MATH 409 Notes

Contents

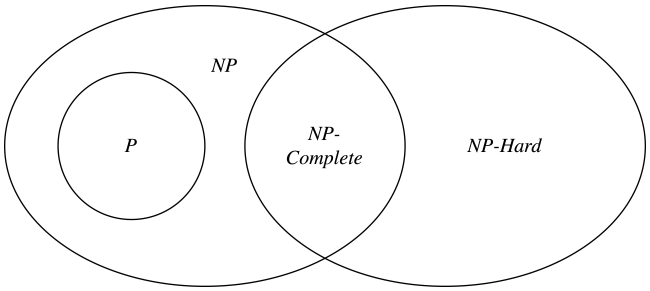
1 Algorithms and Complexity

1.1 Big-O Notation

$f(n)$ is $O(g(n))$ if there exists positive constants c, n_0 such that for all $n \geq n_0$, $f(n) \leq c \cdot g(n)$

- Big-O represents an upper bound for the algorithm run time

1.2 Complexity Classes



1.3 Polynomial

P is the set of all decision problems that have an algorithm that runs in time $O(n^k)$ for some constant k

- All P problems are NP problems

1.4 Non-Deterministic Polynomial

NP is the set of all decision problems such that if the answer is yes, there is a proof which can be verified in polynomial time

- NP is a superset of P

1.5 Polynomial Time Reducible

Problem A reduces to problem B in polynomial time if there exists an algorithm that, using a hypothetical polynomial time algorithm for B, solves A in polynomial time

- Problem A can be translated to problem B and solved using a polynomial time algorithm for B
- This means that problem A is easier than problem B

1.6 NP-Complete

A problem A is NP-complete if A is in NP and all problems in NP reduce to A in polynomial time

- An NP-complete problem is the hardest problem in NP
- A polynomial time algorithm for an NP-complete problem can be used to solve every problem in NP in polynomial time via reduction

2 Graph Theory

2.1 Undirected Graphs

An undirected graph is a graph where the edges do not have direction

- Every undirected graph with n vertices has at least $n-1$ edges and at most $\frac{1}{2}n(n-1)$ edges

2.2 Complete Graphs

A complete graph is an undirected graph where every vertex is connected to every other vertex by a unique edge

2.3 Subgraphs

A subgraph is a graph whose vertices and edges are subsets of another graph

2.4 Cuts

A cut is any partition of the vertices in a graph into two disjoint sets of vertices, denoted (A, B)

- The vertices in each set are not necessarily connected to each other

2.5 Trees

A tree is an undirected acyclic connected graph

- A tree has exactly $n-1$ number of edges

2.6 Forests

A forest is an undirected acyclic graph whose connected components are trees

2.7 Spanning Trees

A tree is a spanning tree of a graph if it spans all vertices in the graph and consists of only edges within the graph

- If $T = (V', E')$ is a spanning tree of $G = (V, E)$, then $V' = V$, $|E'| = |V'| - 1$, and $E' \subseteq E$

2.8 Matchings

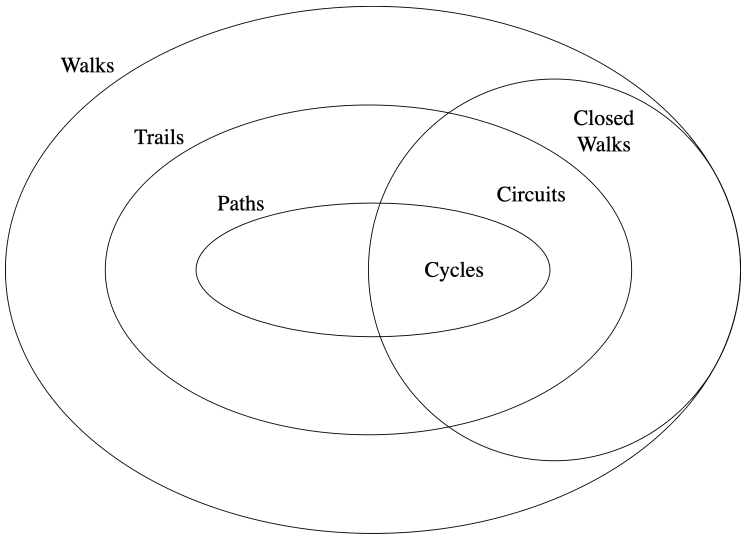
A matching is a set of edges in a graph such that every vertex has degree less than or equal to 1

