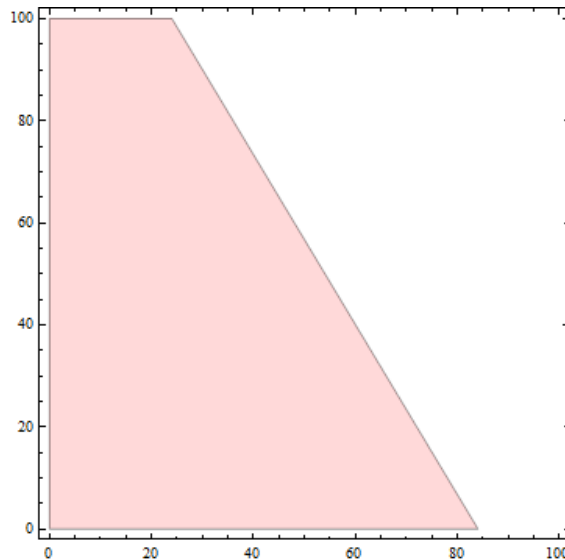
- A farmer has the following problem.
  - He has 100 acres of land, on which he can plant wheat or barley or both.
  - He has 420 kg of fertilizer, and 160 kg of pesticide.
  - Each acre of barley requires 5 kg of fertilizer and 1 kg of pesticide.
  - Each acre of wheat requires 3 kg of fertilizer and 2 kg of pesticide.
  - Wheat sells for \$3 per acre, barley sells for \$4 per acre.
    - Actually, he could probably make \$300 for wheat and \$400 for barley. Choose \$3 and \$4 for simplicity.
- How many acres of wheat and barley should the farmer plant his field to maximize his income?
- Let  $w$ ,  $b$  be acres of wheat and barley farmer plants.
- He wants to maximize  $3w+4b$ , subject to the land, pesticide and fertilizer constraints.

# Linear programming

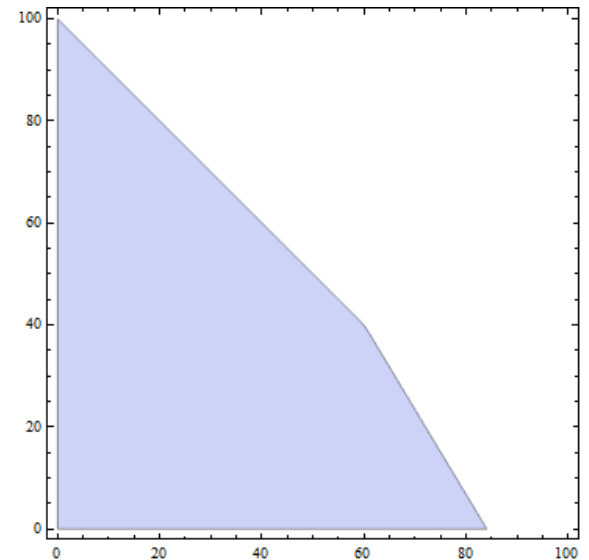
- **maximize  $3w+4b$  subject to**
  - $w+b \leq 100$  (**land**)
  - $3w+5b \leq 420$  (**fertilizer**)
  - $2w+b \leq 160$  (**pesticide**)



land constraint



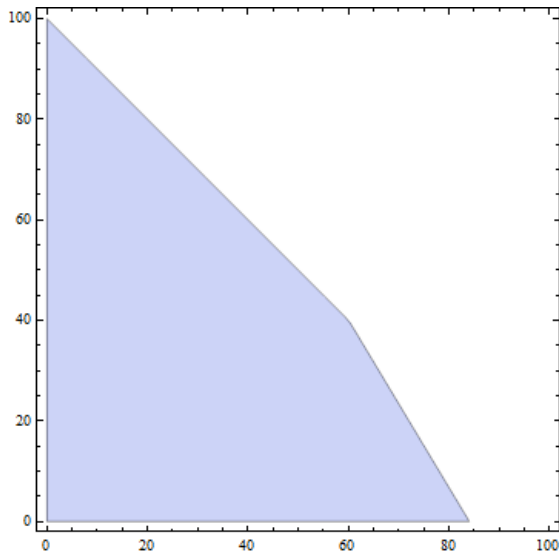
fertilizer constraint



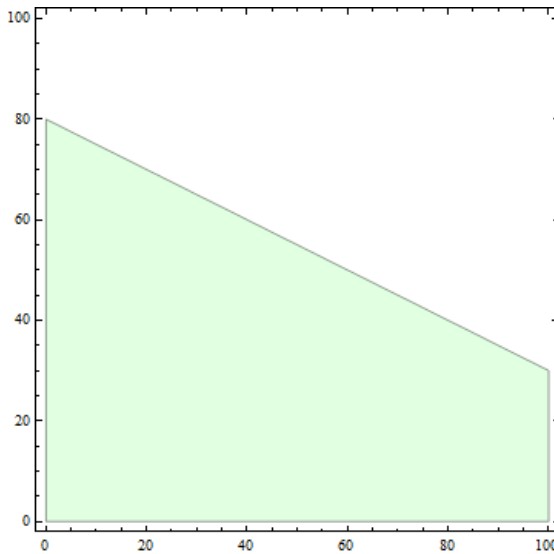
land + fertilizer  
constraints

# Linear programming

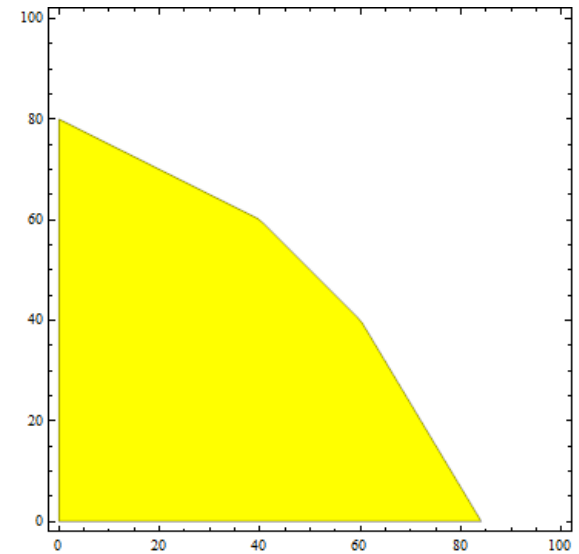
- **maximize  $3w+4b$  subject to**  
 $w+b \leq 100$  (**land**)  
 $3w+5b \leq 420$  (**fertilizer**)  
 $2w+b \leq 160$  (**pesticide**)



