COMP 382: Reasoning about Algorithms

Linear Programming & Duality

Prof. Maryam Aliakbarpour

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

November 11, 2025

Today's Lecture

1. The Baseball Elimination Problem

2. Linear Programming

2.1 Simplex Method

2.2 Duality in Linear Programming

2.3 Duality and the Max-Flow = Min-Cut Theorem

Reading:

- Chapter H in [Erickson, 2019] and lecture notes in [?]

Content adapted from the same references in [Erickson, 2019].

The Baseball Elimination Problem

And its reduction to max-flow problem

Can my team still win?

- **Mid season:** "Is Houston Astros *mathematically* alive?"
- Trivial check: if some opponent already has more wins than our max possible, we're done.

AL WEST	W	L
Seattle Mariners y	90	72
Houston Astros	87	75
Texas Rangers	81	81
Athletics	76	86
Los Angeles Angels	72	90

Can my team still win?

- But often it's *not* trivial: The outcomes of remaining games among *other* teams constrain each other.

Can my team still win?

- But often it's *not* trivial: The outcomes of remaining games among *other* teams constrain each other.
- Two games left. Can Team C still win the championship?

Team	Current Wins
Team A	61
Team B	61
Team C	60

Can my team still win?

- But often it's *not* trivial: The outcomes of remaining games among *other* teams constrain each other.
- Two games left. Can Team C still win the championship?

Team	Current Wins
Team A	61
Team B	61
Team C	60

- No one is *individually* out of reach, yet the *schedule* makes it impossible:
  - C's maximum possible wins are **62** (if C wins both remaining games).
  - Assume Team A and Team B have a final game scheduled against each other.
  - Since A and B play each other, at least one of them is guaranteed to reach **62** wins or more.

Problem Statement: Baseball Elimination

Setting:

- We have a league of n teams labeled $1, 2, \ldots, n$.
- For each team i:
  - $W[i]$ — number of games **already won**.
  - $R[i]$ — number of **remaining games**.
- For each pair of teams (i, j):
  - $G[i, j]$ — number of **remaining head-to-head games** between them.

Problem Statement: Baseball Elimination

Setting:

- We have a league of n teams labeled $1, 2, \ldots, n$.
- For each team i:
  - $W[i]$ — number of games **already won**.
  - $R[i]$ — number of **remaining games**.
- For each pair of teams (i, j):
  - $G[i, j]$ — number of **remaining head-to-head games** between them.

Goal: Determine whether a specific team n (our team) is mathematically eliminated.

- If not, provide a certificate (subset of teams proving possibility).

Our Appraoch

- We assume Team n wins all $R[n]$ of its remaining games.

$$W_{max} := W[n] + R[n]$$

Our Appraoch

- We assume Team n wins all $R[n]$ of its remaining games.

$$W_{max} := W[n] + R[n]$$

- For all other teams $i \in [n-1]$, we update the number of remaining games.

$$R[i] \leftarrow R[i] - G[i, n]$$

Our Appraoch

- We assume Team n wins all $R[n]$ of its remaining games.

$$W_{max} := W[n] + R[n]$$

- For all other teams $i \in [n-1]$, we update the number of remaining games.

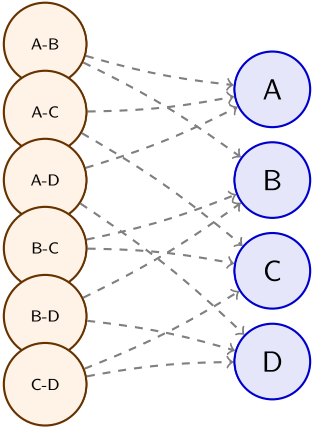
$$R[i] \leftarrow R[i] - G[i, n]$$

- Find a possible results for the remaining games among *other* teams in such a way that no opponent to surpass Team n's maximum score.

$$\text{new wins of } i < W_{max} - W[i]$$

Why Max Flow Can Help

Max flow models the distribution of wins from unplayed games, testing if there's a hypothetical outcome where a no team can catch the leader.



Reduction to Max Flow: The Network G'

The problem of assigning outcomes to games is perfectly modeled by a maximum flow network.

Reduction to Max Flow: The Network G'

The problem of assigning outcomes to games is perfectly modeled by a maximum flow network.