2.9 Perfect Matchings

A perfect matching is a set of edges in a graph such that every vertex has degree exactly 1

- If a matching is perfect, then it is maximal
- Let G be a bipartite graph with a perfect matching. Then a matching is perfect if and only if it is maximal

2.10 Vertex Sequences



2.11 Walks

A walk is a sequence of adjacent vertices in a graph where vertices and edges may be repeated

- If a walk starts and ends at the same vertex, then it is closed
- If a walk starts and ends at different vertices, then it is open

2.12 Trails

A trail is a sequence of adjacent vertices in a graph where vertices may be repeated but edges are distinct

2.13 Paths

A path is a sequence of adjacent vertices in a graph where vertices and edges are distinct

2.14 Circuits

A circuit, also known as a tour, is a sequence of adjacent vertices that starts and ends at the same vertex where vertices may be repeated but edges are distinct

- A circuit is a closed trail

2.15 Cycles

A cycle is a sequence of adjacent vertices that starts and ends at the same vertex where vertices and edges are distinct

- A cycle is a closed path

2.16 Hamiltonian Cycle

A Hamiltonian cycle is a spanning cycle

2.17 Traveling Salesman Problem

Given an undirected weighted graph, find the smallest cost Hamiltonian cycle

2.18 Held-Karp's Traveling Salesman Algorithm

Held-Karp algorithm

Input: Distances $c_{ij} \geq 0$ in a complete n-node graph for $n \geq 3$

Output: Cost of an optimum TSP tour

(1) $C(\{i,j\}, i, j) := c_{ij}$ for all $i \neq j$

(2) FOR $k = 3$ TO n DO

(3) FOR ALL SETS $S \subseteq V : |S| = k$ DO

$$C(S, i, j) := \min_{\ell \in S \setminus \{i,j\}} \{C(S \setminus \{j\}, i, \ell) + c_{\ell,j}\}$$

(4) output the optimum cost $\min_{j \neq 1}\{C(V, 1, j) + c_{j1}\}$

- $C(S, i, j)$ is the shortest path from vertex i to vertex j that passes through every vertex in S
- Held-Karp has $O(2^n n^2)$ running time, where n is the number of vertices in the graph

3 Minimum Spanning Trees

3.1 Minimum Spanning Tree Problem

Given an undirected weighted graph, find the smallest cost spanning tree

3.2 Spanning Tree Properties

Given a graph $G = (V, E)$ with $|V| = n$ vertices

- G has at least $n - 1$ edges
- G has $n - 1$ edges if and only if G is acyclic
- The edges $T \subseteq E$ form a spanning tree if and only if $|T| = n - 1$
- The edges $T \subseteq E$ form a spanning tree if and only if there exists exactly one path from u to v for each pair of vertices $u, v \in V$

3.3 Spanning Tree Edge Swap Argument

Let $T \subseteq E$ be a spanning tree in $G = (V, E)$ and select an edge $e' = (u, v)$ that is not in the spanning tree. Then for any edge e on the path from u to v within the spanning tree, we can construct a new spanning tree $(T \backslash \{e\}) \cup \{e'\}$

- T is a minimum spanning tree if and only if $\text{cost}(T) \leq \text{cost}(T')$ for any edge swap

3.4 Connected Component Relations

Let $G = (V, E)$ be a disconnected graph with connected components $V_1, ..., V_k$ and let u, v be vertices in G. Then $u \sim v$ if and only if there exists a path from u to v in G

- This relation is reflexive, symmetric and transitive
- The equivalence class of this relation represents a connected component

3.5 Kruskal's Minimum Spanning Tree Algorithm

Kruskal's Algorithm

Input: A connected graph G with edge costs $c_e \in \mathbb{R}$, $\forall\ e \in E(G)$.
Output: A MST T of G.
(1) Sort the edges such that $c_{e_1} \leq c_{e_2} \leq \ldots \leq c_{e_m}$.
(2) Set $T = \emptyset$
(3) For i from 1 to m do
If $T \cup \{e_i\}$ is acyclic then update $T := T \cup \{e_i\}$.

