COMP 382: Reasoning about Algorithms

Linear Programming & Duality

Prof. Maryam Aliakbarpour

co-instructors: Prof. Anjum Chida & Prof. Konstantinos Mamouras

November 13, 2025

Today's Lecture

1. **Linear Programming**

2. **What Is NP-Hardness?**

Reading:

- Lecture note in [Goemans, 2015]
- Lecture note in [Trevisan, 2011]
- Chapter 19 of [Roughgarden, 2022]

Content adapted from the same references.

Linear Programming

Problems with Linear Constraints

- Making the best choice under limits (budget, time, capacity).
- When relationships are *linear*, we get **Linear Programming (LP)**.
- LP appears in scheduling, transport, game theory, and machine learning.

Next: real-life examples

The Diet Problem

- We must plan a daily diet using two grains: G_1 and G_2.
- Each grain provides *carb, protein, and vitamins*, and has a cost per kg.
- Goal: meet daily nutritional requirements **at minimum cost**.

	Carb	Protein	Vitamins	Cost ($/oz)
G_1	5	4	2	0.60
G_2	7	2	1	0.35

Requirements per day: 8 units carb, 15 units protein, 3 units vitamins.

The Diet Problem

Variables (amount/day): $x_1 \leftarrow$ amount of G_1, $x_2 \leftarrow$ amount of G_2

$$\min \ 0.6x_1 + 0.35x_2$$

$$\begin{aligned} 5x_1 + 7x_2 &\geq 8 && \text{(carb)} \\ 4x_1 + 2x_2 &\geq 15 && \text{(protein)} \\ 2x_1 + x_2 &\geq 3 && \text{(vitamins)} \\ x_1, x_2 &\geq 0 \end{aligned}$$

Interpretation: pick amounts to meet each need as cheaply as possible.

The Transportation Problem

Two factories F_1, F_2 and three cities C_1, C_2, C_3.

	C_1	C_2	C_3	Supply
F_1	5	5	3	6
F_2	6	4	1	9
Demand	8	5	2	

Minimize total cost subject to all supplies and demands being met.

The Transportation Problem

Decision variables: x_{ij} = thousands of widgets shipped from F_i to C_j.

$$
\begin{aligned}
\min 5x_{11} + 5x_{12} + 3x_{13} + 6x_{21} + 4x_{22} + x_{23} & \\
x_{11} + x_{21} = 8 & \qquad \text{(demand } C_1\text{)} \\
x_{12} + x_{22} = 5 & \qquad \text{(demand } C_2\text{)} \\
x_{13} + x_{23} = 2 & \qquad \text{(demand } C_3\text{)} \\
x_{11} + x_{12} + x_{13} = 6 & \qquad \text{(supply } F_1\text{)} \\
x_{21} + x_{22} + x_{23} = 9 & \qquad \text{(supply } F_2\text{)} \\
x_{ij} \geq 0 & \qquad \text{(no negative shipments)}
\end{aligned}
$$

Interpretation: ship goods to meet all demands at minimum total cost.

What is Linear Programming?

Definition

A **linear program (LP)** optimizes a linear function subject to a set of linear equality or inequality constraints.

- We can always rewrite any LP in a **canonical form**.
- Geometry: intersection of half-spaces (a polyhedron).
- Algorithms: solved efficiently (e.g., *Simplex method*).

From real problems to canonical form

Linear programs can look very different:

$$\min 2x_1 - x_2 \quad \text{s.t.} \quad \begin{cases} x_1 + x_2 \geq 2, \\ 3x_1 + 2x_2 \leq 4, \\ x_1 + 2x_2 = 3, \\ x_1 \text{ free}, \ x_2 \geq 0. \end{cases}$$

From real problems to canonical form

Linear programs can look very different:

$$\min 2x_1 - x_2 \quad \text{s.t.} \quad \begin{cases} x_1 + x_2 \geq 2, \\ 3x_1 + 2x_2 \leq 4, \\ x_1 + 2x_2 = 3, \\ x_1 \text{ free}, \ x_2 \geq 0. \end{cases}$$

To solve any LP systematically or design algorithms for them, we need to convert it into a unified template...

Canonical Form

$$\max \; c^\top x \quad \text{s.t. } Ax \leq b, \; x \geq 0$$

- x: decision variables
- c: objective coefficients
- A: constraint matrix, b: resource limits

Every LP can be written in this form by adding slack variables or sign changes.

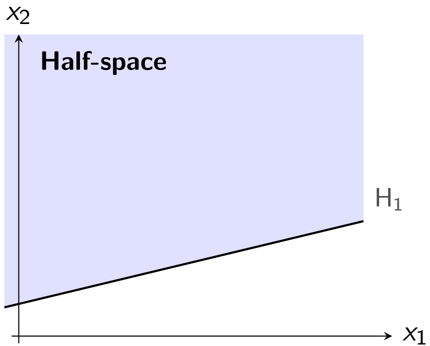
Feasibility Region: From half-spaces to polygons

Step 1. Half-space.

One inequality defines a line and the side that satisfies it.

$$\frac{x_1}{3} - x_2 \leq -1$$

Feasible set: *half-space*.

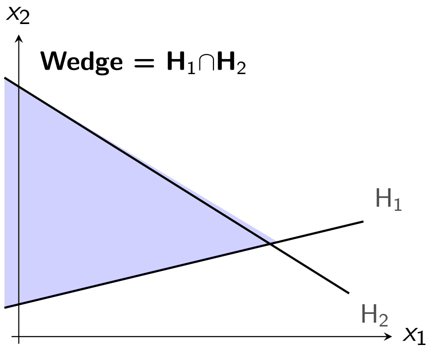


Feasibility Region: From half-spaces to polygons

Step 2. Wedge.

Two inequalities ⇒ intersection of two half-spaces.

Feasible set: *wedge* (two half-spaces).

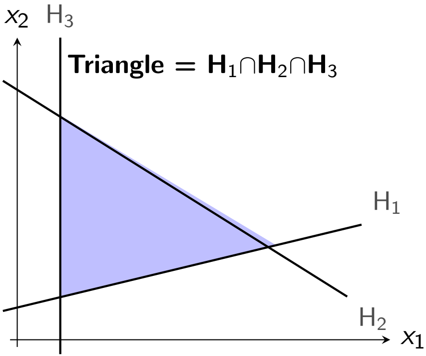


Feasibility Region: From half-spaces to polygons

Step 3. Triangle.

