Solvers for Mixed Integer Programming

Relaxation: A general optimization technique

- Want:
  - $x^* = \text{argmin}_x f(x)$ subject to $x \in S$
  - S is the feasible set
- Start by getting:
  - $x_1 = \text{argmin}_x f(x)$ subject to $x \in T$
  - where $S \subseteq T$
    - T is a larger feasible set, obtained by dropping some constraints
    - Makes problem easier if we have a large # of constraints or difficult ones
  - If we're lucky, it happens that $x_1 \in S$
    - Then $x^* = x_1$, since
      - x_1 is a feasible solution to the original problem
      - no feasible solution better than x_1 (no better $x \in S$ since none anywhere $\in T$)
  - Else, add some constraints back (to shrink T) and try again, getting x_2
    - $x_1, x_2, x_3, \ldots \rightarrow x^*$ as T closes in on S

Relaxation: A general optimization technique

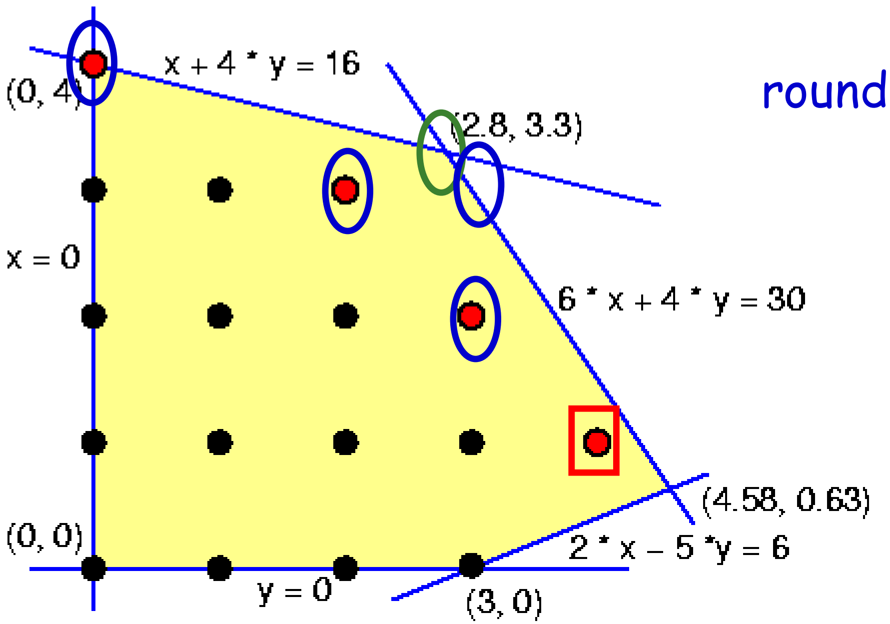
- Want:
  - $x^* = \text{argmin } f(x)$ subject to $x \in S$
  - S is the feasible set
- Start by getting:
  - $x_1 = \text{argmin } f(x)$ subject to $x \in T$
  - where $S \subseteq T$
    - T is a larger feasible set, obtained by dropping some constraints
    - Makes problem easier if we have a large # of constraints or difficult ones

Integrality constraints: if we drop all of these, we can just use simplex. "LP relaxation of the ILP problem."

  - Else, add some constraints back (to shrink T) and try again

But how can we add integrality constraints back? (simplex relies on having dropped them all)

Rounding doesn't work



Function to maximize: f(x, y) = 6 * x + 5 * y

Optimum LP solution (x, y) = (2.8, 3.3)

Pareto optima: (0, 4), (2, 3), (3, 2), (4, 1)

Optimum ILP solution (x, y) = (4, 1)

round to nearest int (3,3)? No, infeasible.
round to nearest feasible int (2,3) or (3,2)?
No, suboptimal.
round to nearest integer <u>vertex</u> (0,4)?
No, suboptimal.

Really do have to add the integrality constraints back somehow, and solve a new optimization problem.

Cutting planes: add new linear constraints

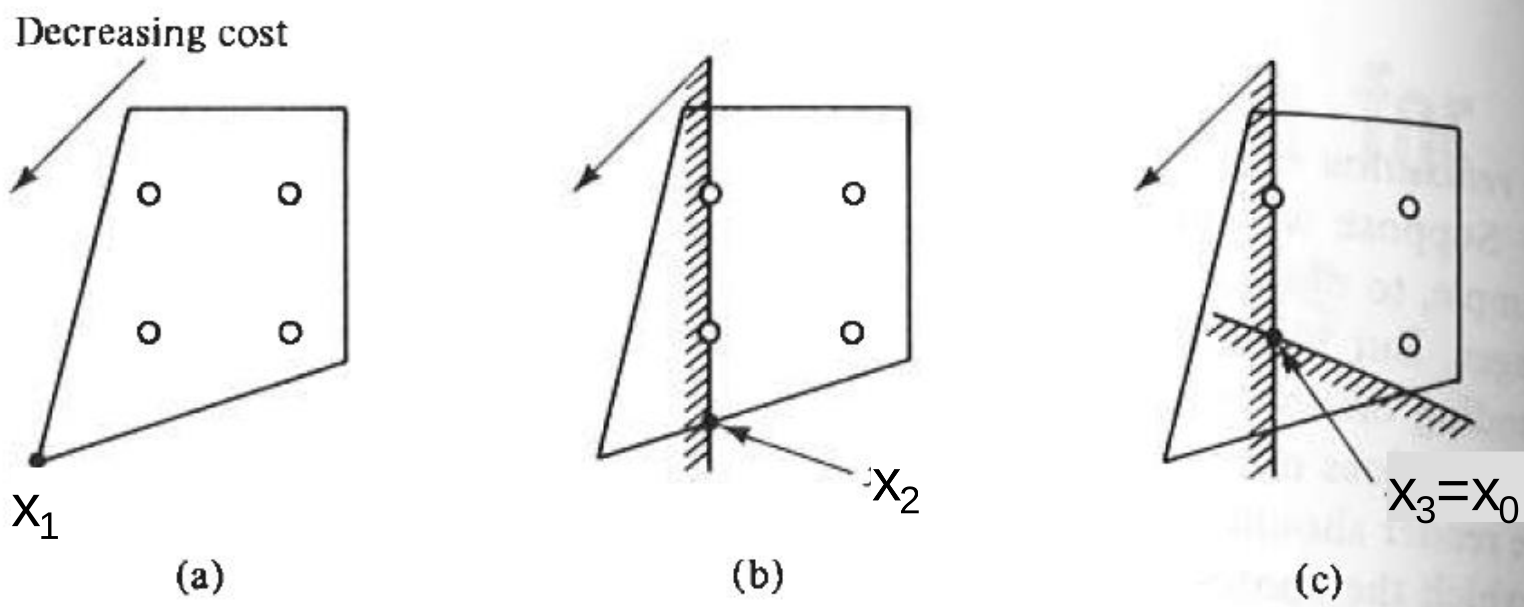
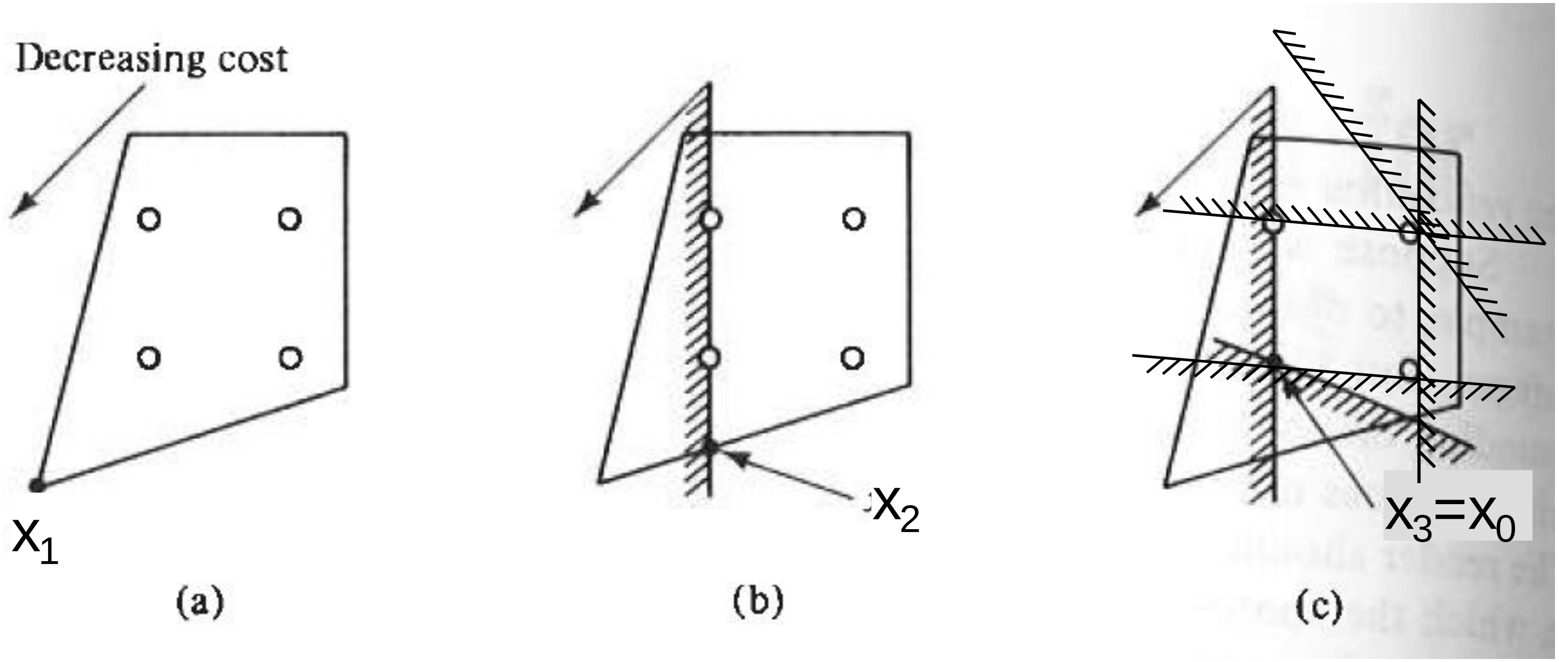


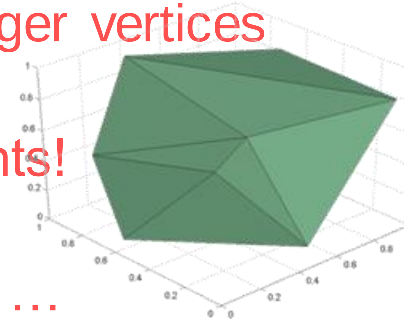
Figure 14–2 Illustration of a cutting-plane algorithm. (a) The continuous optimum x_1. (b) The new x_2 after one cut. (c) The solution of the original ILP after two cuts.

- New linear constraints can be handled by simplex algorithm
- But will collectively rule out non-integer solutions

figure adapted from Papadimitriou & Steiglitz

Add new linear constraints: Cutting planes





- Can ultimately trim back to a new polytope with only integer vertices
  - This is the “convex hull” of the feasible set of the ILP
- Since it’s a polytope, it can be defined by linear constraints!
  - These can replace the integrality constraints
- Unfortunately, there may be exponentially many of them …
  - But hopefully we’ll only have to add a few (thanks to relaxation)

figure adapted from Papadimitriou & Steiglitz

Example

$$x_1 + 4x_2 + x_3 \geqslant 10$$
$$4x_1 + 2x_2 + 2x_3 \geqslant 13$$
$$x_1 + x_2 - x_3 \geqslant 0$$
$$x_1, x_2, x_3 \geqslant 0 \quad \text{and integer.}$$

$\Rightarrow$

$$x_1 + 4x_2 + x_3 \geqslant 10$$
$$x_1 + 3x_2 + x_3 \geqslant 9$$
$$2x_1 + 4x_2 + x_3 \geqslant 13$$
$$x_1 + x_2 + x_3 \geqslant 5$$
$$2x_1 + x_2 + x_3 \geqslant 7$$
$$x_1 + 2x_2 \geqslant 5$$
$$2x_1 + x_2 \geqslant 4$$
$$x_1 + x_2 - x_3 \geqslant 0$$
$$x_1, x_2, x_3 \geqslant 0.$$

No integrality constraints!
But optimal solution is the same.