- At each iteration, Kruskal's algorithm adds the smallest edge that expands the currently reachable set
- Kruskal's algorithm has $O(|E| \log |V|)$ running time

4 Shortest Paths

4.1 Shortest Path Problem

Given a weighted directed graph and two vertices s, t, find the smallest cost s-t path

- Strong assumption: all edges have non-negative cost
- Weak assumption: all cycles have non-negative cost

4.2 Bellman's Principle

Let $G = (V, E)$ be a weighted directed graph with no negative cost cycles and let

$$P_{s,t} = \{(s, v_1), (v_1, v_2), ..., (v_k, t)\}$$

be the shortest s-t path. Then $P_{s,i} = \{(s, v_1), (v_1, v_2), ..., (v_{i-1}, v_i)\}$ is the shortest s-v_i path

4.3 Dijkstra's Shortest Path Algorithm

Dijkstra's algorithm (Dijkstra 1959)

Input: A directed graph $G = (V, E)$ with edge costs c_e. A source vertex s.
Assumption: $c_e \geq 0 \; \forall e \in E$
Output: Length $\ell(v)$ of shortest s-v-paths for all $v \in V$.

(1) Set $\ell(s) = 0$. $R = \{s\}$ and for all $v \in V \backslash \{s\}$: $\ell(v) = \begin{cases} c(s,v) & \text{if } (s,v) \in E \\ \infty & \text{otherwise} \end{cases}$

(2) WHILE $R \neq V$ DO

  (3) Select $v \in V \backslash R$ attaining $\min\{\ell(v) \mid v \in V \backslash R\}$

  (4) FOR all $w \in V \backslash R$ with $(v, w) \in E$ DO $\ell(w) := \min\{\ell(w), \ell(v) + c(v, w)\}$

  (5) Set $R = R \cup \{v\}$.

- Dijkstra's algorithm makes use of the strong assumption
- Dijkstra's algorithm has $O(|E| + |V| \log |V|)$ running time

4.4 Bellman-Ford Algorithm

Bellman-Ford Algorithm

Input: a directed graph $G = (V, E)$; edge cost $c(e) \in \mathbb{R}$ $\forall e \in E$; a source node $s \in V$.
Assumption: There are no negative cost cycles in G
Output: The length $\ell(v)$ of the shortest s-v path, for all $v \in V$.

(1) Set $\ell(s) := 0$ and $\ell(v) = \infty$ for all $v \in V \backslash \{s\}$
(2) For $i = 1, \ldots, n-1$ do
    (3) For each $(v, w) \in E$ do:
        If $\ell(w) > \ell(v) + c_{vw}$ then set $\ell(w) = \ell(v) + c_{vw}$ ("update edge (u, w)")

- At the i^{th} iteration, $\ell(v)$ represents the length of the shortest s-v path with at most i edges
- Bellman-Ford outputs the shortest distance to all vertices from the source
- Bellman-Ford is a shortest path algorithm that works on graphs with negative edge weights
- Bellman-Ford has $O(nm)$ running time, where m is the number of edges

4.5 Reduced Costs

Let G be a directed graph with costs $c_e \in \mathbb{R}$ for all edges $e \in E$ and let $\pi : V \to R$ be a function with value $\pi(v) \in \mathbb{R}$ for all vertices $v \in V$

- The reduced cost of an edge $(i, j) \in E$ with respect to π is $c_{ij} + \pi(i) - \pi(j)$
- π is a feasible potential on G if the reduced costs of all edges in G are non-negative

4.6 Conservative Costs

Let G be a directed graph. Then the vector of costs c is conservative if there are no negative cost cycles in G