A third inequality can bound the region in 2D.

Feasible set: *triangle* (bounded).

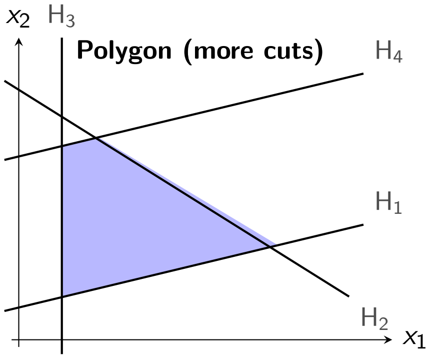


Feasibility Region: From half-spaces to polygons

Step 4. Polygon.

Additional constraints cut off corners
⇒ refined feasible set.

Feasible set: *polygon*.

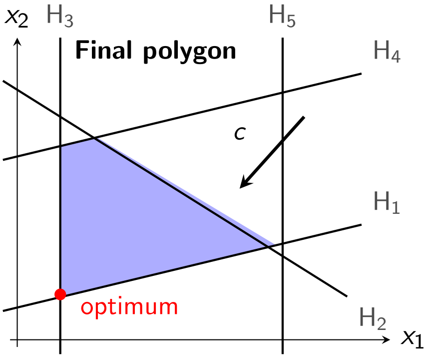


Feasibility Region: From half-spaces to polygons

Step 5. Optimum at a vertex.

Maximizing $c^\top x$ pushes along c to (usually) a vertex of the polygon.

Feasible set: *polygon*;



Simplex Method

A short overview

Simplex Method

- Every LP's feasible region is a **polyhedron** (a polygon in 2D, polytope in 3D).

Simplex Method

- Every LP's feasible region is a **polyhedron** (a polygon in 2D, polytope in 3D).
- A linear objective reaches its maximum (or minimum) at a **vertex** (in non-degenrate cases).

Simplex Method

- Every LP's feasible region is a **polyhedron** (a polygon in 2D, polytope in 3D).
- A linear objective reaches its maximum (or minimum) at a **vertex** (in non-degenrate cases).
- Why? Linear programs are like "flat" landscapes — no hills or valleys.

Simplex Method

- Every LP's feasible region is a **polyhedron** (a polygon in 2D, polytope in 3D).
- A linear objective reaches its maximum (or minimum) at a **vertex** (in non-degenrate cases).
- Why? Linear programs are like "flat" landscapes — no hills or valleys.
- The **Simplex method**:

Simplex Method

- Every LP's feasible region is a **polyhedron** (a polygon in 2D, polytope in 3D).
- A linear objective reaches its maximum (or minimum) at a **vertex** (in non-degenrate cases).
- Why? Linear programs are like “flat” landscapes — no hills or valleys.
- The **Simplex method**:
  1. Starts from one feasible vertex.

Simplex Method

- Every LP's feasible region is a **polyhedron** (a polygon in 2D, polytope in 3D).
- A linear objective reaches its maximum (or minimum) at a **vertex** (in non-degenrate cases).
- Why? Linear programs are like "flat" landscapes — no hills or valleys.
- The **Simplex method**:
  1. Starts from one feasible vertex.
  2. Moves along edges to neighboring vertices that improve the objective.

Simplex Method

- Every LP's feasible region is a **polyhedron** (a polygon in 2D, polytope in 3D).
- A linear objective reaches its maximum (or minimum) at a **vertex** (in non-degenrate cases).
- Why? Linear programs are like "flat" landscapes — no hills or valleys.
- The **Simplex method**:
  1. Starts from one feasible vertex.
  2. Moves along edges to neighboring vertices that improve the objective.
  3. Stops when no further improvement is possible.

Simplex Method

- Every LP's feasible region is a **polyhedron** (a polygon in 2D, polytope in 3D).
- A linear objective reaches its maximum (or minimum) at a **vertex** (in non-degenrate cases).
- Why? Linear programs are like "flat" landscapes — no hills or valleys.
- The **Simplex method**:
  1. Starts from one feasible vertex.
  2. Moves along edges to neighboring vertices that improve the objective.
  3. Stops when no further improvement is possible.
- Each move improves the objective value — and there are finitely many vertices.

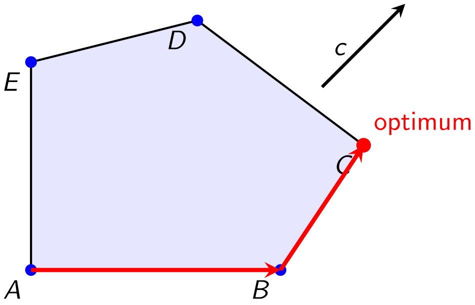
Simplex Method

- Every LP's feasible region is a **polyhedron** (a polygon in 2D, polytope in 3D).
- A linear objective reaches its maximum (or minimum) at a **vertex** (in non-degenrate cases).
- Why? Linear programs are like “flat” landscapes — no hills or valleys.
- The **Simplex method**:
  1. Starts from one feasible vertex.
  2. Moves along edges to neighboring vertices that improve the objective.
  3. Stops when no further improvement is possible.
- Each move improves the objective value — and there are finitely many vertices.
- Simplex always ends at an **optimal vertex** (if one exists).

Simplex Path on a Polygon (2D intuition)

Each step: move along an edge to a better vertex.

"Walk around the polygon" until no edge improves the objective.



Time Complexity of the Simplex Method

- $n \leftarrow$ number of variables

- In the **worst case**, there can be exponentially many vertices:

$$\text{Worst case: } O(2^n)$$

  (Klee–Minty cube example).

- In **practice**, Simplex is extremely fast — polynomial time.