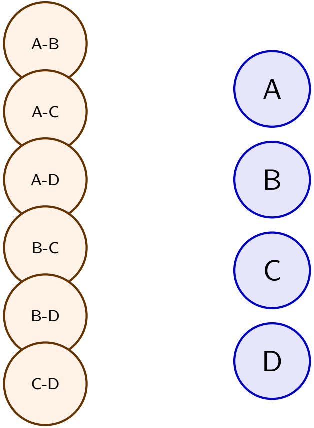
Nodes:

- **s** (source) and **t** (sink).
- **Game Nodes** $g_{i,j}$: For every pair $i, j \neq n$. (Represents $G[i, j]$ games to be played).
- **Team Nodes** t_i: For every opponent $i \neq n$.

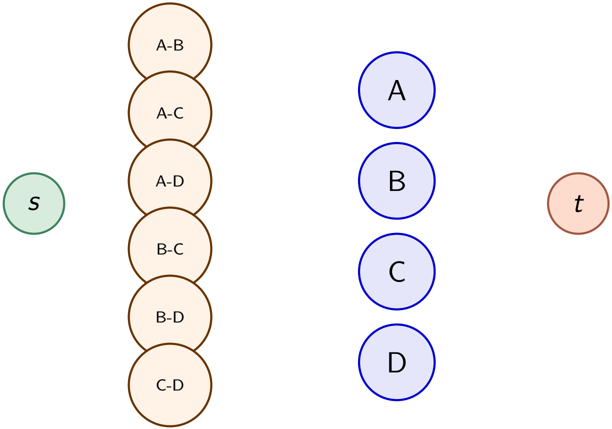
Edges & Capacities:

- $\mathbf{s} \to g_{i,j}$: Capacity $G[i, j]$. (Total flow is the total number of games left).
- $g_{i,j} \to t_i$ and $g_{i,j} \to t_j$: Capacity ∞. (Game outcome: win for i or j).
- $t_i \to \mathbf{t}$: Capacity $\mathbf{W_{max}} - \mathbf{W[i]}$. (Constraint: t_i cannot exceed Team n's max wins).

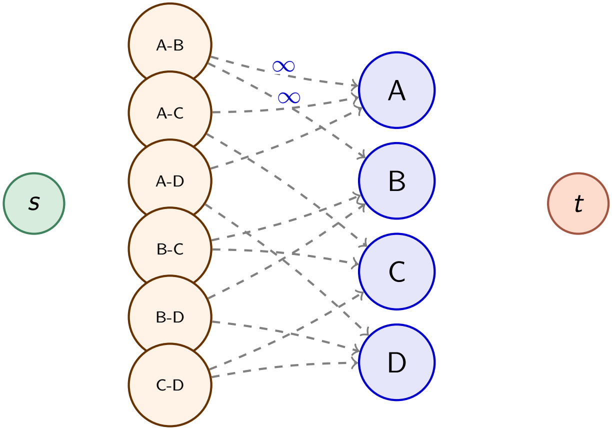
The Flow Network



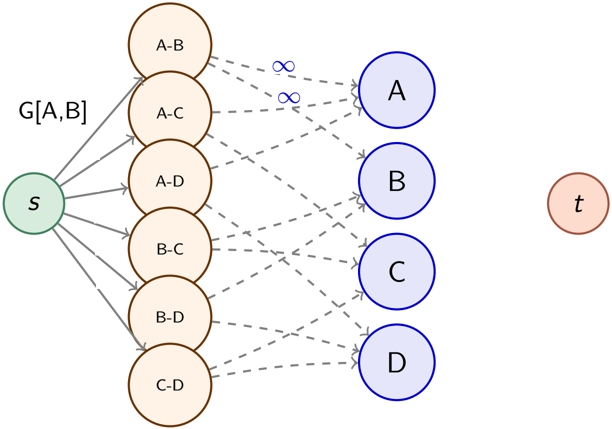
The Flow Network



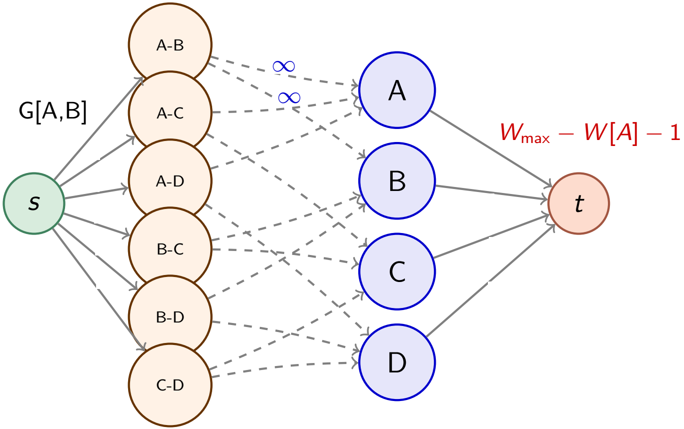
The Flow Network



The Flow Network



The Flow Network



Finding The Final Solution

- Team n can finish in first place if and only if a flow in G' **saturates** every edge leaving s.