land + fertilizer  
constraints



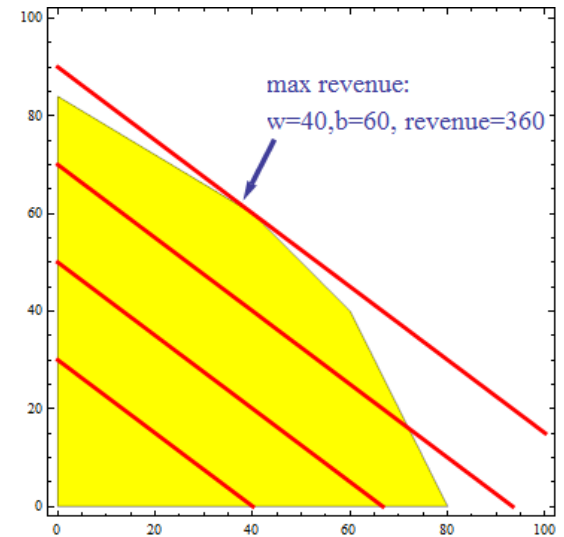
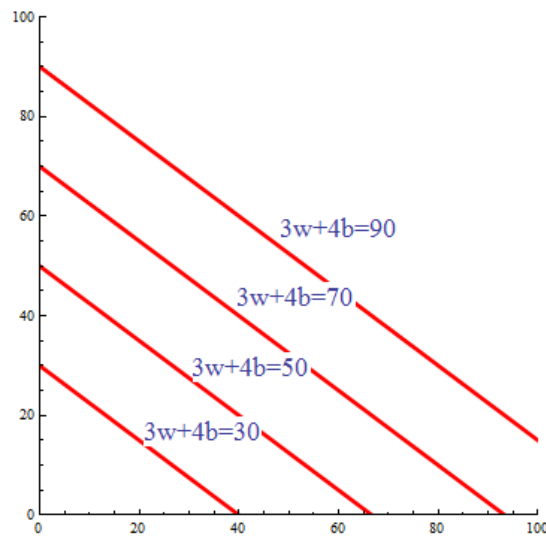
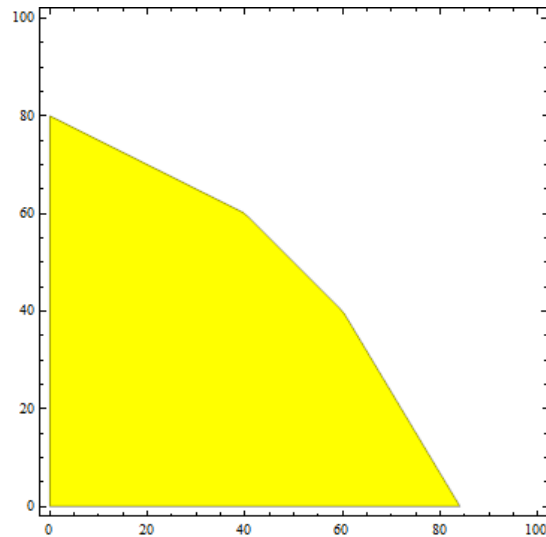
pesticide constraints



all three constraints

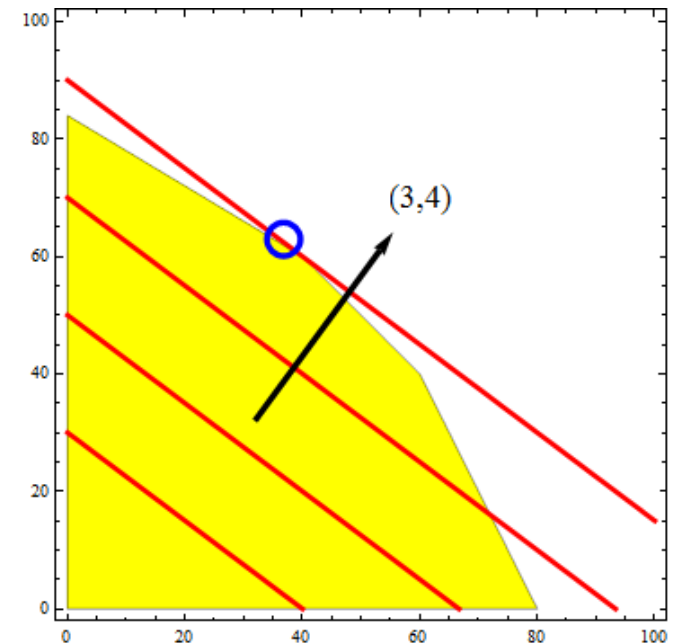
# Linear programming

- **maximize  $3w+4b$  subject to**  
 $w+b \leq 100$  (land)  
 $3w+5b \leq 420$  (fertilizer)  
 $2w+b \leq 160$  (pesticide)