4.7 Detecting Negative Cycles

The directed graph G with vector of costs c has a feasible potential if and only if c is conservative

1. Modify the Bellman-Ford algorithm such that the outer for-loop runs for n iterations instead of $n - 1$ iterations
2. Run the modified Bellman-Ford algorithm on the graph
3. If no edge update is made in the last iteration, then $\pi(v) = \ell(v)$ defines a feasible potential on G such that c is conservative

5 Network Flows

5.1 Network Flow Problem

Given a directed weighted graph, find the maximal s-t flow that respects edge capacities and has incoming flow equal to outgoing flow at each of the vertices

5.2 Network Flow Terminology

- A network (G, u, s, t) consists of a directed graph $G = (V, E)$, edge capacities $u_e \in \mathbb{Z}_{\geq 0}$ for all $e \in E$, a source vertex s, and a sink vertex t
- An s-t flow is a function $f : E \to \mathbb{R}_{\geq 0}$ with the following properties
  - The flow respects the capacities, that is $0 \leq f(e) \leq u_e$
  - The flow satisfies flow conservation at each vertex $v \in V \backslash \{s, t\}$

$$\sum_{e \in \delta^-(v)} f(e) = \sum_{e \in \delta^+(v)} f(e)$$

where $\delta^-(v)$ is the set of edges entering v and $\delta^+(v)$ is the set of edges leaving v

5.3 Ford-Fulkerson Algorithm

Ford and Fulkerson's algorithm for Max Flows

Input: A network (G, u, s, t).
Output: A max (s, t)-flow in the network.

(1) Set $f(e) = 0$ for all $e \in E$.
(2) REPEAT
  (3) Find an f-augmenting path P. If none exist then stop.
  (4) Compute $\gamma := \min\{u_f(e) \mid e \in P\}$.
  (5) Augment f along P by γ.

- Ford-Fulkerson outputs the maximum flow of a graph with integer edge weights
- Ford-Fulkerson has $O(m \cdot \mathrm{OPT})$ running time, where $m = |E|$ and OPT is the max flow
- An f-augmenting path can be found using any algorithm that finds a path from s to t

5.4 Edmonds-Karp Algorithm

The Edmonds-Karp algorithm is implemented the same way as the Ford-Fulkerson algorithm, except that `findPath()` is implemented using breadth first search

- Edmonds-Karp has $O(m^2 n)$ running time, where m is the number of edges and n is the number of vertices

5.5 Residual Graph

The residual graph is an auxiliary graph generated by the Ford-Fulkerson algorithm that indicates the residual capacity and current flow of each edge

- Given two adjacent vertices u and v in the residual graph, $\mathrm{weight}(u, v)$ denotes the residual capacity and $\mathrm{weight}(v, u)$ denotes the current flow of the edge (u, v)

5.6 Minimum s-t Cut

Given a directed graph with a source and a sink, find a cut that minimizes the total weight of the edges going across the partition from the s-subset to the t-subset

- The max-flow min-cut theorem states that the value of the max s-t flow is equal to the value of the min s-t cut

5.7 Network Flow Reductions

Network flow reductions are a class of problems that can be reduced to a network flow problem and solved using the Ford-Fulkerson algorithm

- Bipartite Matching Problem
  Given an undirected bipartite graph, find a maximal matching
  - Given an undirected bipartite graph (A, B), construct a directed graph as follows
    - Assign infinite capacity to all edges from A to B
    - Add a source s and unit weight edges from s to each node in A
    - Add a sink t and unit weight edges from each node in B to t
  - Run Ford-Fulkerson on the constructed graph
  - The max-flow of the constructed graph corresponds to the maximal matching of the original graph
    - If there exists a perfect matching, then the maximal matching is perfect
- Edge Disjoint Paths Problem
  Given a directed graph with a source and a sink, find the maximum number of s-t paths such that none of them share common edges
  - Given a directed graph, construct a directed graph such that all edges have unit weight
  - Run Ford-Fulkerson on the constructed graph
  - The max-flow of the constructed graph corresponds to the maximum number of edge disjoint paths in the original graph