- This has to be the max-flow, since the cut $S = \{s\}$ and $T = V \setminus \{s\}$ is fully saturated.

- Certificate: the flow in g_{ij} and t_j indicates how many games between i and j are won by t_j.

Proof of Correctness: The Two-Way Proof Structure

Part 1: Completeness

We must show that a **valid solution** in the original problem results in a **valid flow** in our new network.

$$\text{Original Solution} \implies \text{Valid Flow}$$

Part 2: Soundness

We must show that a **valid (max) flow** in our network gives us a **valid solution** back in the original problem.

$$\text{Valid Flow} \implies \text{Original Solution}$$

Completeness: Original Solution $\implies$ Valid Flow

- **A solution exists:** We can define a scenario (win assignment) where n finishes first.

Completeness: Original Solution $\implies$ Valid Flow

- **A solution exists:** We can define a scenario (win assignment) where n finishes first.
- Map each win to 1 unit of flow: $g_{i,j} \to t_i$ for a win by i.

Completeness: Original Solution $\implies$ Valid Flow

- **A solution exists:** We can define a scenario (win assignment) where n finishes first.
- Map each win to 1 unit of flow: $g_{i,j} \to t_i$ for a win by i.
- Since every game $G[i, j]$ is assigned, $s \to g_{i,j}$ is saturated.

Completeness: Original Solution $\implies$ Valid Flow

- **A solution exists:** We can define a scenario (win assignment) where n finishes first.
- Map each win to 1 unit of flow: $g_{i,j} \to t_i$ for a win by i.
- Since every game $G[i,j]$ is assigned, $s \to g_{i,j}$ is saturated.
- Since no t_i exceeds W_{max}, the capacity $t_i \to t$ is respected.

Completeness: Original Solution $\implies$ Valid Flow

- **A solution exists:** We can define a scenario (win assignment) where n finishes first.
- Map each win to 1 unit of flow: $g_{i,j} \to t_i$ for a win by i.
- Since every game $G[i,j]$ is assigned, $s \to g_{i,j}$ is saturated.
- Since no t_i exceeds W_{max}, the capacity $t_i \to t$ is respected.
- The flow conservation also holds at all nodes. All the outgoing edges of s are fully saturated.

Completeness: Original Solution $\implies$ Valid Flow

- **A solution exists:** We can define a scenario (win assignment) where n finishes first.
- Map each win to 1 unit of flow: $g_{i,j} \to t_i$ for a win by i.
- Since every game $G[i,j]$ is assigned, $s \to g_{i,j}$ is saturated.
- Since no t_i exceeds W_{max}, the capacity $t_i \to t$ is respected.
- The flow conservation also holds at all nodes. All the outgoing edges of s are fully saturated.
- Hence, **the flow is feasible and maximized.**

Soundness: Valid Flow $\implies$ Original Solution

- **If a valid flow saturates outgoing edges of s:** the flow conservation holds at every node.

Soundness: Valid Flow $\Longrightarrow$ Original Solution

- **If a valid flow saturates outgoing edges of** s**:** the flow conservation holds at every node.

- The flow values $f(g_{i,j} \to t_i)$ define a valid win assignment for all remaining games:

$$f(g_{i,j} \to t_i) + f(g_{i,j} \to t_j) = G[i,j] \, .$$

Soundness: Valid Flow $\implies$ Original Solution

- **If a valid flow saturates outgoing edges of s:** the flow conservation holds at every node.

- The flow values $f(g_{i,j} \rightarrow t_i)$ define a valid win assignment for all remaining games:

$$f(g_{i,j} \rightarrow t_i) + f(g_{i,j} \rightarrow t_j) = G[i,j]\,.$$

- Because of the $t_i \rightarrow t$ capacity constraint, no opponent i can win more than $W_{max} - W[i]$ new games.

Soundness: Valid Flow $\implies$ Original Solution

- **If a valid flow saturates outgoing edges of s:** the flow conservation holds at every node.

- The flow values $f(g_{i,j} \to t_i)$ define a valid win assignment for all remaining games:

$$f(g_{i,j} \to t_i) + f(g_{i,j} \to t_j) = G[i,j]\,.$$

- Because of the $t_i \to t$ capacity constraint, no opponent i can win more than $W_{max} - W[i]$ new games.