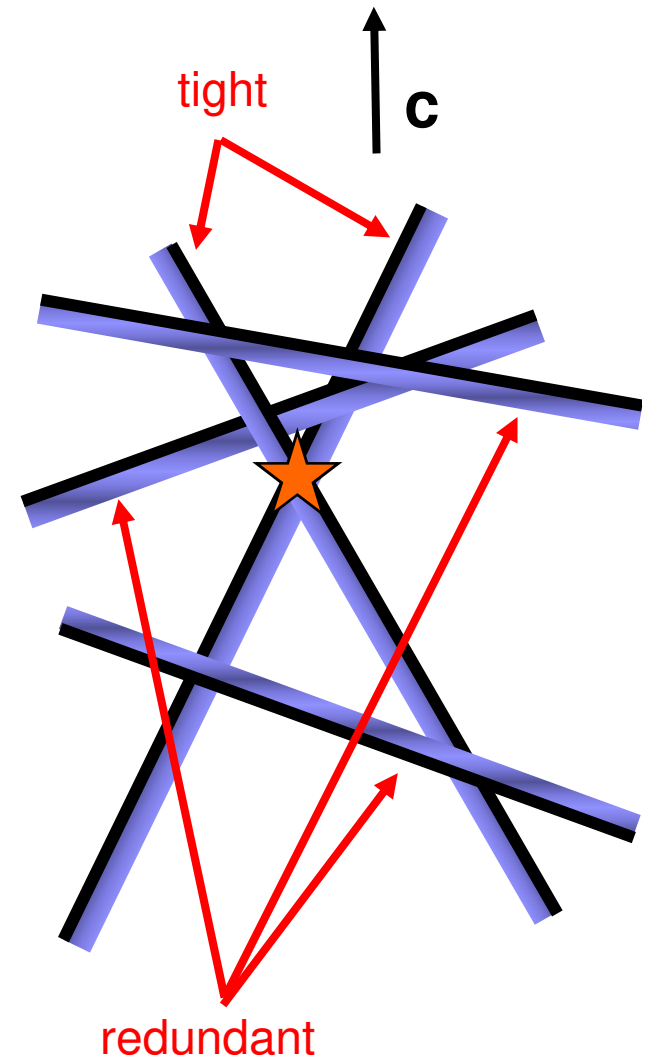
# Linear programming

- The feasible region is the area corresponding in which all the constraints are satisfied.
- **Key Fact** The optimum lies at an extreme point (corner).
- Find optimum by taking a line perpendicular to the direction pointed by the objective function, and shifting the line till when it will stop touching the feasible region.
- The optimum lies at the intersection of two constraints.
  - Call these the basis of the optimum.
  - For simplicity, assume constraints are general position, i.e. no 3 intersect at a point.



# Randomized LP in 2D

- Since the optimum is defined by two constraints, the other constraints are redundant!
- A constraint is tight if it the optimum lies on its defining line.
- Let  $H$  be set of  $n$  constraints. If pick random constraint, there's only  $2/n$  probability it's tight.
- If constraint's not tight, we can discard it without changing optimum.
- How do we tell if it's tight?
  - For any constraint set  $G$ , let  $B(G)$  denote optimum.
- $h \in H$  is redundant iff  $B(H) = B(H - \{h\})$ .
  - I.e. the optimum is the same with or without  $h$ . So opt doesn't lie on  $h$ .





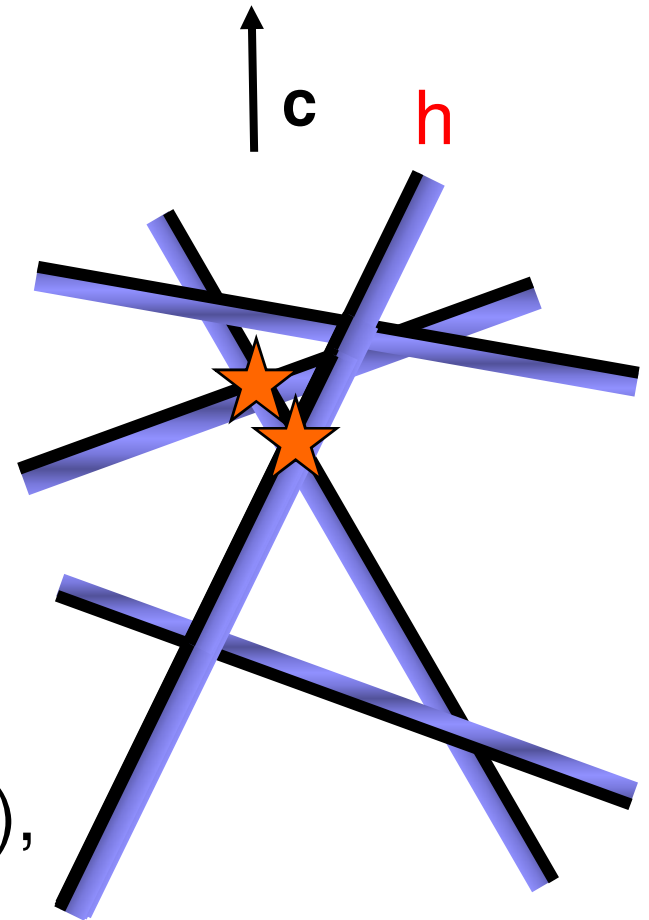
## 2D LP algorithm

- ❖ If  $|H|=2$ , output intersection of the 2 halfplanes.
- ❖ Pick random constraint  $h \in H$ .
- ❖ Recursively find  $\text{opt} = B(H - \{h\})$ .
- ❖ If  $\text{opt}$  doesn't violate  $h$ , output  $\text{opt}$ .
  - ❖  $\text{opt}$  violates  $h$  if  $\text{opt}$  lies outside  $h$ .
- ❖ Else project  $H - \{h\}$  onto  $h$ 's boundary to obtain a 1D LP.
- ❖ Output the  $\text{opt}$  of the 1D LP.



# Projection

- Given constraint  $h$ , let  $\partial(h)$  be its boundary, i.e. the line defining  $h$ .
- Suppose  $B(H-\{h\})$  violates  $h$ .
  - Then  $B(H)$  must lie on the boundary of  $h$ .
- Project a halfplane onto  $\partial(h)$ , reducing it to a line segment bounded on one or two sides.
- After projecting all  $H-\{h\}$  onto  $\partial(h)$ , we're left with a segment representing feasible region to 1D LP.
- Optimizing this is easy. The opt is one of the endpoints.