- Theoretical guarantee (polynomial time) comes from **interior-point methods**

Duality in Linear Programming

An Example of Duality

Primal:

$$\max z = 5x_1 + 4x_2$$

$$\text{s.t.} \begin{cases} x_1 \leq 4 & (1) \\ x_1 + 2x_2 \leq 10 & (2) \\ 3x_1 + 2x_2 \leq 16 & (3) \\ x_1, x_2 \geq 0 \end{cases}$$

- Feasible solution $(x_1, x_2) = (4, 2)$ gives $z = 28 \implies$ lower bound.
- Multiply (3) by 2: $6x_1 + 4x_2 \leq 32 \implies z \leq 32 \implies$ upper bound.
- Adding (1)+(2)+(3): $5x_1 + 4x_2 \leq 30 \implies z \leq 30$.

Combining Inequalities to Bound the Optimum

Multiply constraints by nonnegative multipliers y_1, y_2, y_3:

$$(y_1 + y_2 + 3y_3)x_1 + (2y_2 + 2y_3)x_2 \leq 4y_1 + 10y_2 + 16y_3.$$

Combining Inequalities to Bound the Optimum

Multiply constraints by nonnegative multipliers y_1, y_2, y_3:

$$(y_1 + y_2 + 3y_3)x_1 + (2y_2 + 2y_3)x_2 \leq 4y_1 + 10y_2 + 16y_3.$$

To ensure an upper bound on $z = 5x_1 + 4x_2$, impose:

$$y_1 + y_2 + 3y_3 \geq 5, \quad 2y_2 + 2y_3 \geq 4.$$

Combining Inequalities to Bound the Optimum

Multiply constraints by nonnegative multipliers y_1, y_2, y_3:

$$(y_1 + y_2 + 3y_3)x_1 + (2y_2 + 2y_3)x_2 \le 4y_1 + 10y_2 + 16y_3.$$

To ensure an upper bound on $z = 5x_1 + 4x_2$, impose:

$$y_1 + y_2 + 3y_3 \ge 5, \quad 2y_2 + 2y_3 \ge 4.$$

Then minimize the RHS $4y_1 + 10y_2 + 16y_3$.

Dual:

$$\min\ w = 4y_1 + 10y_2 + 16y_3$$
$$\text{s.t.} \quad \begin{cases} y_1 + y_2 + 3y_3 \ge 5, \\ 2y_2 + 2y_3 \ge 4, \\ y_1, y_2, y_3 \ge 0. \end{cases}$$

Verifying Optimality via Duality

- We have established that for any pair of feasible solutions:

$$z(x) \leq w(y)$$

- Try $(x_1, x_2) = (3, 3.5) \implies z = 5(3) + 4(3.5) = 29.$
- Try $(y_1, y_2, y_3) = (0, 0.5, 1.5) \implies w = 4(0) + 10(0.5) + 16(1.5) = 29.$

Verifying Optimality via Duality

- We have established that for any pair of feasible solutions:

$$z(x) \leq w(y)$$

- Try $(x_1, x_2) = (3, 3.5) \implies z = 5(3) + 4(3.5) = 29.$
- Try $(y_1, y_2, y_3) = (0, 0.5, 1.5) \implies w = 4(0) + 10(0.5) + 16(1.5) = 29.$

- Therefore, when they match, **both are optimal:** $z^* = w^* = 29.$

Duality provides **certificates of optimality**: when a feasible x and y give equal objective values, they must be optimal.

Duality in Canonical Form

$$\text{(P)} \;\; \max c^\top x \quad \text{s.t.} \;\; Ax \le b, \;\; x \ge 0$$
$$\text{(D)} \;\; \min b^\top y \quad \text{s.t.} \;\; A^\top y \ge c, \;\; y \ge 0$$

- Each primal constraint $\Rightarrow$ dual variable.
- Each primal variable $\Rightarrow$ dual constraint.
- The two problems are mirrors of one another.

Weak Duality

$$c^\top x \leq y^\top Ax \leq y^\top b$$

- For any feasible x (primal) and y (dual): $z = c^\top x \leq w = b^\top y$.
- Dual feasible solutions give *upper bounds* on the primal optimum.

Convention: $\max \emptyset = -\infty$, $\min \emptyset = +\infty \implies$ always $z^* \leq w^*$.

Strong Duality

If both (P) and (D) have feasible solutions and one is bounded, then both attain the same finite optimum.

$$z^* = w^*$$

- Proof idea: simplex optimality conditions produce a dual feasible y with equal objective value.

Summary of primal–dual relationships

	Dual finite	Dual unbounded	Dual infeasible
Primal finite	$z^* = w^*$	impossible	impossible
Primal unbounded	impossible	impossible	possible
Primal infeasible	impossible	possible	possible

Interpretation:

- If one is unbounded, the other is infeasible.
- If one has a finite optimum, so does the other, with equal value.
- Both can be infeasible simultaneously.

Max-Flow Min-Cut Theorem with LP Duality

Max-Flow as a Linear Program

Given a directed network $(G = (V, E), s, t, c)$ with capacities $c(u, v)$:

- We can formulate max-flow problem as an LP over variables $f(u, v)$ for each edge $(u, v) \in E$.
- Optimal value = value of the maximum s–t flow.
- Assuming there are no incoming edges to s and no outgoing edges from t.

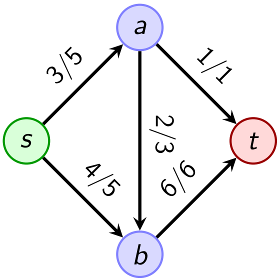
$$\max \quad \sum_{v:(s,v)\in E} f(s, v)$$

$$\text{s.t.} \quad \sum_{u:(u,v)\in E} f(u, v) = \sum_{w:(v,w)\in E} f(v, w), \quad \forall v \in V \setminus \{s, t\} \quad \text{(flow conservation)}$$

$$0 \le f(u, v) \le c(u, v), \qquad \forall (u, v) \in E \quad \text{(capacity)}$$

