Solvers for Mixed Integer Programming
Relaxation: A general optimization technique

- **Want:**
  - $x^* = \arg\min f(x)$ subject to $x \in S$
  - $S$ is the feasible set

- **Start by getting:**
  - $x_1 = \arg\min f(x)$ 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:**
  - \( x^* = \min f(x) \) subject to \( x \in S \)
  - \( S \) is the feasible set

- **Start by getting:**
  - \( x_1 = \min f(x) \) 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 number of constraints or difficult ones
  
- Else, add some constraints back (to shrink \( T \)) and try again

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

**But how can we add integrality constraints back?** (simplex relies on having dropped them all)
Rounding doesn’t work

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

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

image adapted from Jop Sibeyn
Cutting planes: add new \textbf{linear} constraints

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

\textbf{Figure 14–2} Illustration of a cutting-plane algorithm. (a) The continuous optimum $X$. (b) The new $X$ after one cut. (c) The solution of the original ILP after two cuts.
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

\[
x_1 + 4x_2 + x_3 \geq 10 \\
4x_1 + 2x_2 + 2x_3 \geq 13 \\
x_1 + x_2 - x_3 \geq 0 \\
x_1, x_2, x_3 \geq 0 \text{ and integer.}
\]

\[
x_1 + 4x_2 + x_3 \geq 10 \\
x_1 + 3x_2 + x_3 \geq 9 \\
2x_1 + 4x_2 + x_3 \geq 13 \\
x_1 + x_2 + x_3 \geq 5 \\
2x_1 + x_2 + x_3 \geq 7 \\
x_1 + 2x_2 \geq 5 \\
2x_1 + x_2 \geq 4 \\
x_1 + x_2 - x_3 \geq 0 \\
x_1, x_2, x_3 \geq 0.
\]

No integrality constraints!
But optimal solution is the same.

- How can we find these new constraints??

example from H. P. Williams
Chvátal cuts

- 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

---

![Diagram](image)

- Decreasing cost
- $x_1$ (a)
- $x$ (b)
- $x_3 = x_0$ (c)

*figure adapted from Papadimitriou & Steiglitz*
Branch and bound: Disjunctive cutting planes!

Minimise \( 2x_1 + 7x_2 + 2x_3 \)

\[
\begin{align*}
x_1 + 4x_2 + x_3 & \geq 10 \\
4x_1 + 2x_2 + 2x_3 & \geq 13 \\
x_1 + x_2 - x_3 & \geq 0
\end{align*}
\]

\( x_1, x_2, x_3 \geq 0 \) and integer.

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

When does solving the relaxed problem ensure integral \( x_2 \)?

figure from
H. P. Williams
Remember branch-and-bound from constraint programming?

figure thanks to Tobias Achterberg
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:** Minimization problem instance \( R \).

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

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

Can insert a stochastic local search here to try to find a feasible solution \( x^\wedge \) near \( x^v \), to improve upper bound \( c^\wedge \) further.

Pseudocode thanks to Tobias Achterberg.

May simplify ("presolve") it first.

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

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.").
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 $\tilde{x}$ from the convex hull $Q_I$ of integer points of $Q$.
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!
**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 \((\text{min } c \cdot x \text{ subj. to } Ax \leq b, x \geq 0)\)
  - 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 log is
    - \(O(m^2 \log m \log (\text{biggest integer in } A \text{ or } b))\) \([\text{for LP}]\)

**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 \(\log\) is
  - \(O(m^2 \log m \log (\text{biggest integer in } A \text{ or } b))\) [for LP]
  - \(O(\log n + m(\log n + \log (\text{biggest int in } A, b, \text{ or } c))\) [for ILP]

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≥0)
- Where all numbers in A,b,c are integers; n vars, m constraints
- If there’s a finite optimum x, each xᵢ is ≤ a bound whose log 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 ≤ x₅ ≤ 2¹⁰⁰
- 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 x₅ + … ?
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 LP polytope are integral anyway.
  - So regardless of the cost function, the LP 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.)

$$A\vec{x} \leq \vec{b} \quad \text{(or} \quad A\vec{x} = \vec{b})$$
Totally Unimodular Cost Matrix $A$

- A square matrix with determinant +1 or -1 is called unitary.
- A unitary integer matrix is called unimodular. Its inverse is integral too!
  - (follows easily from $A^{-1} = \text{adjoint}(A) / \det(A)$)
  - 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 ($\det=\pm1$) 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

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.)