- How can we find these new constraints??

example from H. P. Williams

Chvátal cuts

$$x_1 + 4x_2 + x_3 \geqslant 10$$
$$4x_1 + 2x_2 + 2x_3 \geqslant 13$$
$$x_1 + x_2 - x_3 \geqslant 0$$
$$x_1, x_2, x_3 \geqslant 0 \quad \text{and integer.}$$

even coeffs: divide by 2

$$2x_1 + x_2 + x_3 \geqslant 6\tfrac{1}{2}$$

Must be integer because of integrality constraints on x_1,x_2,x_3

round

$$2x_1 + x_2 + x_3 \geqslant 7$$
$$+\ 5 * (x_1 + 4x_2 + x_3 \geqslant 10)$$
$$+\ (x_3 \geqslant 0)$$

$$7x_1 + 21x_2 + 7x_3 \geq 57$$

divide by 7 and round

$$x_1 + 3x_2 + x_3 \geqslant 9$$

- Add integer multiples of constraints, divide through, and round using integrality
  - This generates a new (or old) constraint
- Repeat till no new constraints can be generated
  - Generates the convex hull of the ILP!
  - But it's impractical

example from H. P. Williams

Gomory cuts

- Chvátal cuts:
  - Can generate the convex hull of the ILP!
  - But that's impractical
  - And unnecessary (since we just need to find optimum, not whole convex hull)

- Gomory cuts:
  - Only try to cut off current relaxed optimum that was found by simplex
  - "Gomory cut" derives such a cut from the current solution of simplex

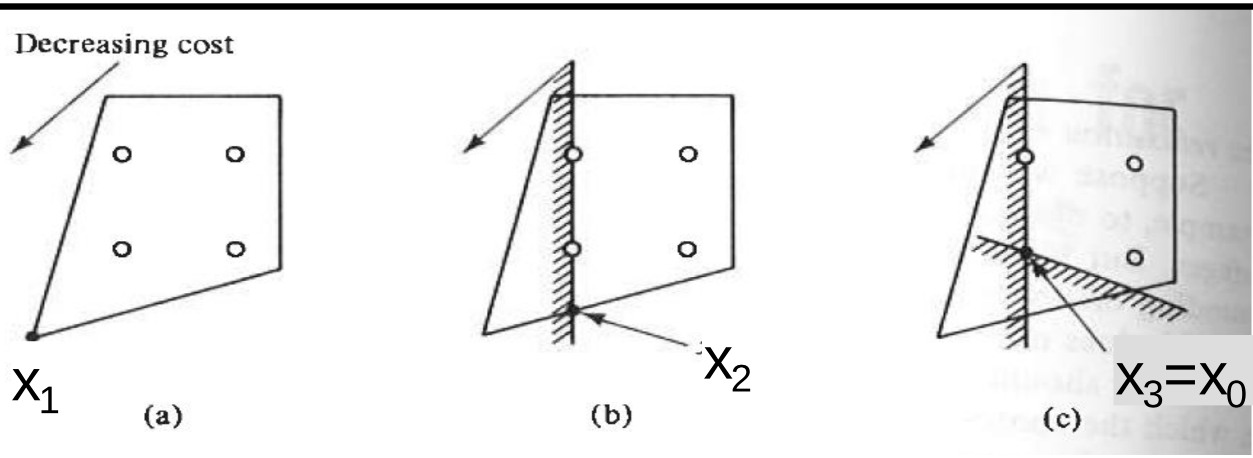


figure adapted from Papadimitriou & Steiglitz

Branch and bound: Disjunctive cutting planes!

Minimise $2x_1 + 7x_2 + 2x_3$

$$x_1 + 4x_2 + x_3 \geqslant 10$$

$$4x_1 + 2x_2 + 2x_3 \geqslant 13$$

$$x_1 + x_2 - x_3 \geqslant 0$$

$x_1, x_2, x_3 \geqslant 0$ and integer.

For each leaf, why is it okay to stop there?

When does solving the relaxed problem ensure integral X2?

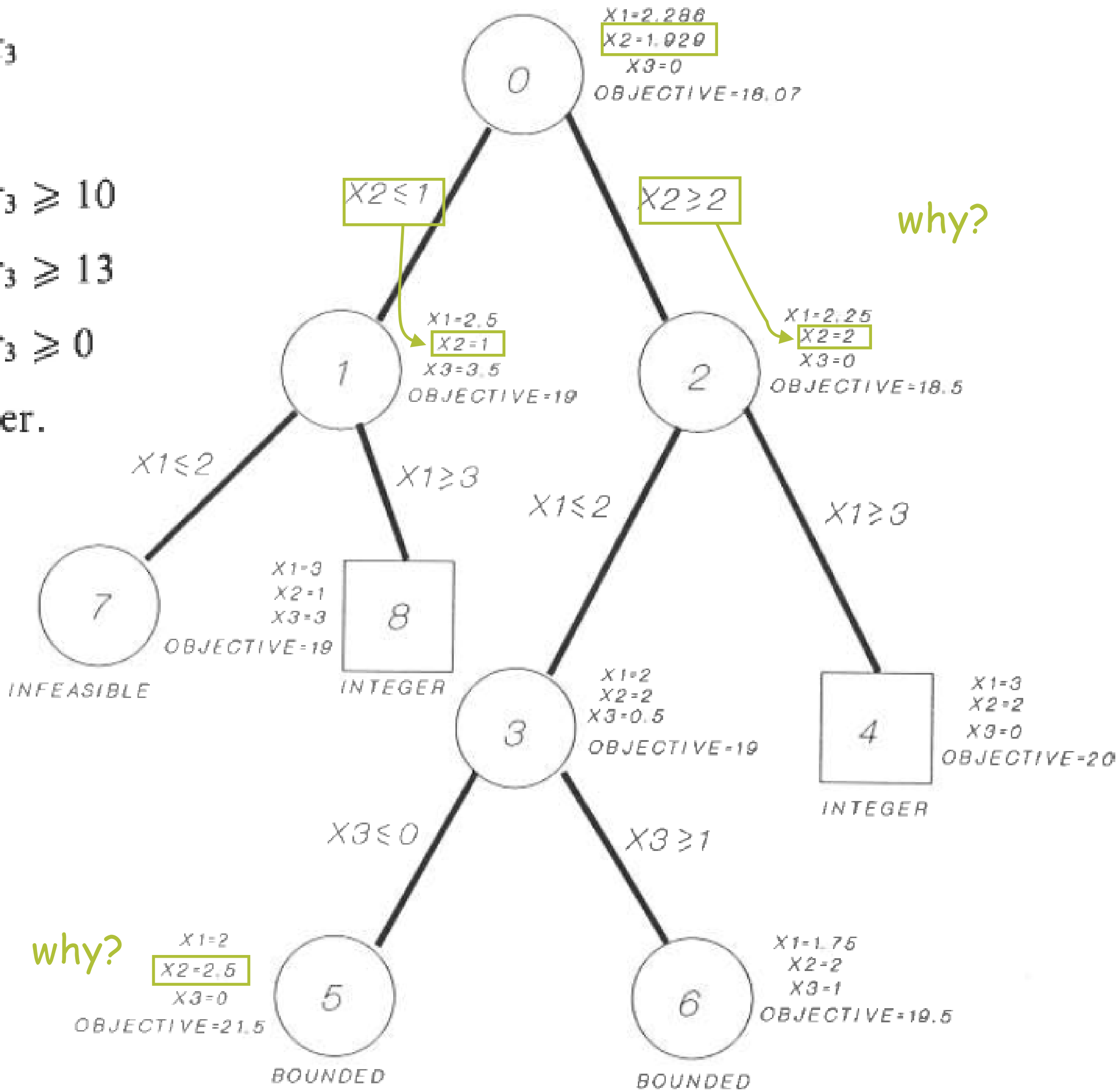


figure from
H. P. Williams

Remember branch-and-bound from constraint programming?

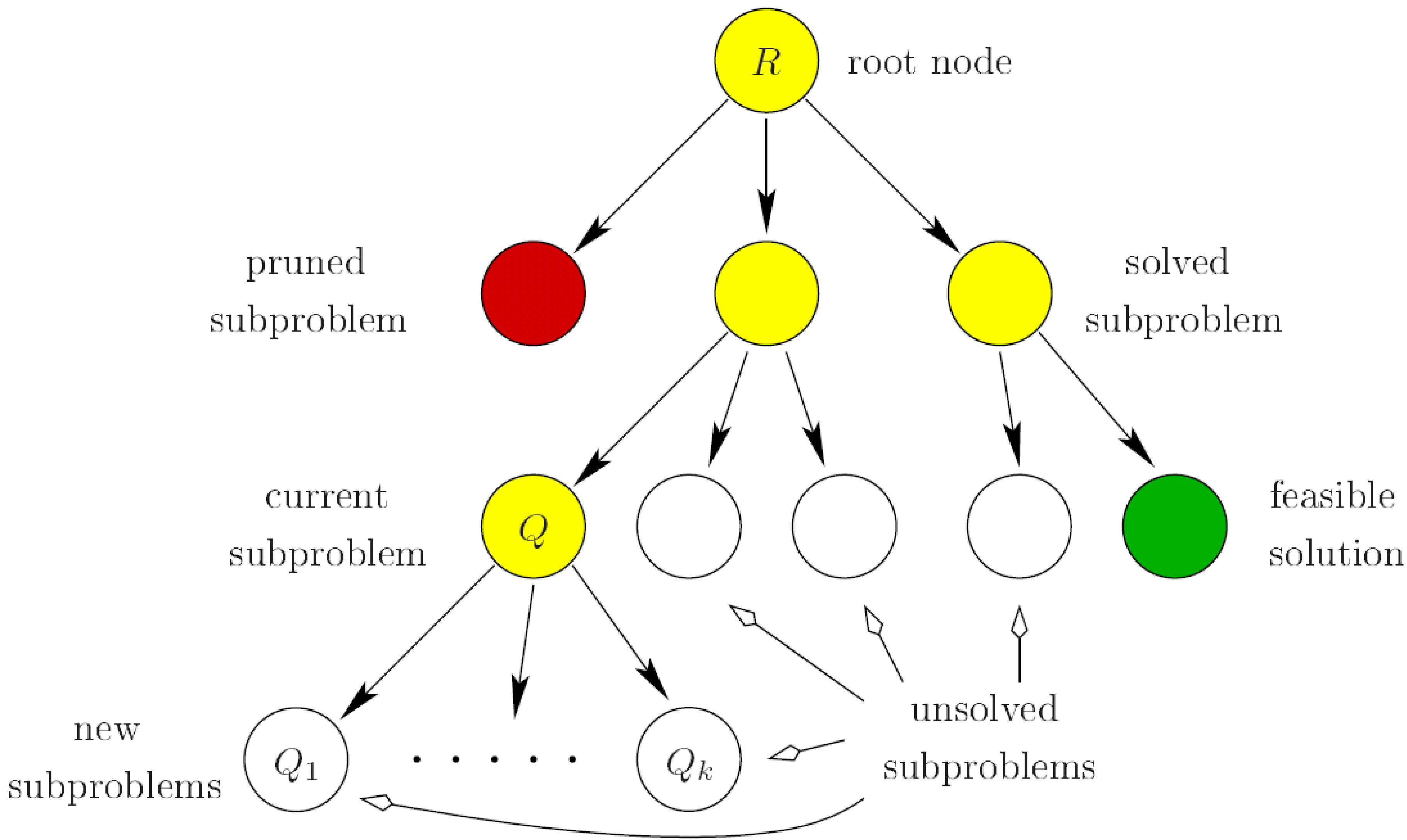


figure thanks to Tobias Achterberg

Or set c^, x^ to a known feasible solution.

Branch and bound: Pseudocode

In notation, ^ is upper bound (feasible but poor objective) – decreases *globally*

v is lower bound (good objective but infeasible) – increases *down the tree*

Input: $\boxed{\text{Minimization}}$ problem instance R. May simplify ("presolve") it first.

Output: Optimal solution $x^\star$ with value $c^\star$, or conclusion that R has no solution, indicated by $c^\star = \infty$.

Simplify if desired by propagation; then relax by dropping integrality constraints

1. Initialize $\mathcal{L} := \{R\}$, $\hat{c} := \infty$. [*init*]
2. If $\mathcal{L} = \emptyset$, stop and return $x^\star = \hat{x}$ and $c^\star = \hat{c}$. [*abort*]
3. Choose $Q \in \mathcal{L}$, and set $\mathcal{L} := \mathcal{L} \setminus \{Q\}$. [*select*]
4. Solve a relaxation Q_{relax} of Q. If Q_{relax} is empty, set $\check{c} := \infty$. Otherwise, let $\check{x}$ be an optimal solution of Q_{relax} and $\check{c}$ its objective value. [*solve*]
5. If $\check{c} \geq \hat{c}$, goto Step 2. [*bound*]
6. If $\check{x}$ is feasible for R, set $\hat{x} := \check{x}$, $\hat{c} := \check{c}$, and goto Step 2. [*check*]
7. Split Q into subproblems $Q = Q_1 \cup \ldots \cup Q_k$, set $\mathcal{L} := \mathcal{L} \cup \{Q_1, \ldots, Q_k\}$, and goto Step 2.

Can round and do stochastic local search to try to find a feasible solution x^ near xv, to improve upper bound c^ further

Branch&cut: add new constraints here (cutting planes, conflict clauses, or pick from huge set ("row generation"))
Branch&price: add new non-0 vars picked from huge set ("column gener.")

pseudocode thanks to Tobias Achterberg

How do we split into subproblems?

Where's the variable ordering?
Where's the value ordering?