Flow Decomposition into Paths

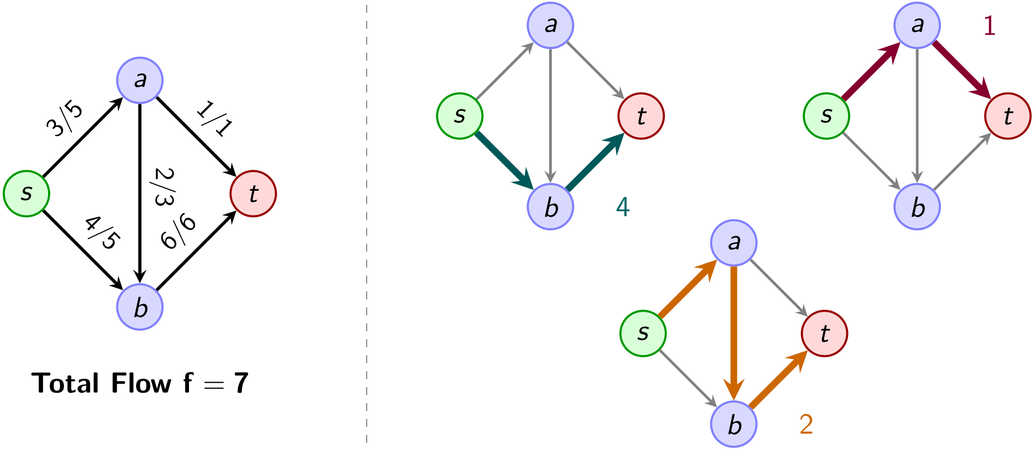
By the flow decomposition theorem, max-flow can be viewed as set of s-t paths.



Total Flow f = 7

Flow Decomposition into Paths

By the flow decomposition theorem, max-flow can be viewed as set of s-t paths.



Alternative view: Path-Based LP Formulation

- Let $\mathcal{P}$ be the set of all simple s–t paths, and for each path $p \in \mathcal{P}$, let x_p be the amount of flow sent along p (possibly exponentially many).

$$\begin{aligned}
\max \quad & \sum_{p\in\mathcal{P}} x_p \\
\text{s.t.} \quad & \sum_{p\in\mathcal{P}:(u,v)\in p} x_p \le c(u,v), \quad \forall (u,v)\in E \quad \text{(capacity)} \\
& x_p \ge 0, \quad \forall p \in \mathcal{P}.
\end{aligned}$$

Alternative view: Path-Based LP Formulation

- Let $\mathcal{P}$ be the set of all simple s–t paths, and for each path $p \in \mathcal{P}$, let x_p be the amount of flow sent along p (possibly exponentially many).

$$
\begin{aligned}
\max \quad & \sum_{p \in \mathcal{P}} x_p \\
\text{s.t.} \quad & \sum_{p \in \mathcal{P}:(u,v) \in p} x_p \le c(u,v), \quad \forall (u,v) \in E \quad \text{(capacity)} \\
& x_p \ge 0, \quad \forall p \in \mathcal{P}.
\end{aligned}
$$

$\max_f \; |f| = \mathrm{OPT}_{\text{primal}}$

Next: Very clean dual!

Alternative view: Path-Based LP Formulation

- Let $\mathcal{P}$ be the set of all simple s–t paths, and for each path $p \in \mathcal{P}$, let x_p be the amount of flow sent along p (possibly exponentially many).

$$
\begin{aligned}
\max \quad & \sum_{p \in \mathcal{P}} x_p \\
\text{s.t.} \quad & \sum_{p \in \mathcal{P} : (u,v) \in p} x_p \le c(u,v), \quad \forall (u,v) \in E \quad \text{(capacity)} \qquad \leftarrow y_{u,v} \\
& x_p \ge 0, \qquad \forall p \in \mathcal{P}.
\end{aligned}
$$

$\max_f \; |f| = \mathrm{OPT}_{\text{primal}}$

Next: Very clean dual!

Dual of the Path-Based LP

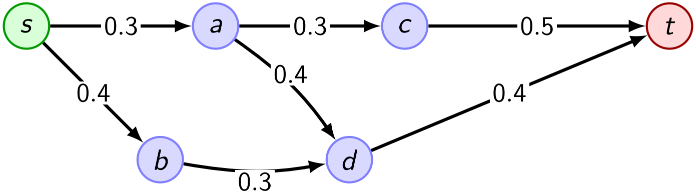
Dual variables: $y_{u,v} \geq 0$ for each edge $(u, v) \in E$.

Dual LP:

$$
\begin{aligned}
\min \quad & \sum_{(u,v)\in E} c(u,v)\, y_{u,v} \\
\text{s.t.} \quad & \sum_{(u,v)\in p} y_{u,v} \;\geq\; 1, && \forall s\text{–}t \text{ paths } p \in \mathcal{P} \\
& y_{u,v} \;\geq\; 0, && \forall (u,v) \in E.
\end{aligned}
$$

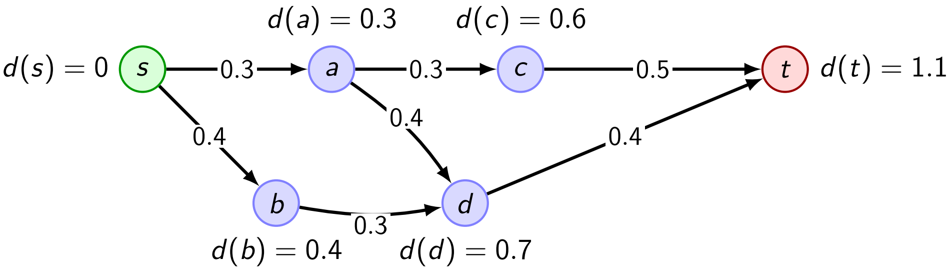
Interpretation of Dual

- Interpret $y_{u,v}$ as a length on edge (u, v).



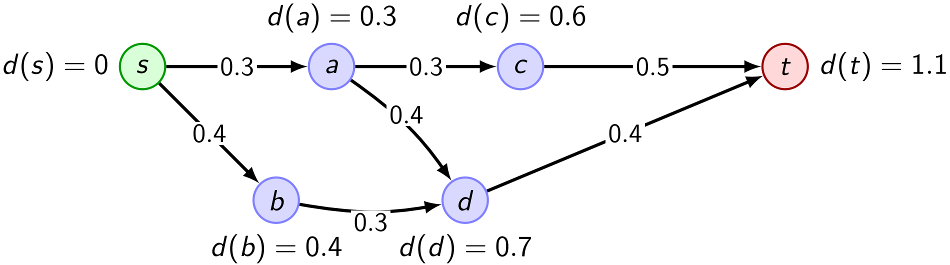
Interpretation of Dual

- Interpret $y_{u,v}$ as a length on edge (u, v).
- Constraint: every s–t path has total length at least 1.
  $\Rightarrow$ in the metric defined by y, distance$(s, t) \geq 1$.