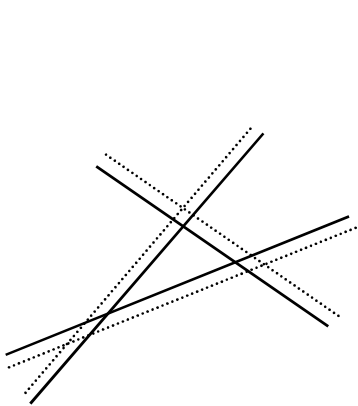


# Analysis

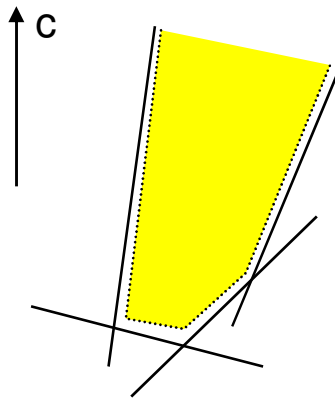
- Let  $T(n)$  be expected time to solve 2D LP with  $n$  constraints.
- $T(n) \leq T(n-1) + O(1) + 2/n(O(n) + O(1))$ .
- $T(n-1)$  time recursively find  $\text{opt} = B(H - \{h\})$ .
- First  $O(1)$  is time to check whether  $\text{opt}$  violates  $h$ .
- There's  $2/n$  probability  $\text{opt}$  violates  $h$ , in which case we project all constraints onto  $\partial(h)$ .
  - $O(n)$  to project  $H - \{h\}$  onto  $\partial(h)$ .
  - Final  $O(1)$  to solve 1D LP.
- $T(n)$  solves to  $O(n)$ .
  - So we can solve 2D LP with  $n$  constraints in expected linear time.

# Corner cases

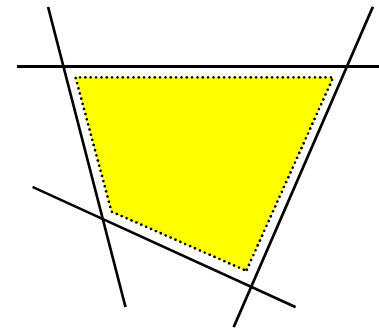
- For simplicity, we ignored several corner cases.
  - Infeasible means no points satisfy all the constraints.
    - Ex Constraints  $x_1 > 1$  and  $x_1 < 0$ .
  - Unbounded means the optimum is infinite.
    - Ex Maximize  $x_2$  s.t.  $x_1 + x_2 > 0$ .
  - Non-unique optimum means an infinite number of points maximize the objective.
    - Ex Maximize  $x_2$  s.t.  $x_2 \leq 0$ .
- Preprocess input to check for corner cases.



Infeasible



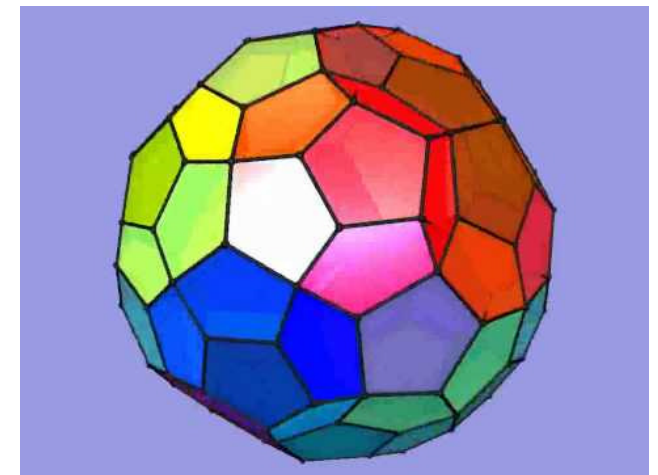
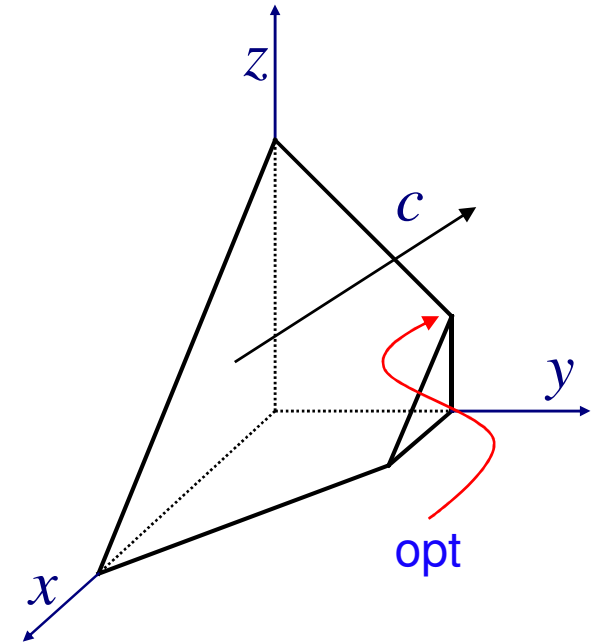
Unbounded



Non-unique optimum

# Higher dimensions

- In  $d > 2$  dimensions, lines become planes and each constraint corresponds to the space to one side of a plane, called a halfspace.
- The intersection of the halfspaces defines the feasible region.
  - This is a convex region called a polytope.
- Each extreme point (corner) of the polytope is the intersection of  $d$  halfspaces.
- The objective function defines a direction. Take a plane perpendicular to this direction and shift it till it stops touching feasible region.
- Hence optimum again lies at intersection of  $d$  halfspaces.
- Polytopes can be very complicated.





## d-Dimensional LP algorithm

- ❖ If  $|H|=d$ , output their intersection.
- ❖ Pick random constraint  $h \in H$ .
- ❖ Recursively find  $\text{opt} = B(H - \{h\})$ .
- ❖ If  $\text{opt}$  doesn't violate  $h$ , output  $\text{opt}$ .
- ❖ Else project  $H - \{h\}$  onto  $h$ 's boundary to obtain a  $d-1$  dimensional LP.
- ❖ Recursively solve the  $d-1$  dim LP.