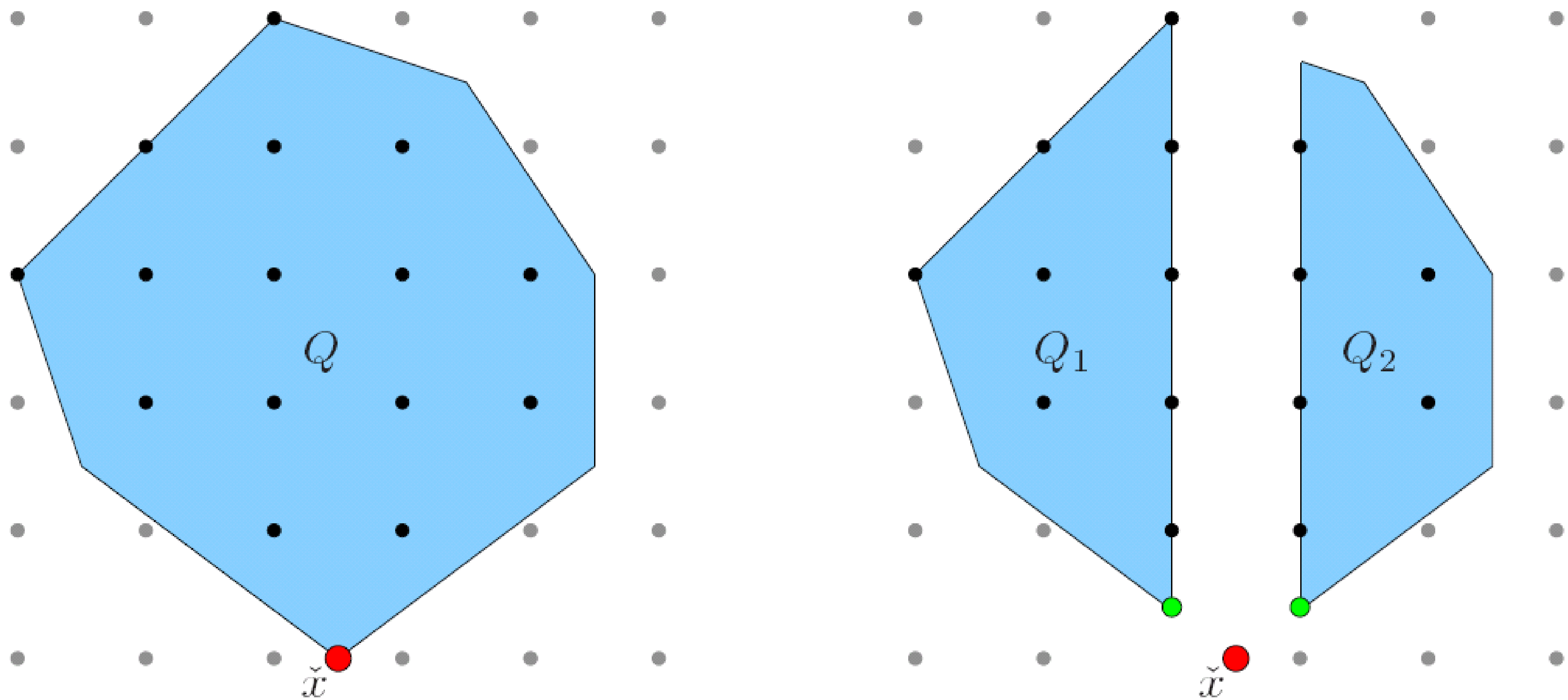


Figure 2.2. LP based branching on a single fractional variable.

figure thanks to Tobias Achterberg

How do we add new constraints?

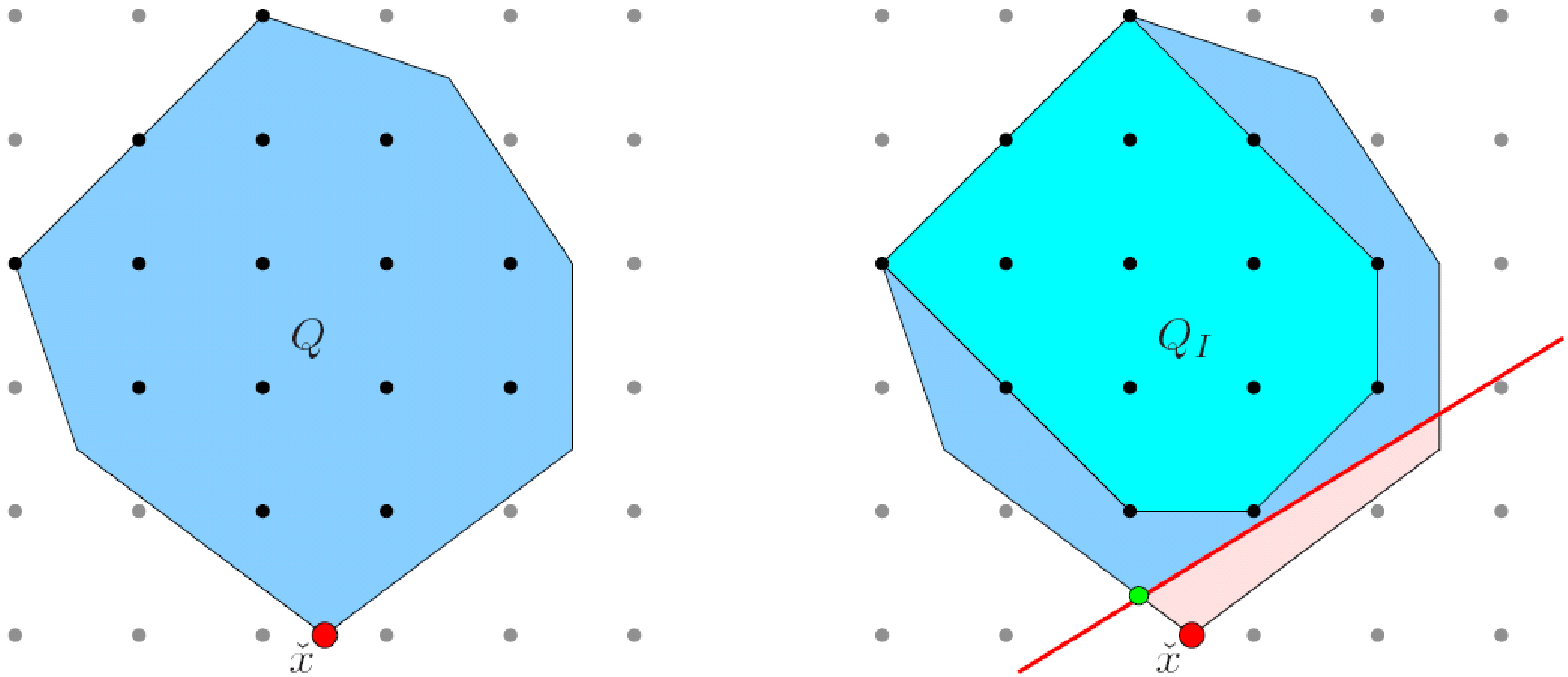


Figure 2.3. A cutting plane that separates the fractional LP solution $\check{x}$ from the convex hull Q_I of integer points of Q.

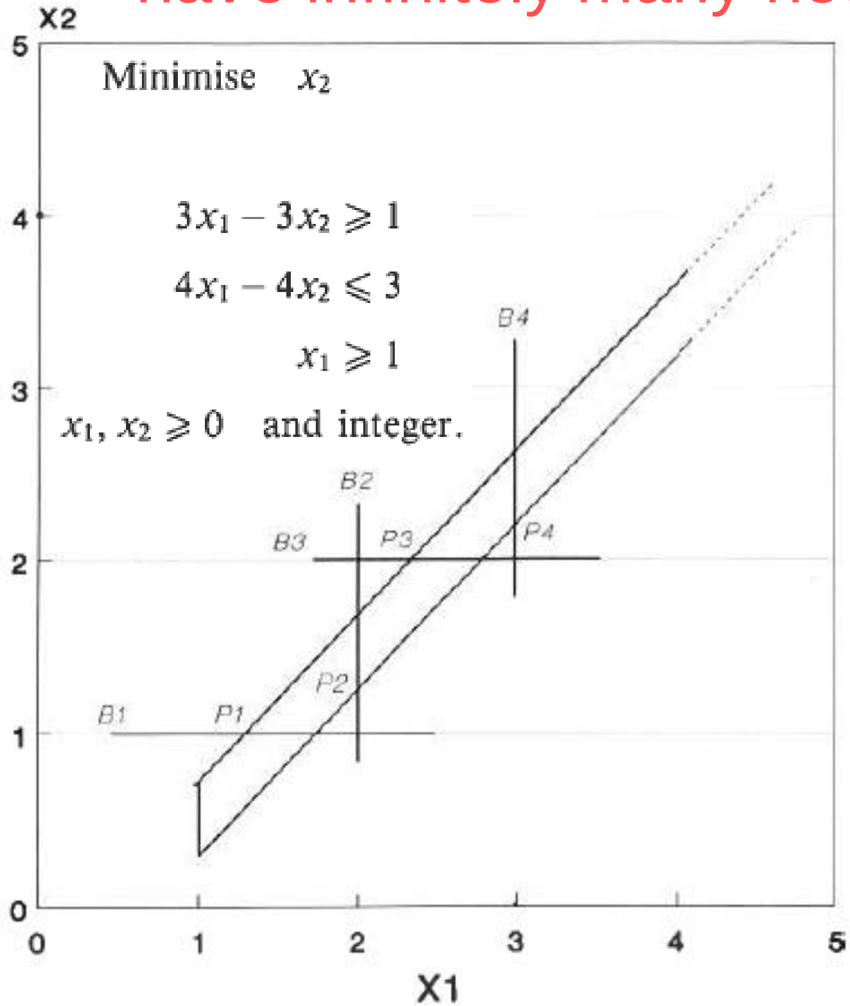
figure thanks to Tobias Achterberg

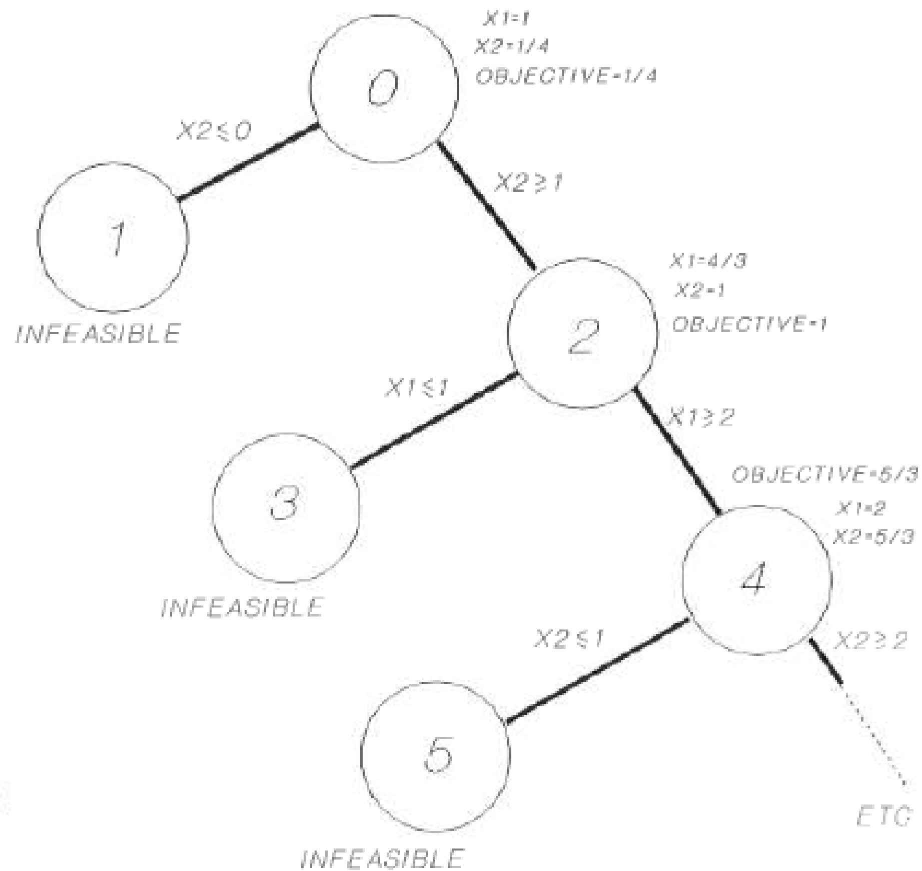
Variable & value ordering heuristics (at a given node)

- **Priorities:** User-specified var ordering
- **Most fractional branching:** Branch on variable farthest from int
- Branch on a variable that should tighten (hurt) the LP relaxation a lot
  - **Strong branching:** For several candidate variables, try rounding them and solving the LP relaxation (perhaps incompletely).
  - **Penalties:** If we rounded x up or down, how much would it tighten objective just on next iteration of dual simplex algorithm? (Dual simplex maintains an overly optimistic cost estimate that relaxes integrality and may be infeasible in other ways, too.)
  - **Pseudo-costs:** When rounding this variable in the past, how much has it *actually* tightened the LP relaxation objective (on average), per unit increase or decrease?
- Branching on SOS1 and SOS2

Warning

- If variables are unbounded, the search tree might have infinitely many nodes!