Interpretation of Dual

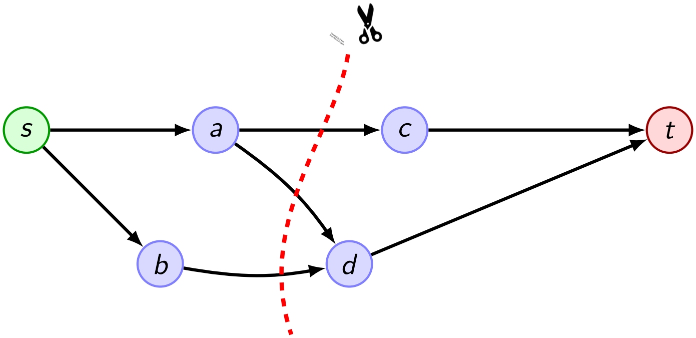
- Interpret $y_{u,v}$ as a length on edge (u, v).
- Constraint: every s–t path has total length at least 1.
  $\Rightarrow$ in the metric defined by y, distance$(s, t) \geq 1$.
- Objective: minimize the capacity-weighted sum of edge lengths.



Cuts $\Rightarrow$ Feasible Dual Solutions

- Given an $(s-t)$-cut A, define

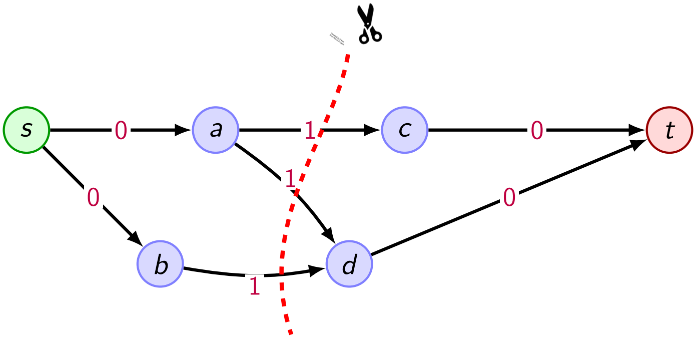
$$y_{u,v} := \begin{cases} 1 & \text{if } u \in A,\ v \notin A \text{ (edge crosses the cut)}, \\ 0 & \text{otherwise.} \end{cases}$$



Cuts $\Rightarrow$ Feasible Dual Solutions

- Given an $(s-t)$-cut A, define

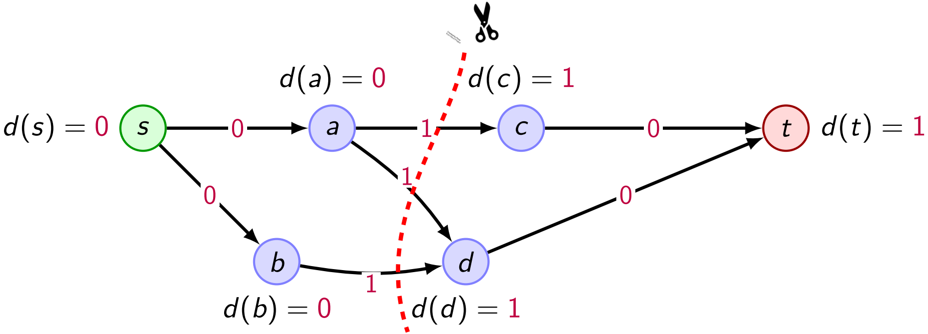
$$y_{u,v} := \begin{cases} 1 & \text{if } u \in A,\ v \notin A \text{ (edge crosses the cut)}, \\ 0 & \text{otherwise.} \end{cases}$$



Cuts ⇒ Feasible Dual Solutions

- Given an $(s-t)$-cut A, define

$$y_{u,v} := \begin{cases} 1 & \text{if } u \in A,\ v \notin A \text{ (edge crosses the cut)}, \\ 0 & \text{otherwise.} \end{cases}$$



Cuts $\Rightarrow$ Feasible Dual Solutions

- Every s–t path must cross the cut at least once, so the path constraints hold:

$$\sum_{(u,v)\in p} y_{u,v} \geq 1\,.$$

- Dual objective value:

$$\mathrm{OPT}_{\mathrm{dual}} \leq \sum_{(u,v)\in E} c(u,v)\, y_{u,v} \;=\; \sum_{u\in A,\, v\notin A} c(u,v) \;=\; \mathrm{capacity}(A)\,.$$

- Therefore,

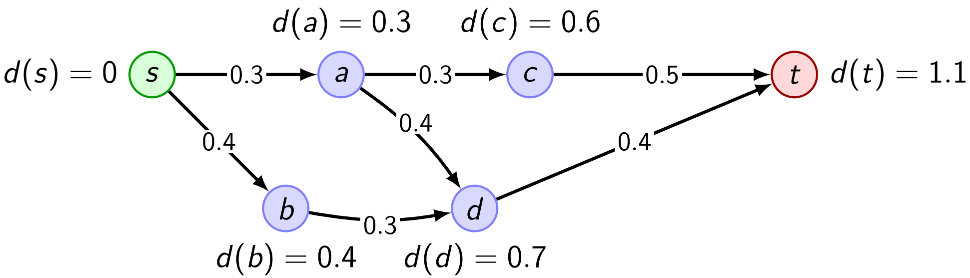
$$\boxed{\mathrm{OPT}_{\mathrm{dual}} \;\leq\; \min_{(s\text{–}t)\ \mathrm{cuts}\ A} \mathrm{capacity}(A)\,.}$$

Dual $\Rightarrow$ Cut

Now go in the other direction: from any dual solution y to a cut.