---

**Example**:

\[
\begin{array}{c|cccc|c|c|c|c}
& x_{1,A} & x_{1,B} & x_{II,B} & x_{III,C} & x_{IV,B} & x_{IV,C} \\
(i=I) & 1 & 1 & 0 & 0 & 0 & 0 \\
(i=II) & 0 & 0 & 1 & 0 & 0 & 0 \\
(i=III) & 0 & 0 & 0 & 1 & 0 & 0 \\
(i=IV) & 0 & 0 & 0 & 0 & 1 & 1 \\
(j=A) & 1 & 0 & 0 & 0 & 0 & 0 \\
(j=B) & 0 & 1 & 1 & 0 & 1 & 0 \\
(j=C) & 0 & 0 & 0 & 1 & 0 & 1 \\
\end{array}
\]

---

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

\[
\begin{align*}
\text{min} & \quad \sum_{ij} c_{ij} x_{ij} \quad \text{with } x_{ij} \text{ binary} \\
\text{subjto} & \quad \sum_j x_{sj} = 1, \sum_j x_{jt} = 1 \\
& \quad (\forall j \notin \{s, t\}) \sum_i x_{ij} = \sum_k x_{jk}
\end{align*}
\]

Can formulate as \( Ax = b, x \geq 0 \) so that A matrix is totally unimodular:

<table>
<thead>
<tr>
<th></th>
<th>( x_{sA} )</th>
<th>( x_{sC} )</th>
<th>( x_{AB} )</th>
<th>( x_{BC} )</th>
<th>( x_{CA} )</th>
<th>( x_{Bt} )</th>
<th>( x_{Ct} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s)</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(j=A)</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(j=B)</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>(j=C)</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(t)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

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

  \[
  \begin{align*}
  \text{min} & \sum_{ij} c_{ij} x_{ij} \quad \text{with } x_{ij} \text{ binary} \\
  \text{subjto} & \sum_j x_{sj} = 1, \sum_j -x_{jt} = -1 \\
  & (\forall j \notin \{s, t\}) \sum_i -x_{ij} + \sum_k x_{jk} = 0
  \end{align*}
  \]

Can formulate as $Ax = b$, $x \geq 0$ 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.”)
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

- \( \text{min } c \cdot x \) subject to \( Ax \leq b, \ x \geq 0 \)
- \( m \) 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.
  - 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!

---

example adapted from Ofer Strichman
Remember variable elimination for SAT?

**Davis-Putnam**

- This procedure (resolution) eliminates all copies of X \textbf{and} \sim X.
  - We’re done in n steps. \textit{So what goes wrong?}
  - Size of formula can square at each step.

Resolution fuses each pair \((V \lor W \lor \sim X) \land (X \lor Y \lor Z)\) into \((V \lor W \lor Y \lor Z)\)

Justification #1: Valid way to eliminate X (reverses CNF \rightarrow 3-CNFS idea).
Justification #2: Want to recurse on a CNF version of \(((\phi \land X) \lor (\phi \land \sim X))\)

Suppose \(\phi = \alpha \land \beta \land \gamma\)

where \(\alpha\) is clauses with \sim X, \(\beta\) with X, \(\gamma\) with neither
Then \(((\phi \land X) \lor (\phi \land \sim X)) = (\alpha' \land \gamma) \lor (\beta' \land \gamma)\) by unit propagation

where \(\alpha'\) is \(\alpha\) with the \sim X’s removed, \(\beta'\) similarly.