- Since team n can win W_{max}, **the assignment implies a solution to the original problem.**

The Equivalence

We have successfully mapped the baseball elimination problem to a flow problem:

$$\text{Original Solution} \quad \Leftrightarrow \quad \text{Valid Flow}$$

Therefore, finding the max flow value directly solves the baseball elimination problem.

Complexity

Network Size (V, E)

- **Vertices (V):** 2 (s, t) $+(n-1)$ (teams) $+\binom{n-1}{2}$ (games)

$$\implies V = O(n^2)$$

- **Edges (E):** $\binom{n-1}{2}$ $(s \to g)$ $+2 \cdot \binom{n-1}{2}$ $(g \to t)$ $+(n-1)$ $(t \to t)$

$$\implies E = O(n^2)$$

Complexity

Network Size (V, E)

- **Vertices (V):** 2 (s, t) $+(n-1)$ (teams) $+\binom{n-1}{2}$ (games)

$$\implies V = O(n^2)$$

- **Edges (E):** $\binom{n-1}{2}$ $(s \to g)$ $+2 \cdot \binom{n-1}{2}$ $(g \to t)$ $+(n-1)$ $(t \to t)$

$$\implies E = O(n^2)$$

Max Flow Computation

- Using Edmond-Karp algorithm: $O(|V|\,|E|^2) = O(n^6)$.

The Max-Flow Reduction Paradigm

- **Graph Construction.** We model the problem as a directed graph $G = (V, E)$ with a designated *source* (s) and *sink* (t). Edge capacities $c(u, v)$ are strategically defined to enforce the *constraints* of the original problem.

The Max-Flow Reduction Paradigm

- **Graph Construction.** We model the problem as a directed graph $G = (V, E)$ with a designated *source* (s) and *sink* (t). Edge capacities $c(u, v)$ are strategically defined to enforce the *constraints* of the original problem.

- **Flow Translates to Solution.** The flow value $f(e)$ on specific edges directly maps back to a solution in the original problem.

The Max-Flow Reduction Paradigm

- **Graph Construction.** We model the problem as a directed graph $G = (V, E)$ with a designated *source* (s) and *sink* (t). Edge capacities $c(u, v)$ are strategically defined to enforce the *constraints* of the original problem.

- **Flow Translates to Solution.** The flow value $f(e)$ on specific edges directly maps back to a solution in the original problem.

- **Soundness and Completeness.** A successful reduction establishes a *two-way equivalence relationship*:

$$\text{Original Solution Exists} \iff \text{Required Flow is Achieved}$$

  This proves that the flow network precisely captures the constraints and objectives of the original problem.

Conclusion

Modeling Feasibility and Optimization. Max Flow provides a powerful framework for solving a wide class of discrete decision and optimization problems by transforming them into a network representation.

This method is particularly effective for problems involving:

- *Resource allocation*
- *Matching*
- *Feasibility checks* subject to capacity constraints.

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

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

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

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.

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

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.

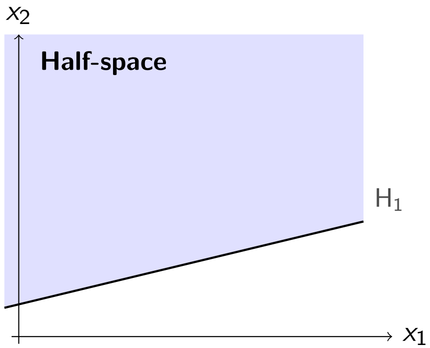
Feasibility Region: From half-spaces to polygons

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*.

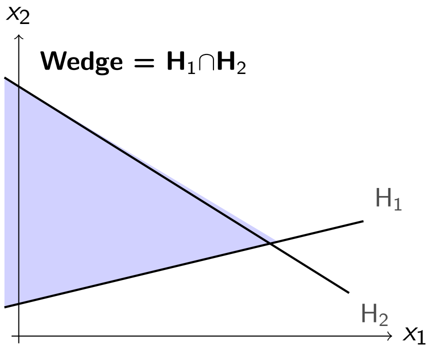


Feasibility Region: From half-spaces to polygons

Step 2. Wedge.

Two inequalities ⇒ intersection of two half-spaces.

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

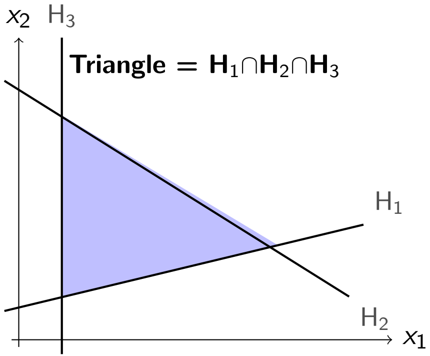


Feasibility Region: From half-spaces to polygons

Step 3. Triangle.