# Analysis

- Let  $T(n,d)$  be expected time to solve  $d$ -dim LP with  $n$  constraints.
- $T(n,d) \leq T(n-1,d) + O(d) + d/n(O(dn) + T(n-1,d-1))$ .
- $T(n-1,d)$  time recursively find  $\text{opt} = B(H - \{h\})$ .
- $O(d)$  time to check whether  $\text{opt}$  violates  $h$ .
- There's  $d/n$  probability  $\text{opt}$  violates  $h$ .
  - Because  $\text{opt}$  is defined by  $d$  of the  $n$  halfspaces.
  - In this case we project all constraints onto  $\partial(h)$ .
  - $O(dn)$  to project  $H - \{h\}$  onto  $\partial(h)$ .
  - We obtain a  $d-1$  dim LP with  $n-1$  constraints.
  - $T(n-1,d-1)$  time to solve this.
- $T(n,d)$  solves to  $O(d! n)$ 
  - Linear in number of constraints.
  - Exponential in dimensions.

# Matrix formulation

- **maximize**  $3w+4b$  **subject to**

$$w+b \leq 100$$

$$3w+5b \leq 420$$

$$2w+b \leq 160$$

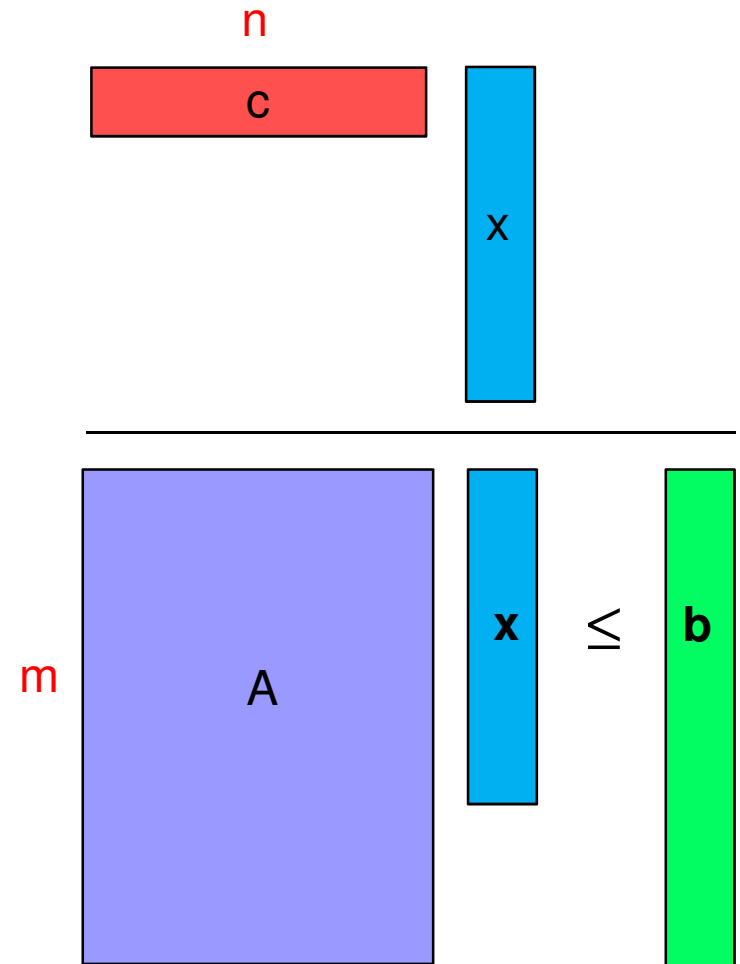
- **maximize**  $[3,4] \cdot \begin{bmatrix} w \\ b \end{bmatrix}$  **s.t.**

$$\begin{bmatrix} 1 & 1 \\ 3 & 5 \\ 2 & 1 \end{bmatrix} \cdot \begin{bmatrix} w \\ b \end{bmatrix} \leq \begin{bmatrix} 100 \\ 420 \\ 160 \end{bmatrix}$$

- Let  $x \in \Re^{n \times 1}, c \in \Re^{1 \times n}, A \in \Re^{m \times n}, b \in \Re^{m \times 1}$ .

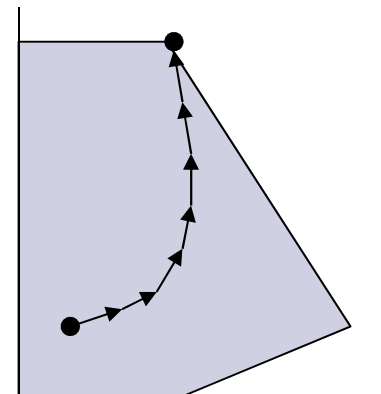
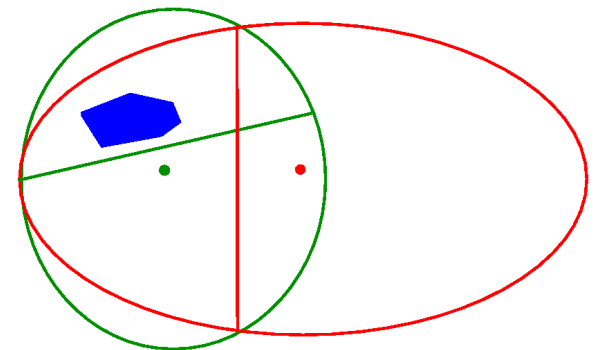
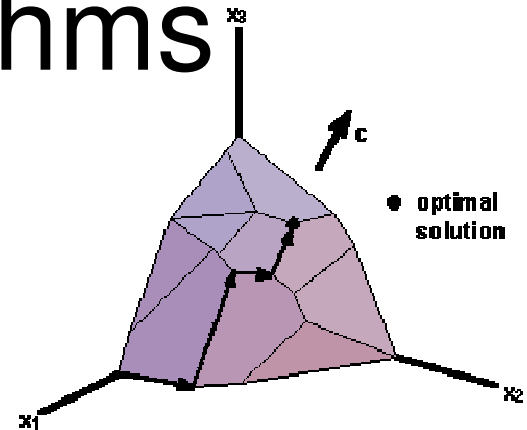
$$\text{maximize } c \cdot x \text{ s.t.}$$

$$A \cdot x \leq b$$



# Deterministic LP algorithms

- Simplex method (Dantzig 1947)
  - Optimum lies at the farthest corner in direction of  $c$ .
  - Walk along boundary of polytope in directions that improve  $c$ .
  - Number of steps can be exponential in worst case.
  - But works fast in practice.
- Ellipsoid method (Khachiyan 1979)
  - First polynomial solution.
  - Interesting mostly in theory.
- Interior point method (Karmarkar 1984)
  - Practical polynomial method.