Step 1: Distances from s

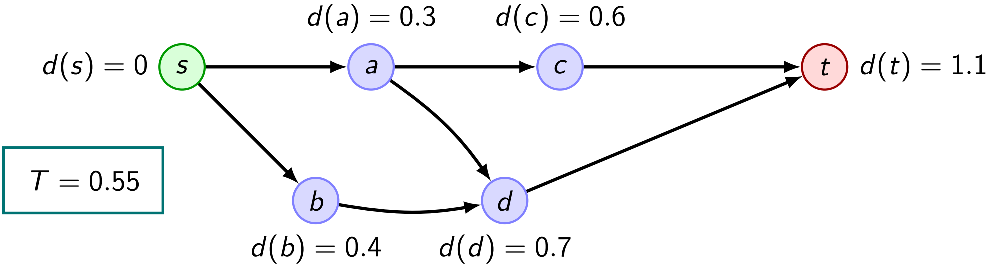
- Compute $d(v)$ = shortest-path distance from s to v (e.g., Dijkstra).
- Dual constraints $\Rightarrow d(t) \geq 1$.



Randomized Rounding: Dual $\Rightarrow$ Cut

Step 2: Random threshold

- Pick T uniformly at random in $[0, 1)$.

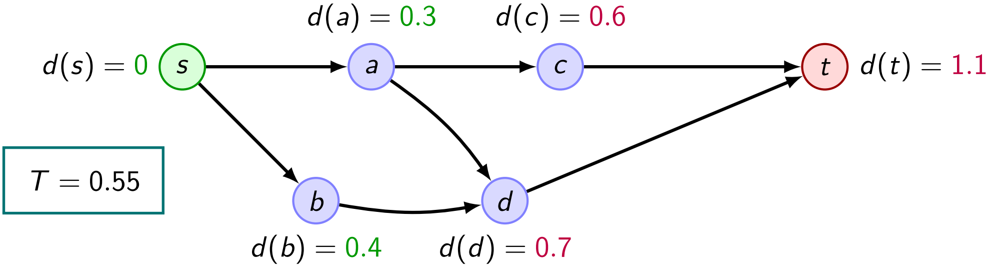


Randomized Rounding: Dual $\Rightarrow$ Cut

Step 2: Random threshold

- Pick T uniformly at random in $[0, 1)$.
- Define the random cut

$$A := \{v \in V : d(v) \leq T\}.$$



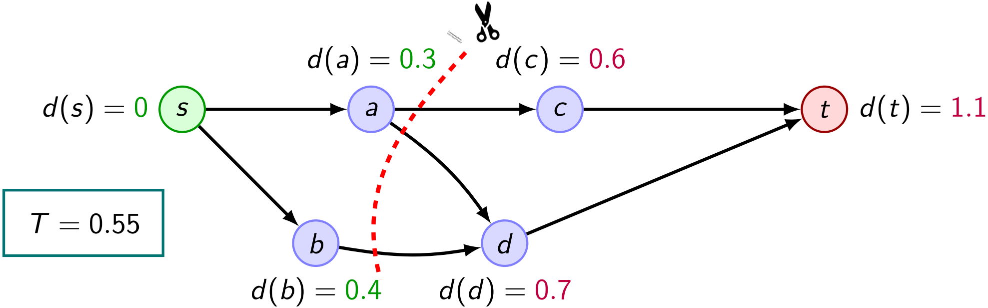
Randomized Rounding: Dual ⇒ Cut

Step 2: Random threshold

- Pick T uniformly at random in $[0, 1)$.
- Define the random cut

$$A := \{v \in V : d(v) \le T\}.$$

- Then $s \in A$ but $t \notin A$, so A is always a valid s–t cut.



Probability of Being a Cut Edge

For an edge (u, v), what is the probability of $u \in A$, and $v \notin A$?

- If $d(u) > d(v) \implies u$ and v will not be part of a cut.

Probability of Being a Cut Edge

For an edge (u, v), what is the probability of $u \in A$, and $v \notin A$?

- If $d(u) > d(v) \implies u$ and v will not be part of a cut.
- So assume $d(u) \leq d(v)$:

$$\mathbf{Pr}[u \in A,\ v \notin A] = \mathbf{Pr}[d(u) \leq T < d(v)] \leq d(v) - d(u)$$

provided $0 \leq d(u) \leq d(v) \leq 1$ (other cases only make this smaller).

Probability of Being a Cut Edge

For an edge (u, v), what is the probability of $u \in A$, and $v \notin A$?

- If $d(u) > d(v) \implies u$ and v will not be part of a cut.
- So assume $d(u) \leq d(v)$:

$$\mathbf{Pr}[u \in A,\ v \notin A] = \mathbf{Pr}[d(u) \leq T < d(v)] \leq d(v) - d(u)$$

  provided $0 \leq d(u) \leq d(v) \leq 1$ (other cases only make this smaller).
- Shortest-path distances satisfy

$$d(v) \ \leq \ d(u) + y_{u,v}, \implies d(v) - d(u) \leq y_{u,v}$$

.

Probability of Being a Cut Edge

For an edge (u, v), what is the probability of $u \in A$, and $v \notin A$?

- If $d(u) > d(v) \implies u$ and v will not be part of a cut.
- So assume $d(u) \leq d(v)$:

$$\mathbf{Pr}[u \in A,\ v \notin A] = \mathbf{Pr}[d(u) \leq T < d(v)] \leq d(v) - d(u)$$

  provided $0 \leq d(u) \leq d(v) \leq 1$ (other cases only make this smaller).
- Shortest-path distances satisfy

$$d(v) \ \leq \ d(u) + y_{u,v}, \implies d(v) - d(u) \leq y_{u,v}$$

  .
- Therefore

$$\mathbf{Pr}[u \in A,\ v \notin A] = \mathbf{Pr}[d(u) \leq T < d(v)] \leq y_{u,v}$$

Bounding the Expected Capacity

Given any dual solution y, expected capacity:

$$\begin{aligned}\mathbf{E}_T[\text{capacity}(A)] &= \sum_{(u,v)\in E} c(u,v)\,\mathbf{Pr}[u \in A,\ v \notin A]. \\ &\leq \sum_{(u,v)\in E} c(u,v)\,y_{u,v}\,.\end{aligned}$$

Bounding the Expected Capacity

Given any dual solution y, expected capacity:

$$\mathbf{E}_T[\text{capacity}(A)] = \sum_{(u,v)\in E} c(u,v)\,\mathbf{Pr}[u \in A,\ v \notin A].$$
$$\leq \sum_{(u,v)\in E} c(u,v)\, y_{u,v}\,.$$