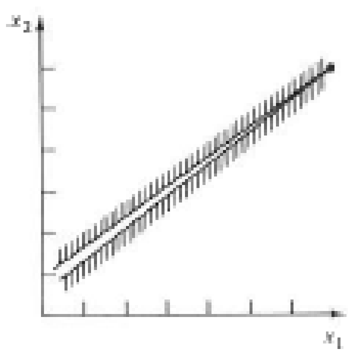




figures from H. P. Williams

Warning

- If variables are unbounded, the search tree might have infinitely many nodes!
- Fortunately, it's possible to compute bounds …
  - Given an LP or ILP problem (min $c \cdot x$ subj. to $Ax \leq b$, $x \geq 0$)
  - Where all numbers in A,b,c are integers; n vars, m constraints
  - If there's a finite optimum $c \cdot x$, each x_i is $\leq$ a bound whose log is
    - $O(m^2 \log m \log($ biggest integer in A or b $))$ [for LP]

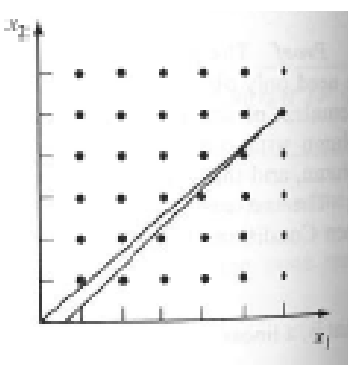


Intuition for LP: Only way to get LP optima far from the origin is to have slopes that are close but not quite equal … which requires large ints.

figures from Papadimitriou & Steiglitz

Warning

- If variables are unbounded, the search tree might have infinitely many nodes!
- Fortunately, it's possible to compute bounds …
  - Given an LP or ILP problem (min $c \cdot x$ subj. to $Ax \leq b$, $x \geq 0$)
  - Where all numbers in A,b,c are integers; n vars, m constraints
  - If there's a finite optimum x, each x_i is $\leq$ a bound whose <u>log</u> is
    - $O(m^2 \log m \log(\text{ biggest integer in A or b }))$ [for LP]
    - $O(\log n + m(\log n + \log(\text{biggest int in A, b, or c})))$ [for ILP]



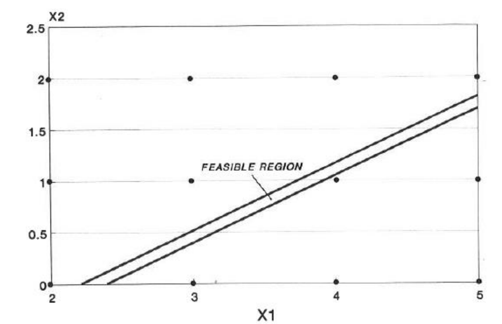


Figure 1.16 An infeasible integer programme with an unbounded linear programming relaxation

For ILP: A little trickier. (Could ILP have huge finite optimum if LP is unbounded? Answer: no, then ILP unbounded too.)

figures from Papadimitriou & Steiglitz

Reducing ILP to 0-1 ILP

- Given an LP or ILP problem (min c.x subj. to Ax=b, $x \geq 0$)
- Where all numbers in A,b,c are integers; n vars, m constraints
- If there's a finite optimum x, each x_i is $\leq$ a bound whose <u>log</u> is
  - O(log n + m(log n + log (biggest int in A, b, or c)) [for ILP]

- If log bound=100, then e.g. $0 \leq x_5 \leq 2^{100}$
- Remark: This bound enables a polytime reduction from ILP to 0-1 ILP
  - Remember: Size of problem = length of encoding, not size of #s
- Can you see how?
- Hint: Binary numbers are encoded with 0 and 1
- What happens to linear function like …+ 3 x_5 + … ?

Totally Unimodular Problems

- There are some ILP problems where nothing is lost by relaxing to LP!
  - "some mysterious, friendly power is at work" -- Papadimtriou & Steiglitz
  - All vertices of the <u>LP</u> polytope are integral anyway.
  - So regardless of the cost function, the <u>LP</u> has an optimal solution in integer variables (& maybe others)
  - No need for cutting planes or branch-and-bound.
  - This is the case when A is a totally unimodular ??? integer matrix, and b is integral. (c can be non-int.)

$$A\vec{x} \leq \vec{b} \quad (\text{or } A\vec{x} = \vec{b})$$

Totally Unimodular Cost Matrix A

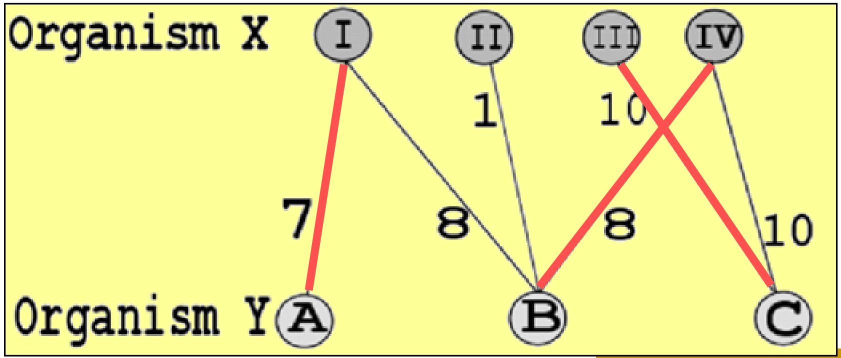
- A square integer matrix is called unimodular if its inverse is also integral.
- **Equivalently:** it has determinant 1 or -1.
  - (if det(A)=±1, then A^{-1} = adjoint(A) / det(A) is integral)
  - (if A, A^{-1} are integral, then det A, det A^{-1} are ints with product 1)
  - Matrices are like numbers, but more general. Unimodular matrices are the matrix generalizations of +1 and -1: you can divide by them without introducing fractions.
- A totally unimodular matrix is one whose square submatrices (obtained by crossing out rows or columns) are all either unimodular (det=±1) or singular (det=0).
  - Matters because simplex inverts non-singular square submatrices.

Some Totally Unimodular Problems

- The following common **graph problems** pick a subset of edges from some graph, or assign a weight to each edge in a graph.
  - Weighted bipartite matching
  - Shortest path
  - Maximum flow
  - Minimum-cost flow
- Their cost matrices are totally unimodular.
  - They satisfy the conditions of a superficial test that is sufficient to guarantee total unimodularity.
  - So, they can all be solved right away by the simplex algorithm or another LP algorithm like primal-dual.
  - All have well-known direct algorithms, but those can be seen as essentially just special cases of more general LP algorithms.

Some Totally Unimodular Problems

- The following common **graph problems** pick a subset of edges from some graph ...
  - Weighted matching in a bipartite graph



edge scores from drawing

Not needed! (just need $x_{ij} \geq 0$)

$$\max \sum_{ij} c_{ij} x_{ij} \quad \text{with } x_{ij} \text{ binary}$$

$$\text{subjto} \quad (\forall i) \sum_j x_{ij} \leq 1$$

$$(\forall j) \sum_i x_{ij} \leq 1$$

each top/bottom node has at most one edge

If we formulate as $Ax \leq b$, $x \geq 0$, the A matrix is totally unimodular:

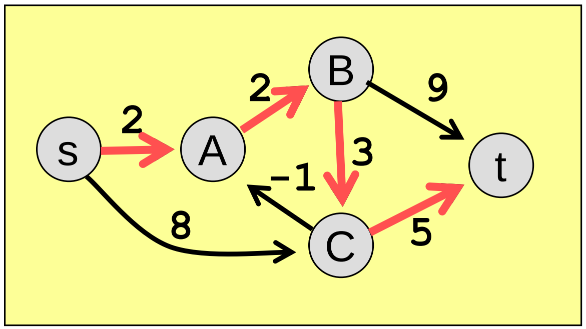
Sufficient condition: Each column (for edge x_{ij}) has at most 2 nonzero entries (for i and j).
These are both +1 (or both -1) and are in different "halves" of the matrix.
(Also okay if they are +1 and -1 and are in same "half" of the matrix.)

	$x_{I,A}$	$x_{I,B}$	$x_{II,B}$	$x_{III,C}$	$x_{IV,B}$	$x_{IV,C}$
(i=I)	1	1	0	0	0	0
(i=II)	0	0	1	0	0	0
(i=III)	0	0	0	1	0	0
(i=IV)	0	0	0	0	1	1
(j=A)	1	0	0	0	0	0
(j=B)	0	1	1	0	1	0
(j=C)	0	0	0	1	0	1

drawing from Edwards M T et al. Nucl. Acids Res. 2005;33:3253-3262 (Oxford Univ. Press)

Some Totally Unimodular Problems

- The following common **graph problems** pick a subset of edges from some graph ...
  - Shortest path from s to t in a directed graph



edge scores from drawing

$$\min \sum_{ij} c_{ij} x_{ij} \quad \text{with } x_{ij} \text{ binary}$$

$$\text{subjto} \quad \sum_j x_{sj} = 1, \sum_j x_{jt} = 1$$

$$(\forall j \notin \{s,t\}) \sum_i x_{ij} = \sum_k x_{jk}$$

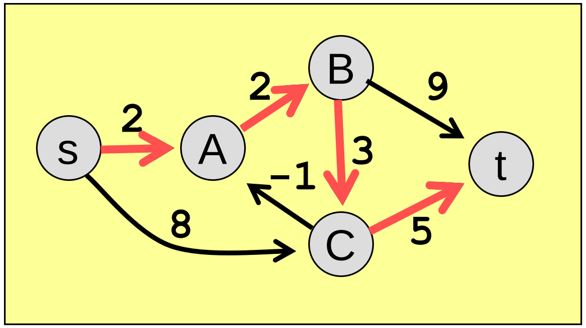
Can formulate as Ax = b, x ≥ 0 so that A matrix is totally unimodular:

	x_{sA}	x_{sC}	x_{AB}	x_{BC}	x_{CA}	x_{Bt}	x_{Ct}
(s)	1	1	0	0	0	0	0
(j=A)	-1	0	1	0	-1	0	0
(j=B)	0	0	-1	1	0	1	0
(j=C)	0	-1	0	-1	1	0	1
(t)	0	0	0	0	0	-1	-1

Q: Can you prove that every *feasible* solution is a path?
A: No: it could be a path plus some cycles.
But then can reduce cost by throwing away the cycles. So *optimal* solution has no cycles.

Some Totally Unimodular Problems

- The following common **graph problems** pick a subset of edges from some graph ...
  - Shortest path from s to t in a directed graph



edge scores from drawing

$$\min \sum_{ij} c_{ij} x_{ij} \quad \text{with } x_{ij} \text{ binary}$$

Not needed! (just need xij ≥ 0

$$\text{subjto} \quad \sum_j x_{sj} = 1, \sum_j -x_{jt} = -1$$

$$(\forall j \notin \{s,t\}) \sum_i -x_{ij} + \sum_k x_{jk} = 0$$

Can formulate as Ax = b, x ≥ 0 so that A matrix is totally unimodular:

Sufficient condition: Each column (for edge x_{ij}) has at most 2 nonzero entries (for i and j).
These are +1 and -1 and are in the same "half" of the matrix.
(Also okay to be both +1 or both -1 and be in different "halves.")

	x_{sA}	x_{sC}	x_{AB}	x_{BC}	x_{CA}	x_{Bt}	x_{Ct}
(s)	1	1	0	0	0	0	0
(j=A)	-1	0	1	0	-1	0	0
(j=B)	0	0	-1	1	0	1	0
(j=C)	0	-1	0	-1	1	0	1
(t)	0	0	0	0	0	-1	-1

(this "half" is empty; can divide rows into "halves" in any way that satisfies sufficient condition)

Some Totally Unimodular Problems

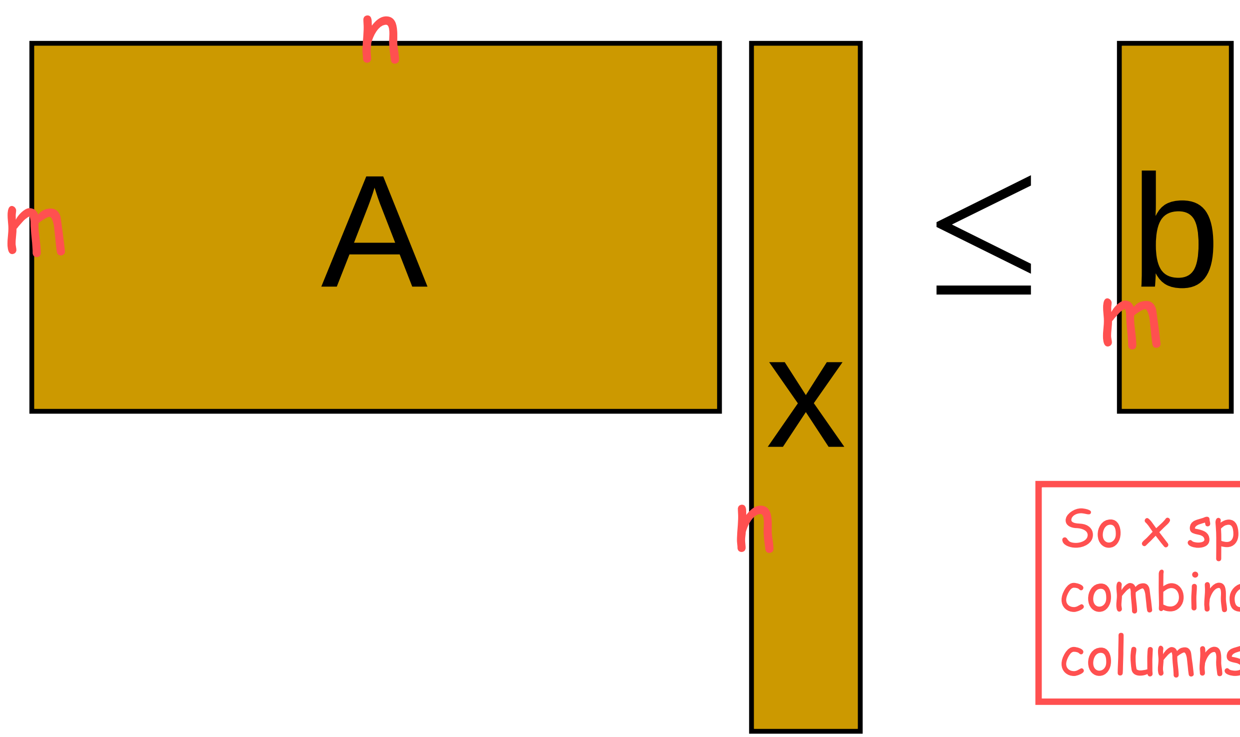
- The following common **graph problems** pick a subset of edges from some graph ...
  - Maximum flow (previous problems can be reduced to this)
  - Minimum-cost flow

- Cost matrix is rather similar to those on the previous slides, but with additional "capacity constraints" like $x_{ij} \leq k_{ij}$
- Fortunately, if A is totally unimodular, so is A with I (the identity matrix) glued underneath it to represent the additional constraints

Solving Linear Programs

Canonical form of an LP

- min c · x   subject to $Ax \le b$, $x \ge 0$
- m constraints (rows)
  n variables (columns)   (usually m < n)



So x specifies a linear combination of the columns of A.