\(= (\alpha' \lor \beta') \land \gamma = (\alpha'_1 \lor \beta'_1) \land (\alpha'_1 \lor \beta'_2) \land \ldots \land (\alpha'_{99} \lor \beta'_{99}) \land \gamma\)
Fourier-Motzkin elimination

- Variable elimination on a set of inequalities

- minimize

- subject to

  \[
  z - y \leq 0 \\
  z - x \leq 0 \\
  2x + y - z \leq 0 \\
  -x \leq 2
  \]

example adapted from Ofer Strichman
Fourier-Motzkin elimination

- Variable elimination on a set of inequalities
  - [blackboard example]

- To eliminate variable $z$, first take each inequality involving $z$ and solve it for $z$. This gives $z \leq \alpha_1$, $z \leq \alpha_2$, … , $z \geq \beta_1$, $z \geq \beta_2$, …
  - The $\alpha_i$ and $\beta_j$ are linear functions of the other vars a,b,…,y

- Replace these inequalities by $\alpha_1 \geq \beta_1$, $\alpha_1 \geq \beta_2$, $\alpha_2 \geq \beta_1$, $\alpha_2 \geq \beta_2$, …
  - Equivalently, $\min \alpha \geq \max \beta$. 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!
    Impractical since, just like resolution, may square the # of constraints.

- Repeat to eliminate variable $y$, etc.

- If one of our equations is “$a = [\text{linear cost function}]$,” then at the end, our only inequalities are lower and upper bounds on $a$.
  - Now it’s easy to min or max $a$! Then back-solve to get $b$, $c$, … $z$ in turn.
From Canonical to Standard Form

- \( \min c \cdot x \) subject to \( Ax \leq b, \ x \geq 0 \)
- \( m \) 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.)

\[ Ax = b \]
From Canonical to Standard Form

- \( \min_c \cdot x \) subject to \( Ax = b, \ x \geq 0 \)
- \( m \) constraints (rows)
- \( n+m \) variables (columns)

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 \( \min c \cdot x \).)

We are looking to express \( b \) as a linear combination of \( A \)'s columns. \( x \) gives the coefficients of this linear combination.
**Standard Form**

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

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 \( \min c \cdot x \).)

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.
Standard Form

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

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

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

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

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 \( \min c \cdot x \).)

Notice that the bfs in the picture is optimal when the cost vector is \( c = (1,1,1,0,0,0,0,...) \). Similarly, any bfs is optimal for some cost vector. Hmm, sounds like polytope vertices...
Canonical vs. Standard Form

**Ax ≤ b**
- x ≥ 0
- m inequalities
- + n inequalities (n variables)

**Ax = b**
- x ≥ 0
- m equalities
- + n+m inequalities (n+m variables)

Add m slack variables (one per constraint)

Eliminate last m vars (how?)

Eliminating last m vars turns the last m “≥ 0” constraints & the m constraints (“Ax=b”) into m inequalities (“Ax ≤ b”).

E.g., have 2 constraints on $x_{n+m}$:
- $x_{n+m} ≥ 0$
- The last row, namely $(h_1x_1 + ... + h_nx_n) + x_{n+m} = b$.

To elim $x_n$, replace them with $(h_1x_1 + ... + h_nx_n) ≤ b$.

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

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

Now can eliminate slack vars.
**Canonical vs. Standard Form**

Ax ≤ b

x ≥ 0

m inequalities
+n inequalities (n variables)

---

Ax = b

x ≥ 0

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

---

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

---

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

---

add m slack variables (one per constraint)

---

Eliminate last m vars

---

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

### Geometric interpretation
- Move to an adjacent vertex
  (n facets define the vertex: change 1) on edge)

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

---

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

\[
\begin{align*}
  x &= (5, 4, 7, 0, 0, 0) & 5C_1 + 4C_2 + 7C_3 &= b \\
  x &= (4.9, 3.8, 7.1, 0, 0.1, 0) & \ldots \\
  x &= (4.8, 3.6, 7.2, 0, 0.2, 0) & \ldots \\
  x &= (5-\varepsilon, 4-2\varepsilon, 7+\varepsilon, 0, \varepsilon, 0) & \ldots \\
  x &= (3, 0, 9, 0, 2, 0) & 3C_1 + 9C_3 + 2C_5 &= b
\end{align*}
\]
is the new bfs
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 put a different vertex 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 ≥ 0
m inequalities
+ n inequalities (n variables)