# Applications of LP

- Linear programming is one of the most important and widely used algorithms in the world.
- NY Times, Nov. 27, 1979

## *A Soviet Discovery Rocks World of Mathematics*

By MALCOLM W. BROWNE

A surprise discovery by an obscure Soviet mathematician has rocked the world of mathematics and computer analysis, and experts have begun exploring its practical applications.

Mathematicians describe the discovery by L.G. Khachian as a method by which computers can find guaranteed solutions to a class of very difficult problems that have hitherto been tackled on a kind of hit-or-miss basis.

Apart from its profound theoretical interest, the discovery may be applicable

in weather prediction, complicated industrial processes, petroleum refining, the scheduling of workers at large factories, secret codes and many other things.

"I have been deluged with calls from virtually every department of government for an interpretation of the significance of this," a leading expert on computer methods, Dr. George B. Dantzig of Stanford University, said in an interview.

The solution of mathematical problems by computer must be broken down into a series of steps. One class of problem sometimes involves so many steps that it

could take billions of years to compute.

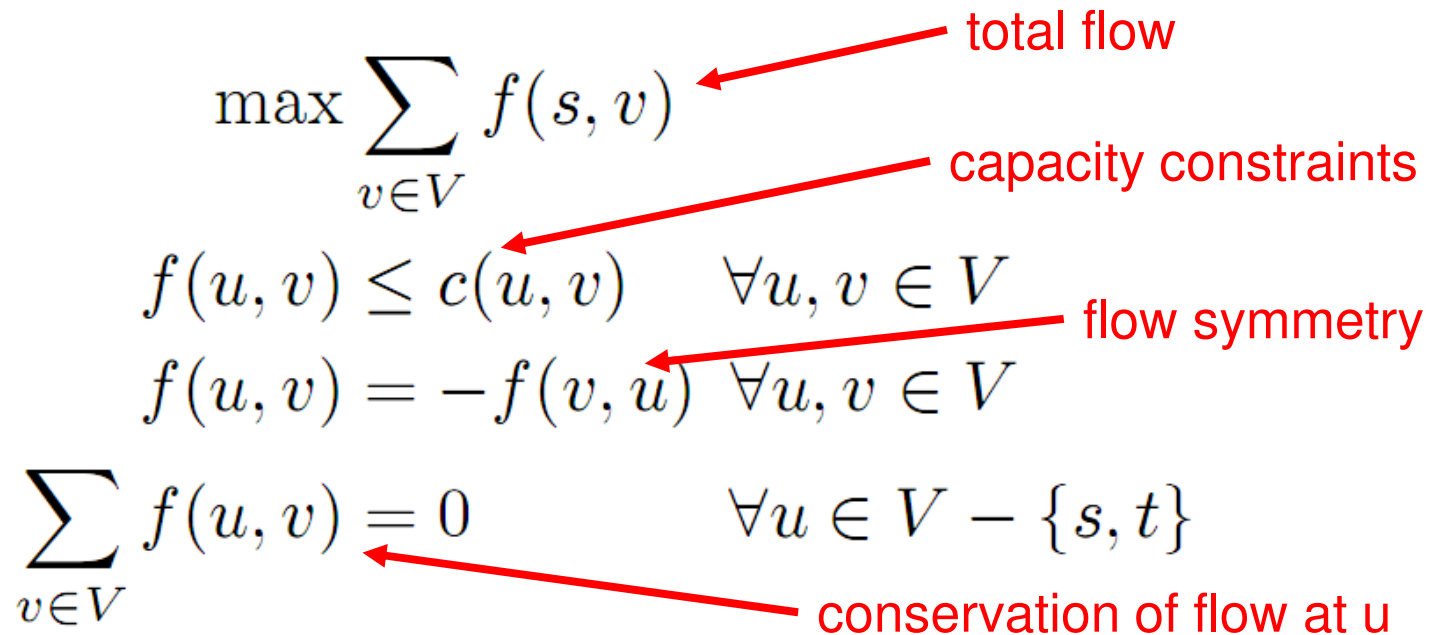
The Russian discovery offers a way by which the number of steps in a solution can be dramatically reduced. It also offers the mathematician a way of learning quickly whether a problem has a solution or not, without having to complete the entire immense computation that may be required.

According to the American journal Sci-

**Continued on Page A20, Column 3**

ONLY \$10.00 A MONTH!!! 24 Hr. Phone Answering Service. Totally New Concept!!! Incredible!!! 279-3870—ADVT.

# Applications of LP: Network flow



total flow

$$\max \sum_{v \in V} f(s, v)$$

capacity constraints

$$f(u, v) \leq c(u, v) \quad \forall u, v \in V$$

flow symmetry

$$f(u, v) = -f(v, u) \quad \forall u, v \in V$$

conservation of flow at u

$$\sum_{v \in V} f(u, v) = 0 \quad \forall u \in V - \{s, t\}$$

- However, it's more efficient to solve max flow using e.g. Dinic's algorithm than more general LP algorithms.