Fourier-Motzkin elimination

- An example of our old friend **variable elimination.**
- Geometrically:
  - Given a bunch of inequalities in x, y, z.
  - These define a 3-dimensional **polyhedron** P_3.
  - Eliminating z gives the shadow of P_3 on the xy plane.
    - A **polygon** P_2 formed by all the (x,y) values for which $\exists z\ (x,y,z) \in P_3$.

    Warning: P_2 may have more edges than P_3 has faces. That is, we've reduced # of variables but perhaps increased # of constraints. As usual, might choose variable z carefully (cf. induced width).
  - Eliminating y gives the shadow of P_2 on the x line.
    - A **line segment** P_1 formed by all the x values for which $\exists y\ (x,y) \in P_2$.
    - Now we know the min and max possible values of x.
  - Backsolving: Choose best $x \in P_1$. For any such choice,
    - can choose y with $(x,y) \in P_2$. And for any such choice,
    - can choose z with $(x,y,z) \in P_3$. A feasible solution with optimal x!

Thanks to the $\exists$ properties above.

Remember variable elimination for SAT?

Davis-Putnam

- This procedure (resolution) eliminates all copies of X **and** ~X.
  - We're done in n steps. So what goes wrong?
  - Size of formula can square at each step.

some two clauses in φ

Resolution fuses <u>each</u> pair (V v W v **~X**) ^ (**X** v Y v Z) into (V v W v Y v Z)
Justification #1: Valid way to eliminate X (reverses CNF → 3-CNF idea).
Justification #2: Want to recurse on a CNF version of ((φ ^ X) v (φ ^ ~X))
Suppose $\varphi = \alpha \wedge \beta \wedge \gamma$
where α is clauses with ~X, β with X, γ with neither
Then ((φ ^ X) v (φ ^ ~X)) = (α' ^ γ) v (β' ^ γ) by unit propagation
where α' is α with the ~X's removed, β' similarly.
= (α' v β') ^ γ = (α'_1 v β'_1) ^ (α'_1 v β'_2) ^ … ^ (α'_{99} v β'_{99}) ^ γ

Fourier-Motzkin elimination

- Variable elimination on a set of inequalities

- ~~minimize x + y + z~~ minimize a

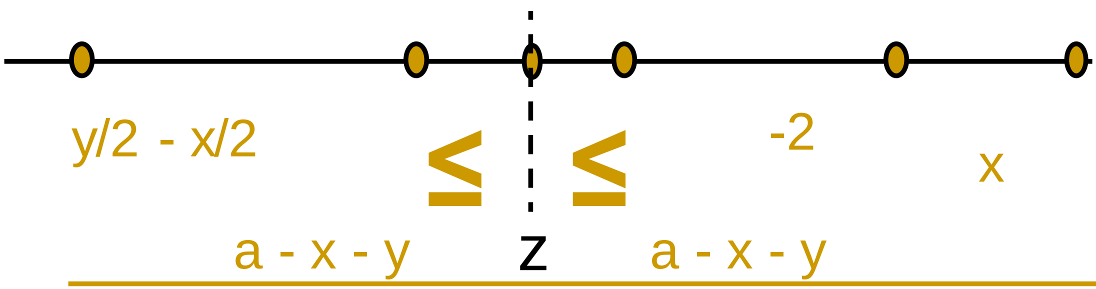
  subject to
  $x - z \ge 0$
  $x - y \ge 0$
  $-z \ge 2$
  $2z + x - y \ge 0$
  $a \le x+y+z$
  $a \ge x+y+z$

➡

Solve for z

$z \le x$
(doesn't mention z; leave alone)
$z \le -2$
$z \ge y/2 - x/2$
$z \ge a - x - y$
$z \le a - x - y$

∃ a value of z satisfying these constraints
iff each of these ≤ each of these



Eliminate z

$y/2 - x/2 \le -2$
$y/2 - x/2 \le x$
$y/2 - x/2 \le a - x - y$

$a - x - y \le -2$
$a - x - y \le x$
$a - x - y \le a - x - y$

$x - y \ge 0$ (unchanged)

example adapted from Ofer Strichman

Fourier-Motzkin elimination

- Variable elimination on a set of inequalities.
- To eliminate variable z, take each inequality involving z and solve it for z. Gives $\mathbf{z \geq \alpha_1, z \geq \alpha_2, \ldots, z \leq \beta_1, z \leq \beta_2, \ldots}$
  - Each α_i or β_j is a linear function of the other vars a,b,…,y.
- Replace these inequalities by $\boldsymbol{\alpha_i \leq \beta_j}$ for each (i,j) pair.
  - Equivalently, max $\boldsymbol{\alpha}$ ≤ min $\boldsymbol{\beta}$.
  - These equations are true of an assignment a,b,…,y iff it can be extended with a consistent value for z.
- Similar to resolution of CNF-SAT clauses in Davis-Putnam algorithm! But similarly, may square the # of constraints. ☹
- Repeat to eliminate variable y, etc.
- If one of our equations is "**a = [linear cost function]**," then at the end, we're left with just lower and upper bounds on a.
  - Now easy to min or max a! Back-solve to get b, c, … z in turn.

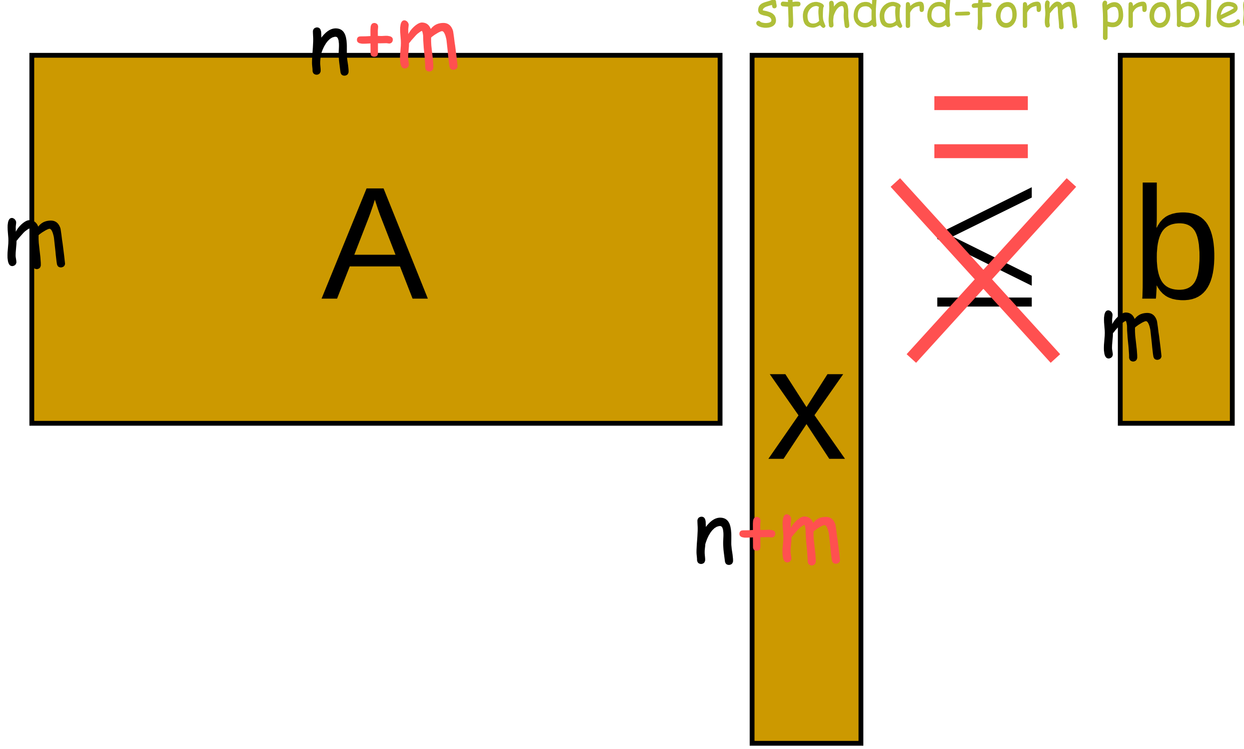
Simplex Algorithm: Basic Insight

- n variables $x_1, \ldots x_n$
- m constraints, plus n more for $x_1, \ldots x_n \geq 0$
- Each constraint is a hyperplane
- Every vertex of the polytope is defined by an intersection of n hyperplanes
- Conversely, given n hyperplanes, we can find their intersection (if any) by solving a system of n linear equations in n variables
- So, we just have to pick *which* n constraints to intersect
- Sometimes we'll get an infeasible solution (not a vertex)
- Sometimes we'll get a suboptimal vertex: then move to an adjacent, better vertex by replacing just 1 of the n constraints

From Canonical to Standard Form

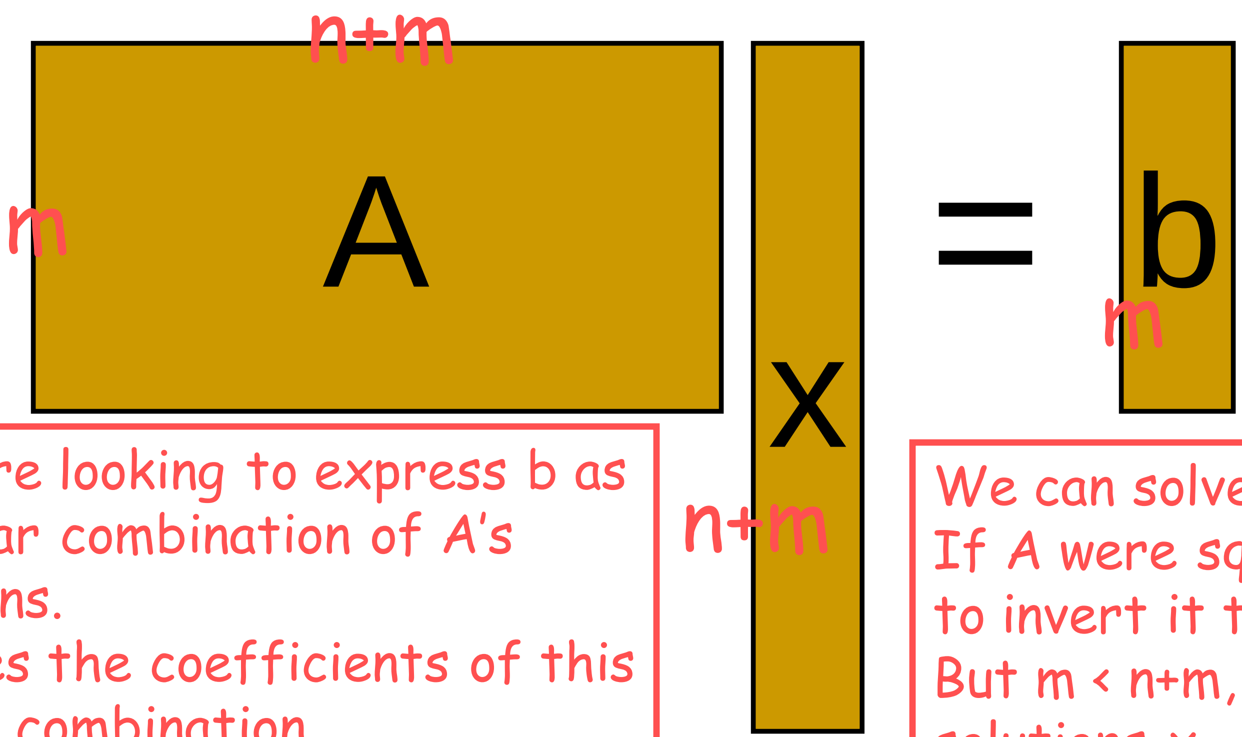
- min $c \cdot x$ subject to $\cancel{Ax \le b}$ $Ax = b$, $x \ge 0$
- m constraints (rows)
  n+m variables (columns)

(Sometimes # vars is still called n, even in standard form. It's usually > # constraints. I'll use n+m to denote the # of vars in a standard-form problem – you'll see why.)