Ax = b
x ≥ 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 equalities to be tight

bfs
(defined by selecting n of the variables to be 0)
Simplex algorithm: More discussion

- How do we pick which column to phase in (i.e., which adjacent vertex to move to)?
  - 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 any 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 (incompatible constraints, so quit and return UNSAT).
    - Or the m rows of A aren’t linearly independent (redundant constraints, so throw away the extras & try again).

---

**vertex**
(defined by intersecting n-m of the constraints, each of which reduces dimensionality by 1)

**bfs**
(defined by selecting n-m of the variables to be 0)

Pick n-m of the m equalities to be tight
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
Duality for Linear Programs (canonical form)

Primal problem
\[
\begin{align*}
\text{max} & \quad c \cdot x \\
Ax & \leq b \\
x & \geq 0
\end{align*}
\]

Dual problem
\[
\begin{align*}
\text{min} & \quad b \cdot y \\
A^T y & \geq c \\
y & \geq 0
\end{align*}
\]

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 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 \(\leq\) constraints, this time to get an upper bound on the objective.
  - As before, there are lots of linear combinations.
  - Different linear combinations \(\Rightarrow\) different upper bounds.
  - Smaller (tighter) upper bounds are more useful.
  - Our \textit{smallest} upper bound might be tight and equal \(\max c \cdot x\).
Where Does Duality Come From?

- As warmup, let’s look at Lagrangian relaxation.
  \[
  \max c(x) \quad \text{subject to } a(x) \leq b \quad (\text{let } x^* \text{ denote the solution})
  \]

  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?

- As warmup, let’s look at Lagrangian relaxation.
  \[
  \text{max } c(x) \quad \text{subject to } a(x) \leq b \quad (\text{let } x^* \text{ denote the solution})
  \]

- Try ordinary constraint relaxation:
  \[
  \text{max } c(x) \quad (\text{let } x_0 \text{ denote the solution})
  \]
  If it happens that \(a(x_0) \leq b\), we’re done! But what if not?

- Then try adding a surplus penalty if \(a(x) > b\) :
  \[
  \text{max } c(x) - \lambda(a(x) - b) \quad (\text{let } x_\lambda \text{ denote the solution})
  \]
  Lagrangian term (penalty rate \(\lambda\) 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?

- As warmup, let’s look at Lagrangian relaxation.
  \[ \max c(x) \quad \text{subject to} \quad a(x) \leq b \quad (\text{let } x^* \text{ denote the solution}) \]

- Try ordinary constraint relaxation:
  \[ \max c(x) \quad (\text{let } x_0 \text{ denote the solution}) \]
  If it happens that \( a(x_0) \leq b \), we’re done! But what if not?

- Then try adding a surplus penalty if \( a(x) > b \):
  \[ \max c(x) - \lambda(a(x) - b) \quad (\text{let } x_\lambda \text{ denote the solution}) \]
  - If \( a(x_\lambda) > b \), then increase penalty rate \( \lambda \geq 0 \) till constraint is satisfied.

Increasing \( \lambda \) gets solutions \( x_\lambda \) with \( a(x_\lambda) = 100 \), then 90, then 80 ...
These are solutions to \( \max c(x) \) with \( a(x) \leq 100, 90, 80 \) ...
So \( \lambda \) is essentially an indirect way of controlling \( b \).
Adjust it till we hit the \( b \) that we want.
Where Does Duality Come From?