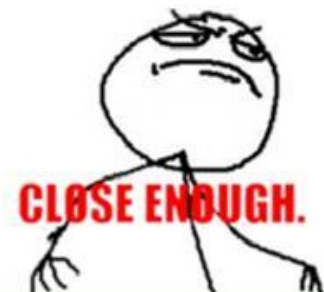
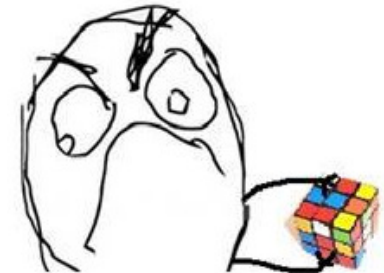
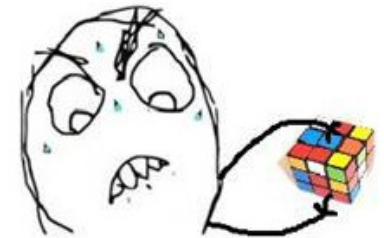
# Approximation algorithms 1

## Set cover

# Approximation algorithms

- Up to now, most of our algorithms have been exact. I.e. they find an optimal solution.
- But there are many problems for which we don't know how to find an optimal solution.
  - A key example is NP-complete problems. We don't know efficient algorithms for any NPC problem.
- Many such problems are important in practice. What do we do?
- If we can't get find the best answer, let's try for good enough.
- Approximation algorithms find an approximately optimal answer.

\*le me struggling with rubic cube



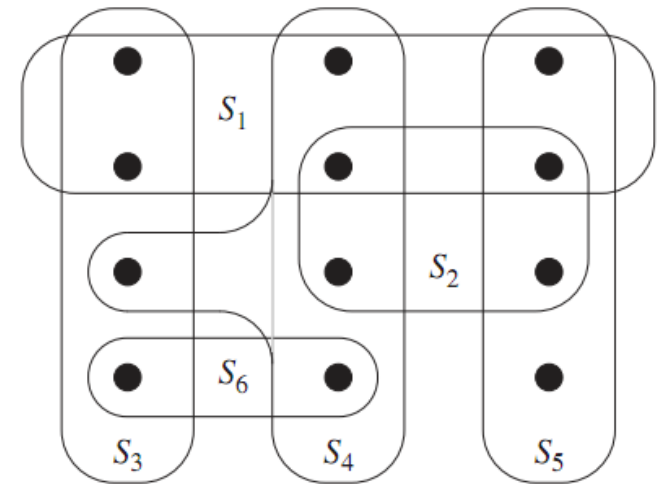


# Approximation ratio

- Let  $X$  be a maximization problem. Let  $A$  be an algorithm for  $X$ .
- Let  $\alpha > 1$  be a constant.
- $A$  is an  $\alpha$ -approximation algorithm for  $X$  if  $A$  always returns an answer that's at least  $1/\alpha$  times the optimal.
  - Ex If  $X$  is max-flow,  $A$  is a 2-approx algorithm if it always returns a flow that's at least  $1/2$  the optimal.
  - The closer  $\alpha$  is to 1, the better the approximation.
- If  $X$  is a minimization problem,  $A$  is an  $\alpha$ -approximation algorithm for  $X$  if it always returns an answer that's at most  $\alpha$  times larger than the optimal.
  - Ex If  $X$  is min-cut,  $A$  is a 2-approx algorithm if it always returns a cut that's at most 2 times the size of the optimal.
  - Again, the closer  $\alpha$  is to 1, the better the approximation.

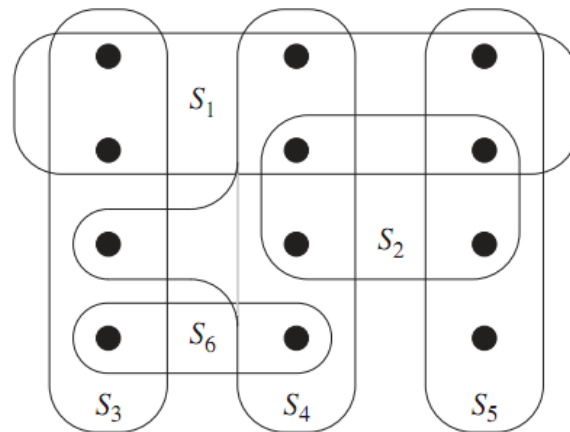
# Coverings

- Suppose there's a set of teachers, and each can teach a certain set of classes.
  - Let  $S_i$  be the set of classes teach  $i$  can teach.
- The entire set of classes is  $X$ .
- We want to pick the minimum set of teachers to teach all the classes.
  - Let  $T$  be set of teachers we pick.
  - We want  $\bigcup_{i \in T} S_i = X$ , and  $T$  to be the smallest possible.



# Set covering

- **Input** A collection  $F$  of sets. Each set has a cost. The union of all the sets is  $X$ .
- **Output** A subset  $G$  of  $F$ , whose union is  $X$ .
- **Goal** Minimize the total cost of the sets in  $G$ .



*If all sets have same cost,  $S_3$ ,  $S_4$  and  $S_5$  is a min cost set cover of  $X$ .*

- Minimum cost set cover is NP-complete.
- We'll see a  $\ln(n)$ -approximation algorithm, where  $n=|X|$ .



# A greedy approximation alg

- A natural greedy heuristic is to choose sets which cover points most cheaply.
  - For each set, let  $c$  be its cost, and  $m$  be the number of points it covers.
  - We want to use the set with the smallest  $c/m$  value, because this is the cheapest way to cover some new points.
- After we pick this set, remove all the points it covers. Then we consider the per unit cost of the remaining sets and again choose the cheapest.



# A greedy approximation alg

- $F$  is the entire collection of sets. The union of these sets is  $X$ .
- Each set  $S$  in  $F$  has a cost  $\text{cost}(S)$ .
- $U$  is the set of elements of  $X$  we haven't covered yet.
- $C$  is the set cover we eventually output.