From Canonical to Standard Form

- min c · x subject to Ax = b, x ≥ 0
- m constraints (rows)
  n+m variables (columns)

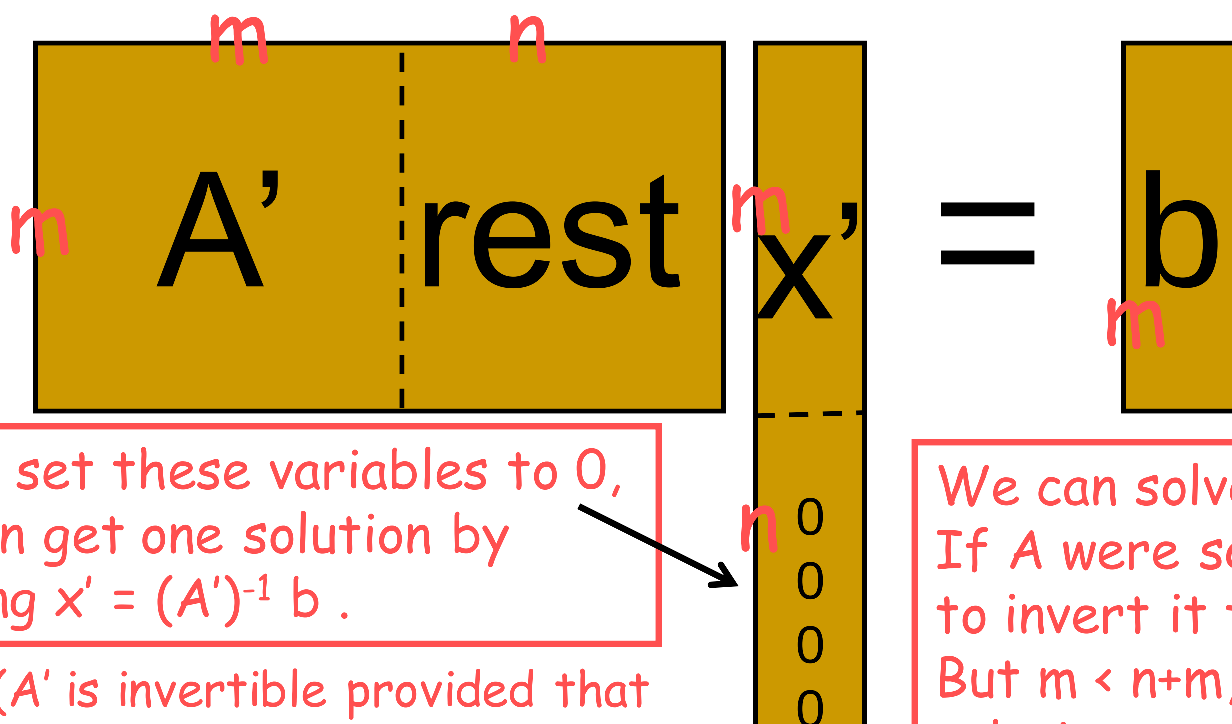


We are looking to express b as a linear combination of A's columns.
x gives the coefficients of this linear combination.

We can solve linear equations!
If A were square, we could try to invert it to solve for x.
But m < n+m, so there are many solutions x.
(To choose one, we min c·x.)

Standard Form

- min c · x   subject to   Ax = b, x ≥ 0
- m constraints (rows)
  n variables (columns)   (usually m < n)



If we set these variables to 0, we can get one solution by setting $x' = (A')^{-1}$ b .

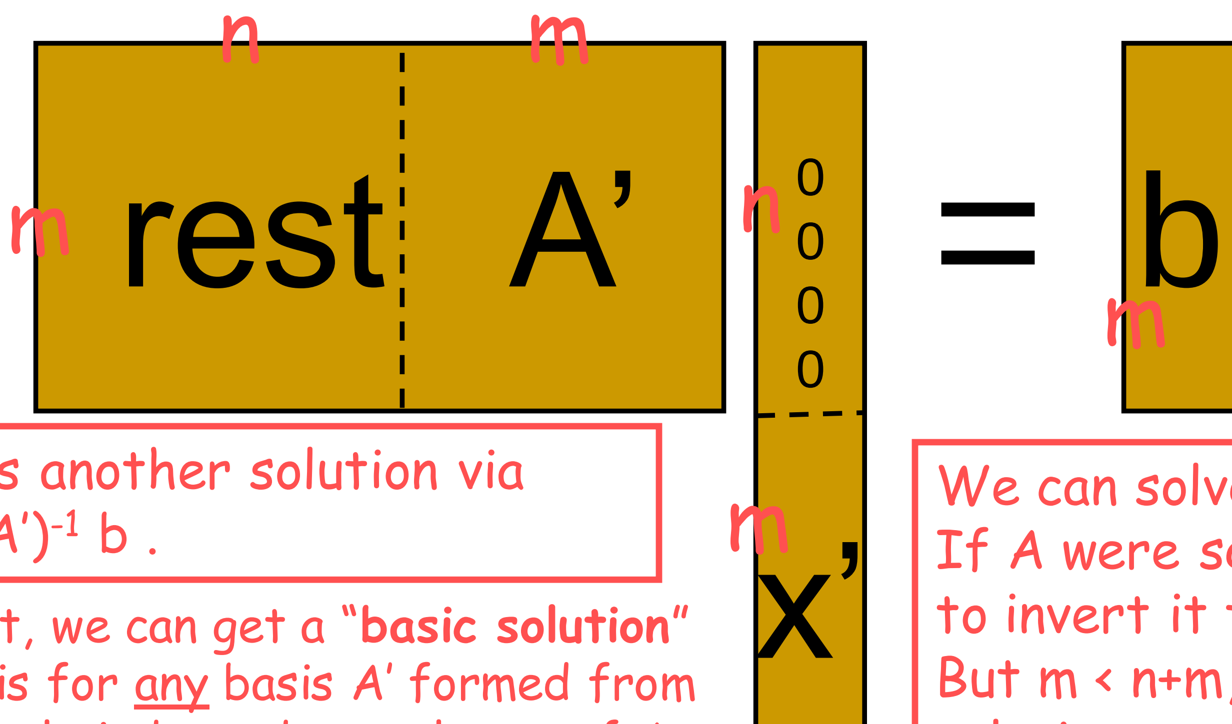
(A' is invertible provided that the m columns of A' are linearly independent.)

We can solve linear equations!
If A were square, we could try to invert it to solve for x.
But m < n+m, so there are many solutions x.
(To choose one, we min c·x.)

Standard Form

Notice that the bfs in the picture is optimal when the cost vector is $c = (1,1,1,1,0,0,0,0,\dots)$
Similarly, <u>any</u> bfs is optimal for <u>some</u> cost vector.
Hmm, sounds like polytope vertices…

- min $c \cdot x$ subject to $Ax = b$, $x \geq 0$
- m constraints (rows)
  n variables (columns) (usually $m < n$)



Remark: If A is totally unimodular, then the bfs $(A')^{-1} b$ will be integral (assuming b is).

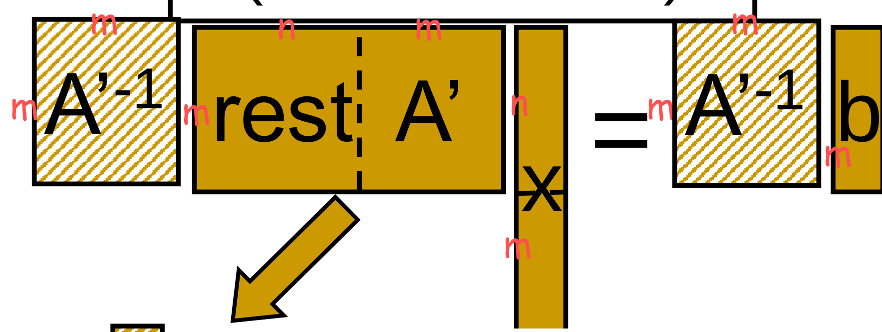
Here's another solution via $x' = (A')^{-1} b$.

In fact, we can get a "**basic solution**" like this for <u>any</u> basis A' formed from m linearly independent columns of A.
This x is a "**basic <u>feasible</u> solution**" **(bfs)** if $x \geq 0$ (recall that constraint?).

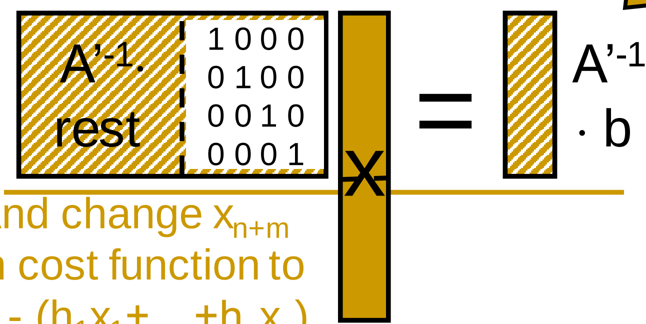
We can solve linear equations!
If A were square, we could try to invert it to solve for x.
But m < n+m, so there are many solutions x.
(To choose one, we min $c \cdot x$.)

Canonical vs. Standard Form

$Ax \leq b$ $x \geq 0$ m inequalities + n inequalities (**n** variables)	add m slack variables (one per constraint) → ← Eliminate last m vars (how?)	$Ax = b$ $x \geq 0$ m equalities + n+m inequalities (**n+m** variables)



Multiply Ax=b through by A'^{-1}. This gives us the kind of Ax=b that we'd have gotten by starting with $Ax \leq b$ and adding 1 slack var per constraint. Now can eliminate slack vars.



And change x_{n+m} in cost function to $b - (h_1x_1+\ldots+h_nx_n)$.

Eliminating last m vars turns the last m "≥ 0" constraints & the m constraints ("Ax=b") into m inequalities ("$Ax \leq b$").

E.g., have 2 constraints on x_{n+m}: $x_{n+m} \geq 0$ and the last row, namely $(h_1x_1+\ldots+h_nx_n) + x_{n+m} = b$. To elim x_n, replace them with $(h_1x_1+\ldots+h_nx_n) \leq b$.

Canonical vs. Standard Form

add m slack variables
(one per constraint)

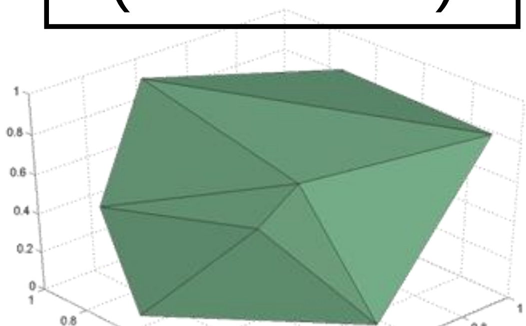
Eliminate last m vars

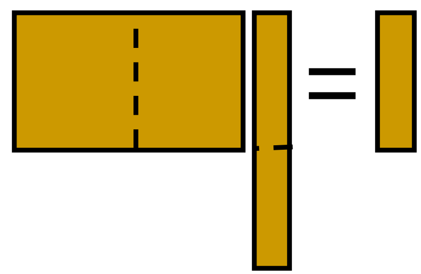
$$Ax \leq b$$
$$x \geq 0$$

m inequalities
+ n inequalities
(**n** variables)

$$Ax = b$$
$$x \geq 0$$

m equalities
+ n+m inequalities
(**n+m** variables)





vertex

(defined by intersecting n of the constraints, each of which reduces dimensionality by 1)

Pick n of the
n+m constraints
to be tight

bfs

(defined by selecting n of the variables to be 0)

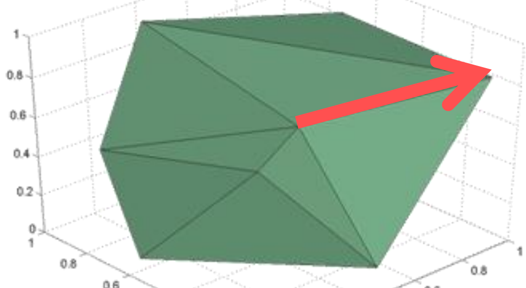
Simplex algorithm

At right, expressed an unused column C_5 as linear combination of basis: $C_5 = C_1 + 2C_2 - C_3$.
Gradually phase in unused column C_5 while phasing out $C_1 + 2C_2 - C_3$, to keep Ax=b.
Easy to solve for max ε (=2) that keeps $x \geq 0$.
Picked C_5 because increasing ε improves cost.

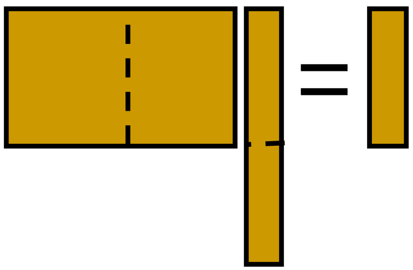
Suppose n+m=6 and n=3
Denote A's columns by $C_1 \ldots C_6$
x=(5,4,7,0,0,0) is the current bfs (n zeroes)
So C_1, C_2, C_3 form a basis of $\mathbf{R}^m$ and Ax=b for

x=(5, 4, 7, 0, 0, 0) $5C_1 + 4C_2 + 7C_3 = b$
x=(4.9, 3.8, 7.1, 0, 0.1, 0) …
x=(4.8, 3.6, 7.2, 0, 0.2, 0) …
x=(5-ε, 4-2ε, 7+ε, 0, ε, 0) …
x=(3, 0, 9, 0, 2, 0) $3C_1 + 9C_3 + 2C_5 = b$
is the new bfs

- **Geometric interpretation**
- Move to an adjacent vertex (current vertex defined by n facets C_4, C_5, C_6; choose to remove C_5 facet by allowing slack; now C_4, C_6 define edge)



- **Computational implementation**
- Move to an adjacent bfs (add 1 basis column, remove 1)



vertex
(defined by intersecting n of the constraints)

Pick n of the n+m constraints to be tight

bfs
(defined by selecting n of the variables to be 0 ➔ n tight constraints; solve for slack in other constraints)

Canonical vs. Standard Form

Eliminate last m vars

$Ax \leq b$

$x \geq 0$

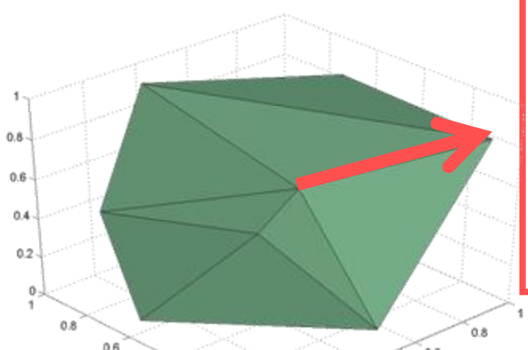
m inequalities
+ n inequalities
(**n** variables)

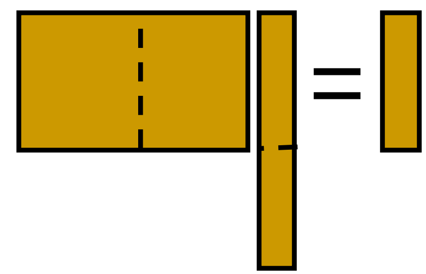
Cost of origin is easy to compute (it's a const in cost function). Eliminating a different set of m variables (picking a different basis) would rotate/reflect/squish the polytope & cost hyperplane to put a different vertex at origin, aligning that vertex's n constraints with the orthogonal $x \geq 0$ hyperplanes. This is how simplex algorithm tries different vertices!