- Averaging principle: There exists a (deterministic) choice of T^* with:

$$\text{capacity}(A_{T^*}) \leq \sum_{(u,v)\in E} c(u,v)\, y_{u,v}\,.$$

- Hence,

$$\min_{(s-t)\ \text{cuts}\ A} \text{capacity}(A) \ \leq\ \text{OPT}_{\text{dual}}\,.$$

Dual $\Leftrightarrow$ Min-Cut

We have shown:

- Any cut A gives a feasible dual solution:

$$\mathrm{OPT}_{\mathrm{dual}} \;\leq\; \min_{(s\text{–}t)\text{ cuts } A} \mathrm{capacity}(A)\,.$$

- Given any dual solution y, we can round it to a cut:

$$\min_{(s\text{–}t)\text{ cuts } A} \mathrm{capacity}(A) \;\leq\; \mathrm{OPT}_{\mathrm{dual}}\,.$$

Combining:

$$\min_{(s\text{–}t)\text{ cuts } A} \mathrm{capacity}(A) = \mathrm{OPT}_{\mathrm{dual}}\,.$$

LP Duality $\Rightarrow$ Max-Flow = Min-Cut

- We have shown:

$$\max_f \; |f| = \mathrm{OPT}_{\mathrm{primal}}$$

$$\min_{(s-t) \text{ cuts } A} \mathrm{capacity}(A) = \mathrm{OPT}_{\mathrm{dual}} \,.$$

- Strong Duality implies:

$$\mathrm{OPT}_{\mathrm{primal}} = \mathrm{OPT}_{\mathrm{dual}}$$

- Putting all of these together implies

$$\max_f \; |f| = \min_{(s-t) \text{ cuts } A} \mathrm{capacity}(A)$$

What Is NP-Hardness?

The Core Problem: Selection Bias

- Introductory algorithm books suffer from **selection bias**.

- They focus on problems with clever, fast algorithms (e.g., sorting, shortest paths, MSTs).

The Core Problem: Selection Bias

- Introductory algorithm books suffer from **selection bias**.
- They focus on problems with clever, fast algorithms (e.g., sorting, shortest paths, MSTs).
- Many important problems have **no fast algorithms known**.
- These problems are deemed “intractable.”

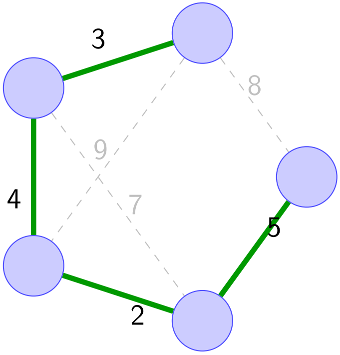
MST vs TSP

An Algorithmic Mystery

"Easy": Minimum Spanning Tree (MST)

Problem: Find a spanning tree (a subset of edges that connects all vertices without cycles) of minimum total edge cost.

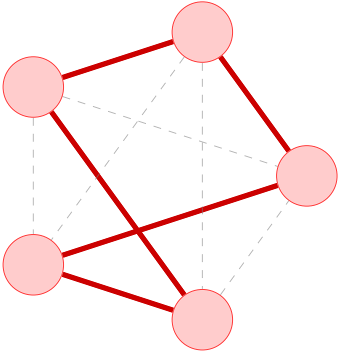
- Solvable by blazingly fast algorithms:
  - Prim's
  - Kruskal's
- **Running Time:** $O((m + n) \log n)$.
- This is a **computationally easy** problem.



"Hard": Traveling Salesman Problem (TSP)

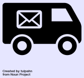
Problem: Find a tour (a cycle visiting every vertex exactly once) of minimum total edge cost.

- The definition looks deceptively similar to MST.
- No fast algorithm is known.
- Exhaustive search is $O(n!)$, which is **infeasible**.
- This is **computationally hard**.



Why TSP Matters: Real-World Intractability

TSP is a powerful template for many practical optimization problems.

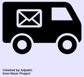


Mail Deliveries

finding the shortest route for deliveries.

Why TSP Matters: Real-World Intractability

TSP is a powerful template for many practical optimization problems.



Mail Deliveries

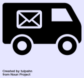
finding the shortest route for deliveries.

Genome Sequencing

Finding the most plausible ordering of overlapping gene fragments.

Why TSP Matters: Real-World Intractability

TSP is a powerful template for many practical optimization problems.

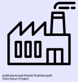


Mail Deliveries

finding the shortest route for deliveries.

Genome Sequencing

Finding the most plausible ordering of overlapping gene fragments.



Factory Assembly

Minimizing setup costs between assembling different car models.

Defining “Easy” and “Hard” Problems

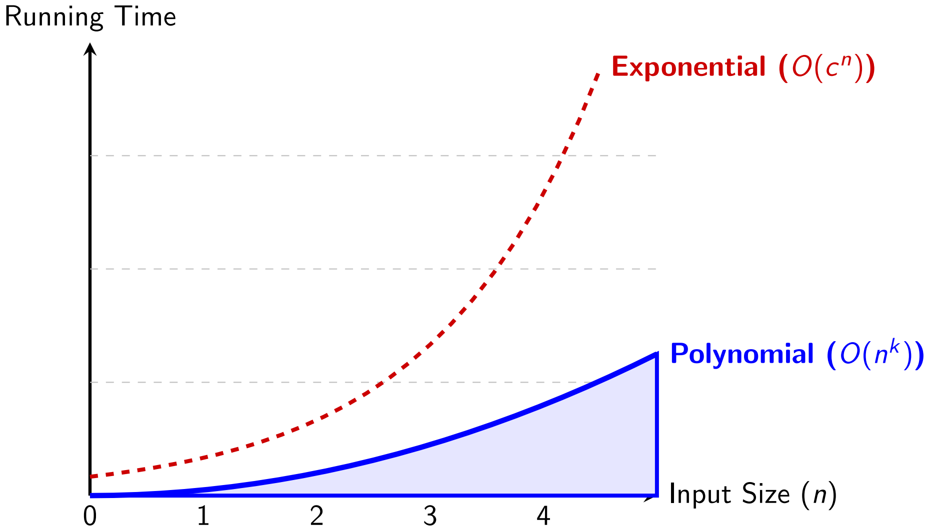
Or, a gentle introduction to complexity classes

Easy and Hard Problems

An oversimplified view:

- **Easy:** can be solved with a polynomial-time algorithm.
- **Hard:** require exponential time in the worst case.