- As warmup, let’s look at Lagrangian relaxation.
  \[
  \max c(x) \quad \text{subject to} \quad a(x) \leq b \quad (\text{let } x^* \text{ denote the solution})
  \]

- Try ordinary constraint relaxation:
  \[
  \max c(x) \quad (\text{let } x_0 \text{ denote the solution})
  \]
  If it happens that \( a(x_0) \leq b \), we’re done! But what if not?

- Then try adding a surplus penalty if \( a(x) > b \) :
  \[
  \max c(x) - \lambda(a(x) - b) \quad (\text{let } x_\lambda \text{ denote the solution})
  \]
  - If \( a(x_\lambda) > b \), then increase penalty rate \( \lambda \geq 0 \) till constraint is satisfied.
  - **Important:** If \( \lambda \geq 0 \) gives \( a(x_\lambda) = b \), then \( x_\lambda \) is an *optimal* soln \( x^* \).
    - Why? Suppose there were a better soln \( x' \) with \( c(x') > c(x_\lambda) \) and \( a(x') \leq b \).
    - Then it would have beaten \( x_\lambda \) :
      \[
      c(x') - \lambda(a(x') - b) \geq c(x_\lambda) - \lambda(a(x_\lambda) - b)
      \]
    - But no \( x' \) achieved this.

  \begin{align*}
  \text{Lagrangian is } &\leq 0, \\
  \text{since by assumption} &\quad a(x') \leq b \\
  \text{Lagrangian is } &\geq 0, \\
  \text{since by assumption} &\quad a(x_\lambda) = b
  \end{align*}

(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?

- As warmup, let’s look at Lagrangian relaxation.
  \[
  \text{max } c(x) \text{ subject to } a(x) \leq b \quad (\text{let } x^* \text{ denote the solution})
  \]

- Try ordinary constraint relaxation:
  \[
  \text{max } c(x) \quad (\text{let } x_0 \text{ denote the solution})
  \]
  If it happens that \( a(x_0) \leq b \), we’re done! But what if not?

- Then try adding a surplus penalty if \( a(x) > b \):
  \[
  \text{max } c(x) - \lambda(a(x) - b) \quad (\text{let } x_\lambda \text{ denote the solution})
  \]
  - If \( a(x_\lambda) > b \), then increase penalty rate \( \lambda \geq 0 \) till constraint is satisfied.
  - **Important:** If \( \lambda \geq 0 \) gives \( a(x_\lambda) = b \), then \( x_\lambda \) is an optimal soln \( x^* \).
    - Why? Suppose there were a better soln \( x' \) with \( c(x') > c(x_\lambda) \) and \( a(x') \leq b \).
      Then it would have beaten \( x_\lambda \):
      \[
      c(x') - \lambda(a(x') - b) \geq c(x_\lambda) - \lambda(a(x_\lambda) - b)
      \]
  - If \( \lambda \) is too small (constraint is “too relaxed”): infeasible solution.
    \( a(x_\lambda) > b \) still, and \( c(x_\lambda) \geq c(x^*) \). Upper bound on true answer (prove it!).
  - If \( \lambda \) is too large (constraint is “overenforced”): suboptimal solution.
    \( a(x_\lambda) < b \) now, and \( c(x_\lambda) \leq c(x^*) \). Lower bound on true answer.

**Tightest upper bound:** \( \min c(x_\lambda) \text{ subject to } a(x_\lambda) \geq b \). See where this is going?
Where Does Duality Come From?

- As warmup, let’s look at Lagrangian relaxation.
  \[ \max c(x) \text{ subject to } a(x) \leq b \quad (\text{let } x^* \text{ denote the solution}) \]

- Try ordinary constraint relaxation:
  \[ \max c(x) \quad (\text{let } x_0 \text{ denote the solution}) \]
  If it happens that \( f(x_0) \leq c \), we’re done! But what if not?