$Ax = b$

$x \geq 0$

m equalities
+ n+m inequalities
(**n+m** variables)





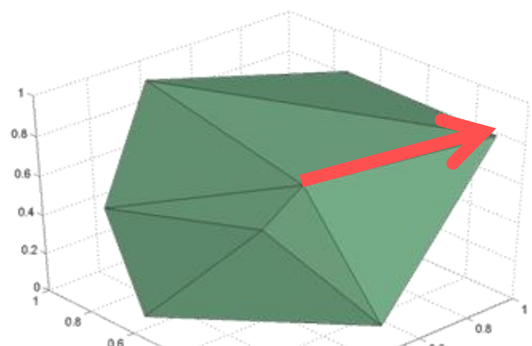
vertex
(defined by intersecting n of the constraints)

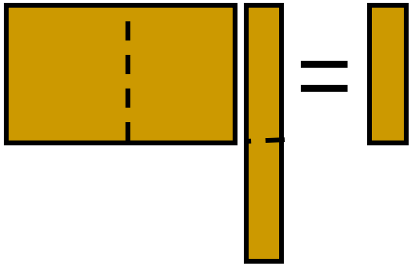
Pick n of the n+m constraints to be tight

bfs
(defined by selecting n of the variables to be 0 → n tight constraints; solve for slack in other constraints)

Simplex algorithm: More discussion

- How do we pick which column to phase out (determines which edge to move along)?
  - How to avoid cycling back to an old bfs (in case of ties)?
- Alternative and degenerate solutions?
- What happens with unbounded LPs?
- How do we find a first bfs to start at?
  - Simplex phase I: Add “artificial” slack/surplus variables to make it easy to find a bfs, then phase them out via simplex. (Will happen automatically if we give the artificial variables a high cost.)
  - Or, just find <u>any</u> basic solution; then to make it feasible, phase out negative variables via simplex.
  - Now continue with phase II. If phase I failed, no bfs exists for original problem, because:
    - The problem was infeasible (<u>incompatible</u> constraints, so quit and return UNSAT).
    - Or the m rows of A aren’t linearly independent (<u>redundant</u> constraints, so throw away the extras & try again).





vertex

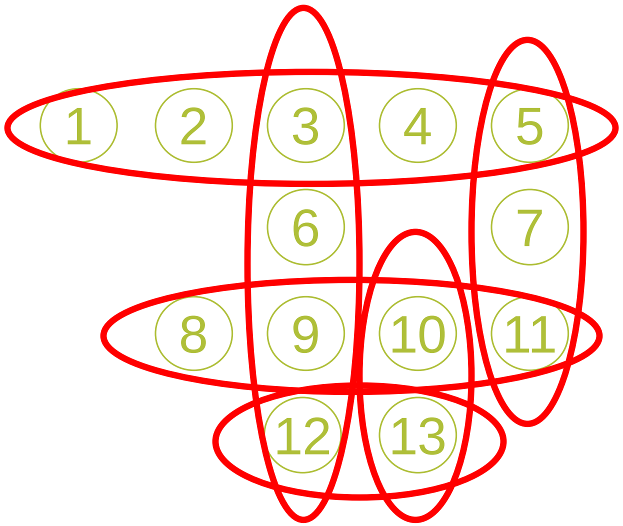
(defined by intersecting n of the constraints)

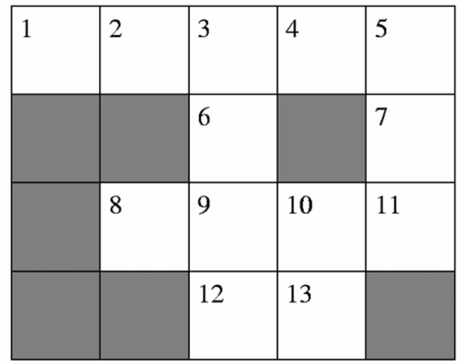
Pick n of the n+m equalities to be tight

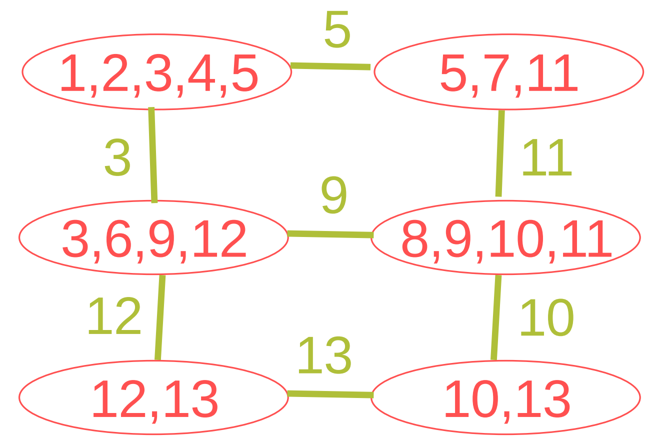
bfs

(defined by selecting n of the variables to be 0 ➔ n tight constraints; solve for slack in other constraints)

Recall: Duality for Constraint Programs







original ("primal") problem:
one variable per letter,
constraints over up to 5 vars

transformed ("dual") problem:
one var per word, 2-var constraints.

Old constraints → new vars

Old vars → new constraints

Warning: Unrelated to AND-OR duality from SAT

slide thanks to Rina Dechter (modified)

Duality for Linear Programs *(canonical form)*

Primal problem		Dual problem	
	$\max c \cdot x$	$\min b \cdot y$	
(m)	$Ax \le b$	$A^T y \ge c$	(n)
(n)	$x \ge 0$	$y \ge 0$	(m)

dualize

Old constraints → new vars

Old vars → new constraints

Where Does Duality Come From?

- We gave an asymptotic upper bound on max $c \cdot x$ (to show that integer linear programming was in NP).
- But it was very large. Can we get a tighter bound?

- As with Chvátal cuts and Fourier-Motzkin elimination, let's take linear combinations of the ≤ constraints, this time to get an upper bound on the *objective*.
  - As before, there are lots of linear combinations.
  - Different linear combinations ➔ different upper bounds.
  - Smaller (tighter) upper bounds are more useful.
  - Our *smallest* upper bound might be tight and equal max $c \cdot x$.

Where Does Duality Come From?

- Back to linear programming. Let's take linear combinations of the ≤ constraints, to get various upper bounds on the *objective*.
- max $\quad 4x_1 + x_2$ subject to $x_1, x_2 \geq 0$ and
  C_1: $\quad 2x_1 + 5x_2 \leq 10$
  C_2: $\quad x_1 - 2x_2 \leq 8$
- Can you find an upper bound on the objective?
  - Hint: Derive a new inequality from $C_1 + 2*C_2$
- What if the objective were $3x_1 + x_2$ instead?
  - Does it help that we already got a bound on $4x_1 + x_2$?

example from Rico Zenklusen

Where Does Duality Come From?

- Back to linear programming. Let's take linear combinations of the ≤ constraints, to get various upper bounds on the *objective*.
- max $2x_1 + 3x_2$ subject to $x_1, x_2 \geq 0$ and
  C_1: $x_1 + x_2 \leq 12$
  C_2: $2x_1 + x_2 \leq 9$
  C_3: $x_1 \leq 4$
  C_4: $x_1 + 2x_2 \leq 10$
- objective = $2x_1 + 3x_2 \leq 2x_1 + 4x_2 \leq 20$ $(2{*}C_4)$
- objective = $2x_1 + 3x_2 \leq 2x_1 + 3x_2 \leq 22$ $(1{*}C_1 + 1{*}C_4)$
- objective = $2x_1 + 3x_2 \leq 3x_1 + 3x_2 \leq 19$ $(1{*}C_2 + 1{*}C_4)$

positive coefficients
so ≤ doesn't flip

example from Rico Zenklusen

Where Does Duality Come From?

- Back to linear programming. Let's take linear combinations of the ≤ constraints, to get various upper bounds on the *objective*.
- max $2x_1 + 3x_2$ subject to $x_1, x_2 \geq 0$ and

$$
\begin{array}{llcll}
C_1: & x_1 + x_2 & \leq & 12 \\
C_2: & 2x_1 + x_2 & \leq & 9 \\
C_3: & x_1 & \leq & 4 \\
C_4: & x_1 + 2x_2 & \leq & 10
\end{array}
$$

General case:
$y_1C_1 + y_2C_2 + y_3C_3 + y_4C_4$
with $y_1,\ldots y_4 \geq 0$
so that inequalities don't flip

- $(y_1+2y_2+y_3+y_4)x_1 + (y_1+y_2+2y_4)x_2 \leq 12y_1 + 9y_2+4y_3+10y_4$
- Gives an upper bound on the objective $2x_1 + 3x_2$ if $y_1+2y_2+y_3+y_4 \geq 2$, $y_1+y_2+2y_4 \geq 3$
- We want to find the smallest such bound:

  min $12y_1 + 9y_2+4y_3+10y_4$

example from Rico Zenklusen

Duality for Linear Programs *(canonical form)*

Primal problem		Dual problem	
	$\max\ c \cdot x$	$\min\ b \cdot y$	
(m)	$Ax \le b$	$A^T y \ge c$	(n)
(n)	$x \ge 0$	$y \ge 0$	(m)

dualize

- The form above assumes (max,≤) ⇔ (min,≥).
- Extensions for LPs in general form:
  - Any reverse constraints ((max,≥) or (min,≤)) ⇔ negative vars
  - So, any equality constraints ⇔ unbounded vars
    (can simulate with pair of constraints ⇔ pair of vars)
- Also, degenerate solution (# tight constraints > # vars)
  ⇔ alternative optimal solutions (choice of nonzero vars)

Dual of dual = Primal

Linear programming duals are "reflective duals" (not true for some other notions of duality)

Primal problem → dualize → Dual problem

	Primal problem		Dual problem	
	$\max c \cdot x$		$\min b \cdot y$	
(m)	$Ax \leq b$		$A^T y \geq c$	(n)
(n)	$x \geq 0$		$y \geq 0$	(m)

Just negate A, b, and c

Equivalent to primal ← dualize ← Equivalent to dual

	Equivalent to primal		Equivalent to dual	
	$\min (-c) \cdot x$		$\max (-b) \cdot y$	
(m)	$(-A)x \geq -b$		$(-A^T)y \leq (-c)$	(n)
(n)	$x \geq 0$		$y \geq 0$	(m)

Primal & dual "meet in the middle"

Primal problem			Dual problem
	$\max c \cdot x$		$\min b \cdot y$
(m)	$Ax \le b$	(n)	$A^T y \ge c$
(n)	$x \ge 0$	(m)	$y \ge 0$

- We've seen that for <u>any</u> feasible solutions x and y, $c \cdot x \le b \cdot y$.
  - $b \cdot y$ provides a Lagrangian upper bound on $c \cdot x$ for any feasible y.
- So if $c \cdot x = b \cdot y$, both must be optimal!
  - (Remark: For nonlinear programming, the constants in the dual constraints are partial derivatives of the primal constraint and cost function. The equality condition is then called the Kuhn-Tucker condition. Our linear programming version is a special case of this.)
- For LP, the converse is true: optimal solutions always have $c \cdot x = b \cdot y$!
  - Not true for nonlinear programming or ILP.

Primal & dual "meet in the middle"

dual

$\min b\cdot y$

$A^T y \geq c$

$y \geq 0$

Not feasible under primal constraints

$b\cdot y$

Max achievable under primal constraints

Min achievable under dual constraints

$c\cdot x$

primal

$\max c\cdot x$

$Ax \leq b$

$x \geq 0$

Not feasible under dual constraints

$c\cdot x \leq b\cdot y$ for all feasible (x,y).
(So if one problem is unbounded, the other must be infeasible.)

Duality for Linear Programs *(standard form)*

Primal and dual are related constrained optimization problems, each in n+m dimensions

Primal problem	Dual problem
$\max c \cdot x$	$\min b \cdot y$
$Ax + s = b$	$A^T y - t = c$
$x \geq 0$ (n struct vars)	$y \geq 0$ (m struct vars)
$s \geq 0$ (m slack vars)	$t \geq 0$ (n surplus vars)

- Now we have n+m variables and they are in 1-to-1 correspondence.
- At primal optimality:
  - Some m "basic" vars of primal can be ≥ 0. The n non-basic vars are 0.
- At dual optimality:
  - Some n "basic" vars of dual can be ≥ 0. The m non-basic vars are 0.