A third inequality can bound the region in 2D.

Feasible set: *triangle* (bounded).

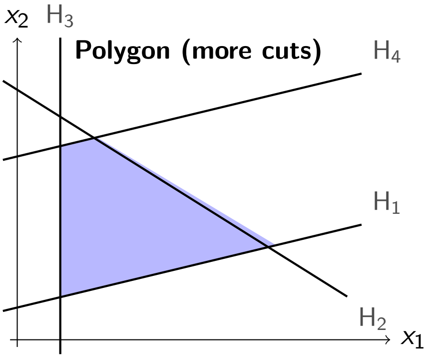


Feasibility Region: From half-spaces to polygons

Step 4. Polygon.

Additional constraints cut off corners $\Rightarrow$ refined feasible set.

Feasible set: *polygon*.

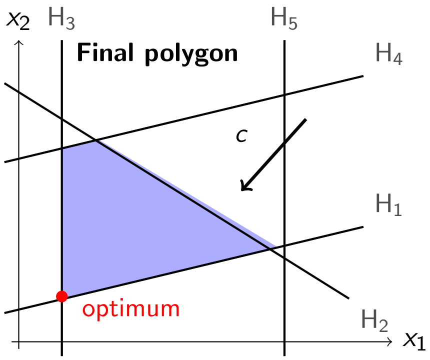


Feasibility Region: From half-spaces to polygons

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.

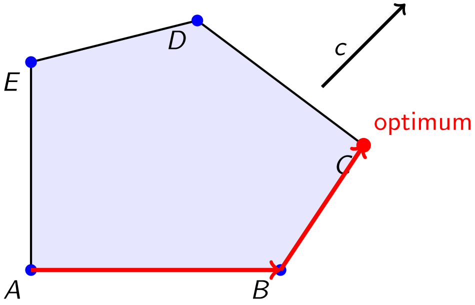
Simplex Method

- 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:

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

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

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

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:

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

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

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

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

Dual:

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

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.

Duality and the Max-Flow = Min-Cut Theorem

Max-Flow Problem as a Linear Program

Given a directed graph $G = (V, E)$ with capacities u_{ij}, source s, sink t.

$$\max \sum_{(s,j)\in E} f_{sj} - \sum_{(i,s)\in E} f_{is}$$

$$\text{s.t.} \begin{cases} \sum_{(i,v)\in E} f_{iv} - \sum_{(v,j)\in E} f_{vj} = 0, & \forall v \in V \setminus \{s,t\}, \\ 0 \leq f_{ij} \leq u_{ij}, & \forall (i,j) \in E. \end{cases}$$

- Decision variables: f_{ij} = amount of flow on edge (i,j).
- Objective: maximize net flow leaving s (equals entering t).
- Constraints: capacity limits and flow conservation.

The Dual: Minimum s–t Cut

Introduce dual variables:

- π_v for each vertex conservation constraint (potential or "height").
- $\lambda_{ij} \geq 0$ for each capacity constraint $f_{ij} \leq u_{ij}$.

The dual LP becomes

$$\min \sum_{(i,j)\in E} u_{ij}\,\lambda_{ij}$$

$$\text{s.t. } \pi_i - \pi_j + \lambda_{ij} \geq 0 \quad \forall (i,j) \in E,$$

$$\pi_s - \pi_t \geq 1, \quad \lambda_{ij} \geq 0.$$

- π encodes a potential difference between s and t.
- $\lambda_{ij} > 0$ only on edges where the inequality is tight — these edges "cross the cut".

Dual $\Rightarrow$ a Cut; Equality via Strong Duality

From the dual constraints:

$$\pi_i - \pi_j + \lambda_{ij} \geq 0$$

we can take $\pi_v \in \{0, 1\}$ (thresholding the potentials):

$$\pi_i - \pi_j = \begin{cases} 1 & \text{if } i \in S, j \in T \\ 0 & \text{otherwise} \end{cases} \quad \Rightarrow \quad \lambda_{ij} = \begin{cases} 1 & \text{if } (i,j) \in \delta^+(S) \\ 0 & \text{else.} \end{cases}$$

Then

$$\min \sum_{(i,j)\in E} u_{ij}\lambda_{ij} = \sum_{(i,j)\in\delta^+(S)} u_{ij} = \text{capacity of the cut } (S, T).$$

By strong duality: max flow $=$ min cut.

Feasible flow $\Rightarrow$ lower bound; feasible cut $\Rightarrow$ upper bound; when they meet, we have optimality

References

Erickson, J. (2019).
Algorithms.
Self-published.