Polynomial vs. Exponential Time



P: Polynomial Time Solvable Problems

- Complexity theory classifies problems based on their *inherent difficulty*;
- Algorithms can be fast or slow, clever or naive, but our statements about the *problem itself*.
- A problem is polynomial time solvable if there is an algorithm that correctly solves it in $O(n^k)$ time, for some constant k, where n is the input length.
- still polynomial even $k = 10^10$.
- This is worst-case running time. (maximum running time over all possible inputs of size n)
- **P**: Problems solvable in **P**olynomial time (easy to **solve**).

NP: Nondeterministic Polynomial time

- **NP**: Problems whose solutions are **verifiable** in **P**olynomial time (easy to **check**).

NP: Nondeterministic Polynomial time

- **NP**: Problems whose solutions are **verifiable** in **P**olynomial time (easy to **check**).

- We know that $\mathbf{P} \subseteq \mathbf{NP}$, e.g., MST $\in$ NP.

NP: Nondeterministic Polynomial time

- **NP**: Problems whose solutions are **verifiable** in **P**olynomial time (easy to **check**).

- We know that **P** $\subseteq$ **NP**, e.g., MST $\in$ NP.

- For many problems in NP, no polynomial-time algorithm is known, (e.g., TSP).

NP: Nondeterministic Polynomial time

- **NP**: Problems whose solutions are **verifiable** in **P**olynomial time (easy to **check**).

- We know that $\mathbf{P} \subseteq \mathbf{NP}$, e.g., MST $\in$ NP.

- For many problems in NP, no polynomial-time algorithm is known, (e.g., TSP).

- A problem is NP-hard if *every* NP problem reduces to it.

Decision Problems: The Formal Foundation

- Complexity classes are formally defined using problems that yield a simple **YES or NO** answer.
- This restriction is necessary to create a clean mathematical framework for verification.

Decision Problems: The Formal Foundation

- Complexity classes are formally defined using problems that yield a simple **YES or NO** answer.
- This restriction is necessary to create a clean mathematical framework for verification.
- Optimization problems (finding the minimum or the maximum) are closely connected to their related decision problems (is the minimum $\leq k$?).

Decision Problems: The Formal Foundation

- Complexity classes are formally defined using problems that yield a simple **YES or NO** answer.
- This restriction is necessary to create a clean mathematical framework for verification.
- Optimization problems (finding the minimum or the maximum) are closely connected to their related decision problems (is the minimum $\leq k$?).

Decision

- **MST (Decision):** Is there a spanning tree with total cost $\leq k$?
- **TSP (Decision):** Is there a tour with total cost $\leq k$?

Optimization

- **MST (Optimization):** Find the minimum cost spanning tree.
- **TSP (Optimization):** Find the shortest tour.

The P vs. NP Conjecture

Conjecture: $\mathbf{P} \neq \mathbf{NP}$. Most experts believe this is true.

> *If P=NP, then the world would be a profoundly different place than we usually assume it to be. There would be no special value in "creative leaps," no fundamental gap between solving a problem and recognizing the solution once it's found. Everyone who could appreciate a symphony would be Mozart; everyone who could follow a step-by-step argument would be Gauss; everyone who could recognize a good investment strategy would be Warren Buffett. It's possible to put the point in Darwinian terms: if this is the sort of universe we inhabited, why wouldn't we already have evolved to take advantage of it?*

— Scott Aaronson, on Shtetl-Optimized

What is “NP-Hard”?

- A problem is **NP-hard** if a polynomial-time algorithm for it would **refute the P $\neq$ NP conjecture**.

What is "NP-Hard"?

- A problem is **NP-hard** if a polynomial-time algorithm for it would **refute the P $\neq$ NP conjecture**.

- It is one of the hardest problems in NP (or harder).

What is "NP-Hard"?

- A problem is **NP-hard** if a polynomial-time algorithm for it would **refute the P $\neq$ NP conjecture**.
- It is one of the hardest problems in NP (or harder).
- A fast algorithm for one NP-hard problem (like TSP) would solve **thousands** of other unsolved problems.

What is "NP-Hard"?

- A problem is **NP-hard** if a polynomial-time algorithm for it would **refute the P $\neq$ NP conjecture**.
- It is one of the hardest problems in NP (or harder).
- A fast algorithm for one NP-hard problem (like TSP) would solve **thousands** of other unsolved problems.
- This powerful implication is the "strong evidence" of its intractability.

Algorithmic Strategies

The "You Can't Have It All" Principle

An algorithm for an NP-hard problem cannot be all three (assuming $\mathbf{P} \neq \mathbf{NP}$):

General-Purpose Solves all possible inputs.

Correct Always finds the optimal solution.

Fast Runs in polynomial time.

You must compromise on at least one.

Three Algorithmic Strategies

- **Compromise on Generality:** Solve only **special cases** or constrained versions of the problem.
  - *Example:* Weighted Independent Set on path graphs is easy, but on general graphs is NP-hard.

- **Compromise on Correctness:** Use **heuristics** (e.g., Greedy, Local Search).
  - They are fast but may not be optimal. Good for "approximate" answers.

- **Compromise on Speed:** Use an **exact algorithm** that is faster than exhaustive search, but still exponential.
  - *Example:* Dynamic Programming for TSP, or sophisticated SAT/MIP Solvers.

References

Goemans, M. (2015).
Lecture notes on linear programming.
Lecture notes for 18.310A Principles of Discrete Applied Mathematics.
Accessed on November 10, 2025.

Roughgarden, T. (2022).
Algorithms Illuminated: Omnibus Edition.
Soundlikeyourself Publishing, LLC.

Trevisan, L. (2011).
The linear programming formulation of maximum cut and its dual.
Lecture Notes for CS261: Optimization and Algorithmic Paradigms, Lecture 15.