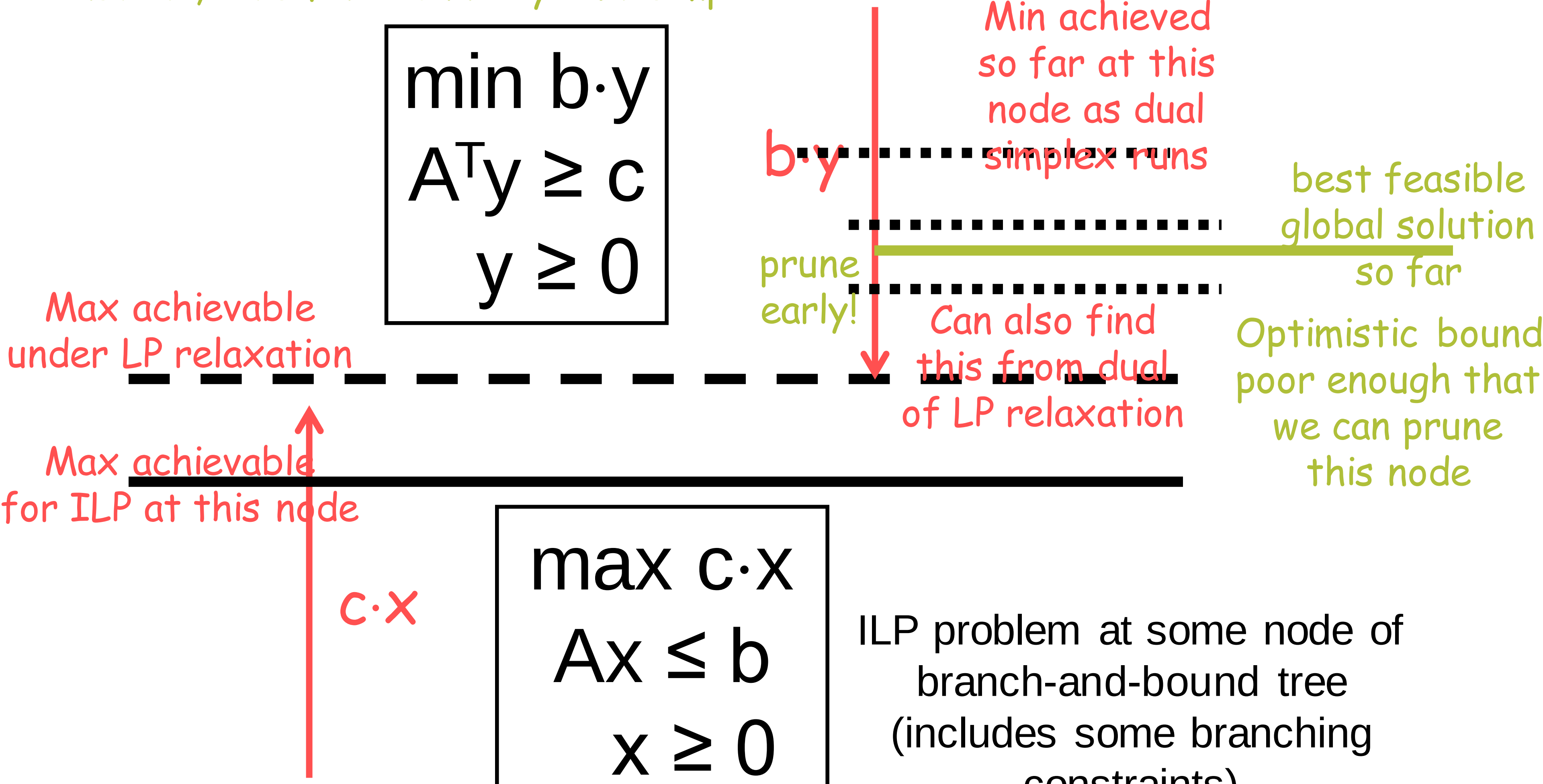
$$\boxed{x \cdot t + s \cdot y = 0}$$

- Complementary slackness: The basic vars in an optimal solution to one problem correspond to the non-basic vars in an optimal solution to the other problem.
- If a structural variable in one problem > 0, then the corresponding constraint in the other problem must be tight (its slack/surplus variable must be 0).
- And if a constraint in one problem is loose (slack/surplus var > 0), then the corresponding variable in the other problem must be 0. *(logically equiv. to above)*

Why duality is useful for ILP

Instead, let's find bound by **dual** simplex

$$\min\ b\cdot y \qquad A^T y \ge c \qquad y \ge 0$$



$$\max\ c\cdot x \qquad Ax \le b \qquad x \ge 0 \qquad x \text{ integer}$$

ILP problem at some node of branch-and-bound tree (includes some branching constraints)

Multiple perspectives on duality

Drop the names s and t now; use standard form, but call all the variables x and y.

1. As shown on earlier slide: The $y_i \geq 0$ are coefficients on a nonnegative linear combination of the primal constraints. Shows $c \cdot x \leq b \cdot y$, with equality iff complementary slackness holds.

2. Geometric interpretation of the above:
At a primal vertex x, cost hyperplane (shifted to go through the vertex) is a linear combination of the hyperplanes that intersect at that vertex. This is a <u>nonnegative</u> linear combination ($y \geq 0$, which is feasible in the dual) iff the cost hyperplane is tangent to the polytope at x (doesn't go through middle of polytope; technically, it's a subgradient at x), meaning that x is optimal.

3. "Shadow price" interpretation: Optimal y_i says how rapidly the primal optimum (max $c \cdot x$) would improve as we relax primal constraint i. (A derivative.) Justify this by Lagrange multipliers (next slide). It's 0 if primal constraint i has slack at primal optimum.

4. "Reduced cost" interpretation: Each $y_i \geq 0$ is the rate at which $c \cdot x$ would get worse if we phased x_i into the basis while preserving $Ax=b$. This shows that (for an optimal vertex x), if $x_i > 0$ then $y_i = 0$, and if $y_i > 0$ then $x_i = 0$. At non-optimal x, y is infeasible in dual.

Where Does Duality Come From?

- More generally, let's look at Lagrangian relaxation.

  max $c(x)$ subject to $a(x) \le b$ (let x^* denote the solution)

Technically, this is not the method of Lagrange multipliers.
Lagrange (18th century) only handled equality constraints.
Karush (1939) and Kuhn & Tucker (1951) generalized to inequalities.

Where Does Duality Come From?

- More generally, let's look at Lagrangian relaxation.
  max $c(x)$ subject to $a(x) \le b$ (let x^* denote the solution)
- Try ordinary constraint relaxation:
  max $c(x)$ (let x_0 denote the solution)

  If it happens that $a(x_0) \le b$, we're done! But what if not?
- Then try adding a surplus penalty if $a(x) > b$:
  max $c(x) - \lambda(a(x) - b)$ (let x_λ denote the solution)

  Lagrangian term (penalty rate λ is a "Lagrange multiplier")

  - Still an unconstrained optimization problem, yay! Solve by calculus, dynamic programming, etc. – whatever's appropriate for the form of this function. (c and a might be non-linear, x might be discrete, etc.)

Where Does Duality Come From?

- More generally, let's look at Lagrangian relaxation.
  max $c(x)$ subject to $a(x) \le b$ (let x^* denote the solution)
- Try ordinary constraint relaxation:
  max $c(x)$ (let x_0 denote the solution)

  If it happens that $a(x_0) \le b$, we're done! But what if not?
- Then try adding a surplus penalty if $a(x) > b$:
  max $c(x) - \lambda(a(x) – b)$ (let x_λ denote the solution)
  - If $a(x_\lambda) > b$, then increase penalty rate $\lambda \ge 0$ till constraint is satisfied.

Increasing λ gets solutions x_λ with $a(x_\lambda) = 100$, then 90, then 80 …
These are solutions to max $c(x)$ subject to $a(x) \le 100, 90, 80$ …
So λ is essentially an *indirect* way of controlling b.
Adjust it till we hit the b that we want.

Each y_i from LP dual acts like a λ (in fact y is λ upside down!)

Where Does Duality Come From?

- More generally, let's look at Lagrangian relaxation.
  max $c(x)$ subject to $a(x) \le b$ (let x^* denote the solution)
- Try ordinary constraint relaxation:
  max $c(x)$ (let x_0 denote the solution)

  If it happens that $a(x_0) \le b$, we're done! But what if not?
- Then try adding a surplus penalty if $a(x) > b$:
  max $c(x) - \lambda(a(x) - b)$ (let x_λ denote the solution)
  - If $a(x_\lambda) > b$, then increase penalty rate $\lambda \ge 0$ till constraint is satisfied.
  - **Important:** If $\lambda \ge 0$ gives $\mathbf{a(x_\lambda) = b}$, then x_λ is an *optimal* soln x^*.
    - Why? Suppose there were a better soln x' with $c(x') > c(x_\lambda)$ and $a(x') \le b$.
    - Then it would have beaten x_λ: $c(x') - \underbrace{\lambda(a(x') - b)}_{\substack{\text{Lagrangian is} \le 0,\\ \text{since by assumption}\\ a(x') \le b}} \ge c(x_\lambda) - \underbrace{\lambda(a(x_\lambda) - b)}_{\substack{\text{Lagrangian is } 0\\ \text{since by assumption}\\ a(x_\lambda) = b}}$
    - But no x' achieved this.

(In fact, Lagrangian actually *rewards* x' with $a(x') < b$. These x' didn't win despite this unfair advantage, because they did worse on c.)

Where Does Duality Come From?

- More generally, let's look at Lagrangian relaxation.
  max $c(x)$ subject to $a(x) \le b$ (let x^* denote the solution)
- Try ordinary constraint relaxation:
  max $c(x)$ (let x_0 denote the solution)

  If it happens that $a(x_0) \le b$, we're done! But what if not?
- Then try adding a surplus penalty if $a(x) > b$:
  max $c(x) - \lambda(a(x) - b)$ (let x_λ denote the solution)
  - If $a(x_\lambda) > b$, then increase penalty rate $\lambda \ge 0$ till constraint is satisfied.
  - **Important:** If $\lambda \ge 0$ gives $\mathbf{a(x_\lambda) = b}$, then x_λ is an *optimal* soln x^*.
    - Why? Suppose there were a better soln x' with $c(x') > c(x_\lambda)$ and $a(x') \le b$. Then it would have beaten x_λ: $c(x') - \lambda(a(x') - b) \ge c(x_\lambda) - \lambda(a(x_\lambda) - b)$
  - If λ is too **small** (constraint is "too relaxed"): **infeasible solution**. $a(x_\lambda) > b$ still, and $c(x_\lambda) \ge c(x^*)$. Upper bound on true answer (prove it!).
  - If λ is too **large** (constraint is "overenforced"): **suboptimal solution**. $a(x_\lambda) < b$ now, and $c(x_\lambda) \le c(x^*)$. Lower bound on true answer.

Tightest upper bound: min $c(x_\lambda)$ subject to $a(x_\lambda) \ge b$. See where this is going?

Where Does Duality Come From?

- More generally, let's look at Lagrangian relaxation.
  max $c(x)$ subject to $a(x) \le b$ (let x^* denote the solution)
- Try ordinary constraint relaxation:
  max $c(x)$ (let x_0 denote the solution)
  If it happens that $f(x_0) \le c$, we're done! But what if not?
- Then try adding a slack penalty if $g(x) > c$:
  max $c(x) - \underbrace{\lambda(a(x) - b)}_{\text{Lagrangian}}$ (let x_λ denote the solution)
- **Complementary slackness:** "We found x_λ with Lagrangian=0."
  - That is, either $\lambda=0$ or $a(x_\lambda)=b$.
  - Remember, $\lambda=0$ may already find x_0 with $a(x_0) \le b$. Then x_0 optimal.
  - Otherwise we increase $\lambda > 0$ until $a(x_\lambda)=b$, we hope. Then x_λ optimal.
  - Is complementary slackness necessary for x_λ to be an optimum?
    - Yes if $c(x)$ and $a(x)$ are linear, or satisfy other "regularity conditions."
    - No for integer programming. $a(x)=b$ may be unachievable, so the soft problem only gives us upper and lower bounds.

Where Does Duality Come From?

- More generally, let's look at Lagrangian relaxation.
  max c(x) subject to a(x) ≤ b (let x* denote the solution)
- Try ordinary constraint relaxation:
  max c(x) (let x_0 denote the solution)
  If it happens that $f(x_0) \le c$, we're done! But what if not?
- Then try adding a slack penalty if g(x) > c :
  $\max c(x) - \underbrace{\lambda(a(x) - b)}_{\text{Lagrangian}}$ (let x_λ denote the solution)

- **Can we always find a solution just by unconstrained optimization?**
  - No, not even for linear programming case. We'll still need simplex method.
  - Consider this example: max x subject to x ≤ 3. Answer is x*=3.
    - But max x - λ(x-3) gives $x_\lambda = -\infty$ for λ > 1 and $x_\lambda = \infty$ for λ < 1.
    - λ=1 gives a huge tie, where some solutions x_λ satisfy constraint and others don't.

Where Does Duality Come From?

- More generally, let's look at Lagrangian relaxation.
  max $c(x)$ subject to $a(x) \le b$ (let x^* denote the solution)
- Try ordinary constraint relaxation:
  max $c(x)$ (let x_0 denote the solution)
  If it happens that $f(x_0) \le c$, we're done! But what if not?
- Then try adding a slack penalty if $g(x) > c$:
  $\max c(x) - \underbrace{\lambda(a(x) - b)}_{\text{Lagrangian}}$ (let x_λ denote the solution)
- **How about multiple constraints?**
  max $c(x)$ subject to $a_1(x) \le b_1$, $a_2(x) \le b_2$
  - Use several Lagrangians:
    $\max c(x) - \lambda_1(a_1(x) - b_1) - \lambda_2(a_2(x) - b_2)$
  - Or in vector notation:
    $\max c(x) - \lambda \cdot (a(x) - b)$ where λ, $a(x)$, b are vectors