- $U = X$

- $C = \emptyset$

- while  $U \neq \emptyset$

- choose  $S \in F - C$  with min  $|\text{cost}(S)| / |S \cap U|$

- $C = C \cup \{S\}$

- $U = U - S$

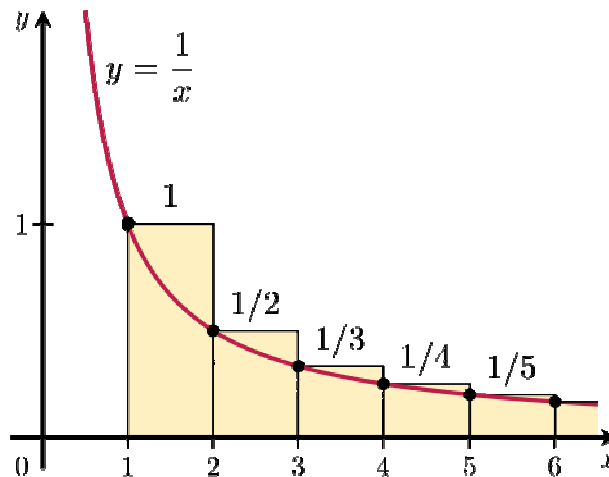
- output  $C$

Per unit cost to cover  
new elements.



# Proof of correctness

- We always output a set cover, because the while loop continues till  $X$  is covered.
- We'll prove the approximation ratio is at most  $1 + 1/2 + 1/3 + \dots + 1/n \approx \ln(n)$ .
  - If the min cost of a set cover is  $V$ , our set cover costs at most  $\ln(n) * V$ .
- The basic plan is to bound the cost of the set cover the algorithm outputs using the “average cost” per element.





# Proof of correctness

- Order the sets in  $C$  by when they're added to  $C$ , earliest set first.
  - Let the order be  $S_1, S_2, \dots, S_m$ .
- Cost of the set cover is  $L = \sum_i \text{cost}(S_i)$ .
- Order the elements in  $X$  by when they're added, earliest element first.
  - Let the order be  $e_1, e_2, \dots, e_n$ .
  - So, the first few  $e$ 's are added by  $S_1$ , the next few added by  $S_2$ , etc.
  - Every element in  $X$  is in the list, because  $C$  covers  $X$ .

# Proof of correctness

- Let  $n_i$  be the number of new elements  $S_i$  covers.
  - So,  $n_i$  is the number of elements in  $S_i$ , but not in  $S_1, \dots, S_{i-1}$ .
- Divide the cost of  $S_i$  evenly among the new elements it covers.
  - If  $e$  is newly covered by  $S_i$ , then  $\text{cost}(e) = \text{cost}(S_i)/n_i$ .
- $\sum_k \text{cost}(e_k) = \sum_i n_i * \text{cost}(S_i)/n_i = \sum_i \text{cost}(S_i) = L$ .
  - Every element is covered by some  $S_i$ , and  $S_i$  covers  $n_i$  new elements.
- We'll prove  $\text{cost}(e_k) \leq \text{OPT}/(n-k+1)$ , for any  $k$ .
- Suppose this is true, then
$$L = \sum_k \text{cost}(e_k) \leq \sum_k \text{OPT}/(n-k+1) \approx \ln(n) * \text{OPT}$$

# The per element cost

- Let's focus on some element  $e_k$ , and let  $S_j$  be the set which covers  $e_k$  for the first time.
- Let  $C_1, \dots, C_m$  be the sets in an optimal cover, each of which covers some elements of  $U = \{e_k, e_{k+1}, e_{k+2}, \dots, e_n\}$ .
  - Let  $n'_1, \dots, n'_m$  be the number of elements of  $U$  which  $C_1, \dots, C_m$  cover.
- **Obs 1**  $\sum_i n'_i \geq n - k + 1$ .
  - Because  $C_1, \dots, C_m$  cover  $U$ .
- **Obs 2**  $\sum_i \text{cost}(C_i) \leq \text{OPT}$ .
  - Because  $C_1, \dots, C_m$  are a subset of an optimal cover, which has cost  $\text{OPT}$ .

# The per element cost

- **Obs 3** None of  $C_1, \dots, C_m$  are among  $S_1, \dots, S_{j-1}$ .
  - If some  $C_i$  is among  $S_1, \dots, S_{j-1}$ , then since  $C_i$  covers some  $e$  in  $U$ ,  $e$  would be covered by  $\{S_1, \dots, S_{j-1}\}$ . So,  $e$  would be among the first  $k-1$  elements covered. Contradiction.
- **Obs 4** There exists some  $C_i$  among  $C_1, \dots, C_m$  with  $\text{cost}(C_i)/n'_i \leq \text{OPT}/(n-k+1)$ .
  - If every  $C_i$  in  $C_1, \dots, C_m$  has  $\text{cost}(C_i)/n'_i > \text{OPT}/(n-k+1)$ , then
$$\begin{aligned} \text{OPT} &\geq \sum_i \text{cost}(C_i) = \sum_i n'_i * \text{cost}(C_i)/n'_i > \\ \sum_i n'_i * \text{OPT}/(n-k+1) &\geq \text{OPT}/(n-k+1) \sum_i n'_i \geq \\ \text{OPT}/(n-k+1) * (n-k+1) &= \text{OPT}. \end{aligned}$$
Contradiction.

# Proof of approximation ratio

- **Lemma**  $\text{cost}(S_j)/n_j \leq \text{OPT}/(n-k+1)$ .
- **Proof** When choosing  $S_j$ , the only sets the algorithm is not allowed to choose are  $S_1, \dots, S_{j-1}$ .
  - By obs 3,  $C_1, \dots, C_m$  aren't in  $S_1, \dots, S_{j-1}$ .
  - By obs 4, there's some  $C_i$  in  $C_1, \dots, C_m$ , with  $\text{cost}(C_i)/n'_i \leq \text{OPT}/(n-k+1)$ .
  - $S_j$  was chosen so that  $\text{cost}(S_j)/n_j$  is min among all sets not in  $S_1, \dots, S_{j-1}$ .
  - So  $\text{cost}(S_j)/n_j \leq \text{cost}(C_i)/n'_i \leq \text{OPT}/(n-k+1)$ .
- Since  $\text{cost}(S_j)/n_j = \text{cost}(e_k)$ , we have  $\text{cost}(e_k) \leq \text{OPT}/(n-k+1)$ .
- The approx ratio follows because
$$L = \sum_k \text{cost}(e_k) = \sum_k \text{OPT}/(n-k+1) \approx \ln(n) * \text{OPT}$$