- Then try adding a slack penalty if \( g(x) > c \):
  \[ \max c(x) - \lambda(a(x) - b) \quad (\text{let } x_{\lambda} \text{ denote the solution}) \]

- **Complementary slackness:** “We found \( x_{\lambda} \) with Lagrangian=0.”
  - That is, either \( \lambda=0 \) or \( a(x_{\lambda})=b \).
  - Remember, \( \lambda=0 \) may already find \( x_0 \) with \( a(x_0) \leq b \). Then \( x_0 \) optimal.
  - Otherwise we increase \( \lambda > 0 \) until \( a(x_{\lambda})=b \), we hope. Then \( x_{\lambda} \) optimal.
  - Is complementary slackness necessary for \( x_{\lambda} \) 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?

- As warmup, let’s look at Lagrangian relaxation.
  \[ \max c(x) \text{ subject to } a(x) \leq b \quad \text{(let } x^* \text{ denote the solution)} \]

- Try ordinary constraint relaxation:
  \[ \max c(x) \quad \text{(let } x_0 \text{ denote the solution)} \]
  If it happens that \( f(x_0) \leq c \), we’re done! But what if not?

- Then try adding a slack penalty if \( g(x) > c \):
  \[ \max c(x) - \lambda(a(x) - b) \quad \text{(let } x_\lambda \text{ denote the solution)} \]

- 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: \( \max x \text{ subject to } x \leq 3 \). Answer is \( x^* = 3 \).
    - But \( \max x - \lambda(x-3) \) gives \( x_\lambda = \infty \) for \( \lambda > 1 \) and \( x_\lambda = \infty \) for \( \lambda < 1 \).
    - \( \lambda = 0 \) gives a huge tie, where some solutions \( x_\lambda \) satisfy constraint and others don’t.
Where Does Duality Come From?

- As warmup, let’s look at Lagrangian relaxation.
  \[
  \max c(x) \quad \text{subject to } a(x) \leq b \quad (\text{let } x^* \text{ denote the solution})
  \]

- Try ordinary constraint relaxation:
  \[
  \max c(x) \quad (\text{let } x_0 \text{ denote the solution})
  \]
  If it happens that \( f(x_0) \leq c \), we’re done! But what if not?

- Then try adding a slack penalty if \( g(x) > c \) :
  \[
  \max c(x) - \lambda (a(x) - b) \quad (\text{let } x_\lambda \text{ denote the solution})
  \]

- How about multiple constraints?
  \[
  \max c(x) \quad \text{subject to } a_1(x) \leq b_1, \ a_2(x) \leq b_2
  \]
  - Use several Lagrangians:
    \[
    \max c(x) - \lambda_1 (a_1(x) - b_1) - \lambda_2 (a_2(x) - b_2)
    \]
  - Or in vector notation:
    \[
    \max c(x) - \lambda \cdot (a(x) - b) \quad \text{where } \lambda, \ a(x), \ b \text{ are vectors}
    \]
Where Does Duality Come From?

- Back to linear programming. Let’s take linear combinations of the $\leq$ constraints, to get various upper bounds on the objective.

- $\max \quad 2x_1 + 3x_2 \quad \text{subject to } \begin{align*}
x_1, x_2 &\geq 0 \\
C_1: \quad &x_1 + x_2 \leq 12 \\
C_2: \quad &2x_1 + x_2 \leq 9 \\
C_3: \quad &x_1 \leq 4 \\
C_4: \quad &x_1 + 2x_2 \leq 10
\end{align*}$

- Objective: $2x_1 + 3x_2 \leq 2x_1 + 4x_2 \leq 20 \quad (2*C_4)$
- Objective: $2x_1 + 3x_2 \leq 2x_1 + 3x_2 \leq 22 \quad (1*C_1 + 1*C_4)$
- Objective: $2x_1 + 3x_2 \leq 3x_1 + 3x_2 \leq 19 \quad (1*C_2 + 1*C_4)$

example from Rico Zenklusen

positive coefficients so $\leq$ doesn't flip
Where Does Duality Come From?

- Back to linear programming. Let’s take linear combinations of the \( \leq \) constraints, to get various upper bounds on the objective.

\[
\begin{align*}
\text{max} & \quad 2x_1 + 3x_2 \\
\text{subject to} & \quad x_1, x_2 \geq 0 \quad \text{and} \\
C_1 & : \quad x_1 + x_2 \leq 12 \\
C_2 & : \quad 2x_1 + x_2 \leq 9 \\
C_3 & : \quad x_1 \leq 4 \\
C_4 & : \quad x_1 + 2x_2 \leq 10
\end{align*}
\]

- \((y_1 + 2y_2 + y_3 + y_4)x_1 + (y_1 + y_2 + 2y_4)x_2 \leq 12y_1 + 9y_2 + 4y_3 + 10y_4\)

- 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

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
Duality for Linear Programs (canonical form)

- The form above assumes $(\text{max}, \leq) \Leftrightarrow (\text{min}, \geq)$.
- Extensions for LPs in general form:
  - Any reverse constraints ($(\text{max}, \geq)$ or $(\text{min}, \leq)$) $\Leftrightarrow$ negative vars
  - So, any equality constraints $\Leftrightarrow$ unbounded vars (can simulate with pair of constraints $\Leftrightarrow$ pair of vars)
- Also, degenerate solution ($\# \text{ tight constraints} > \# \text{ vars}$) $\Leftrightarrow$ 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

\[
\begin{align*}
\text{max} & \quad c \cdot x \\
Ax & \leq b \\
x & \geq 0
\end{align*}
\]

Just negate A, b, and c

Dual problem

\[
\begin{align*}
\text{min} & \quad b \cdot y \\
A^T y & \geq c \\
y & \geq 0
\end{align*}
\]

Equivalent to primal

\[
\begin{align*}
\text{min} & \quad (-c) \cdot x \\
(-A)x & \geq -b \\
x & \geq 0
\end{align*}
\]

Equivalent to dual

\[
\begin{align*}
\text{max} & \quad (-b) \cdot y \\
(-A^T)y & \leq (-c) \\
y & \geq 0
\end{align*}
\]
We’ve seen that for any feasible solutions \( x \) and \( y \), \( c \cdot x \leq b \cdot y \).

- \( b \cdot y \) 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”

**Primal**
- Max achievable under primal constraints
- \( c \cdot x \)
- \( Ax \leq b \)
- \( x \geq 0 \)

**Dual**
- Min achievable under dual constraints
- \( b \cdot y \)
- \( A^T y \geq c \)
- \( y > 0 \)

Primal: \( c \cdot x \leq b \cdot y \) 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

\[
\begin{align*}
\text{max } & \mathbf{c} \cdot \mathbf{x} \\
\text{Ax + s = b} & \\
\mathbf{x} \geq 0 & \\
\mathbf{s} \geq 0 &
\end{align*}
\]

(m slack vars)

(n struct vars)

Dual problem

\[
\begin{align*}
\text{min } & \mathbf{b} \cdot \mathbf{y} \\
\mathbf{A}^\mathsf{T} \mathbf{y} - \mathbf{t} = \mathbf{c} & \\
\mathbf{y} \geq 0 & \\
\mathbf{t} \geq 0 &
\end{align*}
\]

(n surplus vars)

(m struct 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 \( \geq 0 \). The \( n \) non-basic vars are 0.

- At dual optimality:
  - Some \( n \) “basic” vars of dual can be \( \geq 0 \). The \( m \) non-basic vars are 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.
  - So, 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. \((\text{logically equiv. to above})\)
Why duality is useful for ILP

Instead, let's find bound by dual simplex

\[
\begin{align*}
\text{min} & \quad b \cdot y \\
A^T y & \geq c \\
y & \geq 0
\end{align*}
\]

Max achievable under LP relaxation

Can also find this from dual of LP relaxation

Max achievable for ILP at this node

Can also find this from dual of LP relaxation

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

Min achieved so far at this node as dual simplex runs

Best feasible global solution so far

Optimistic bound poor enough that we can prune this node

prune early!
Multiple perspectives on duality

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

1. The \( y_i \geq 0 \) are coefficients on a 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 nonnegative 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; 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.

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.