5.8 König's Theorem

Given a bipartite graph, the number of edges in a maximum matching equals the number of vertices in a minimum vertex cover

- Let $S \subseteq V$ be a minimum s-t cut. Then $U = (V_1 \backslash S) \cup (V_2 \cap S)$ is a vertex cover for G
- A vertex cover is a set of vertices such that every edge is incident to at least one vertex

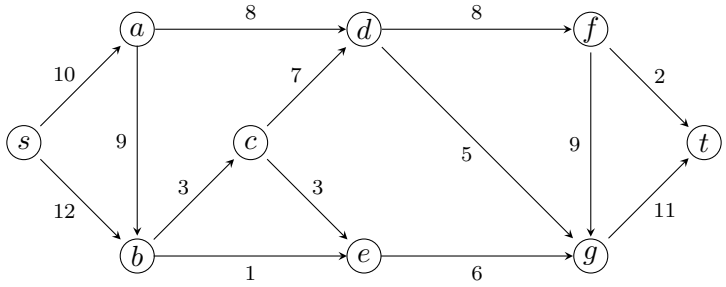
5.9 Hall's Theorem

Given a bipartite graph $G = (V_1 \cup V_2, E)$ with $|V_1| = |V_2| = n$, G has a perfect matching if and only if $|N(U)| \geq |U|$ for all $U \subseteq V_1$

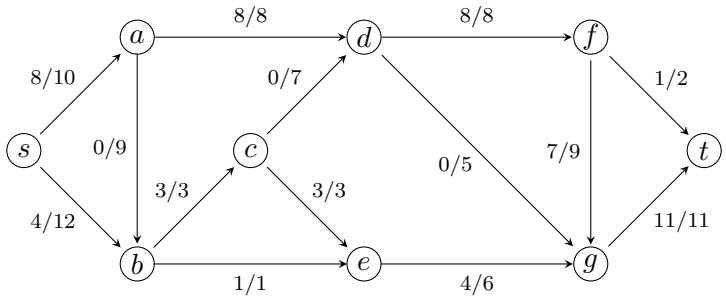
- $N(U) = \{v \in V_2 \mid \exists (u, v) \in E,\ u \in U\}$ is the neighborhood of U
  - The neighborhood of U is the set of vertices adjacent to at least one vertex in U

5.10 Network Flow Example

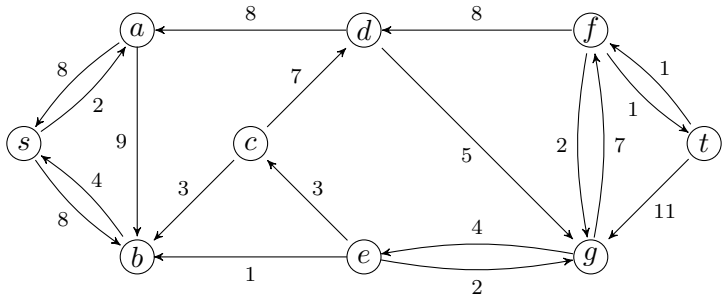
Suppose we are given the following graph



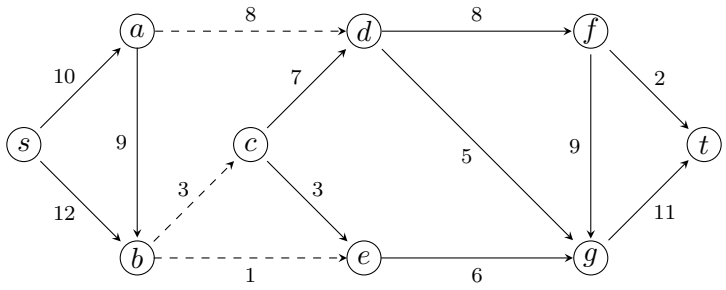
The maximum s-t flow for the graph is as follows



The corresponding residual graph for this maximum flow is as follows



The minimum cut for this maximum flow is as follows



6 Linear Programming

6.1 Linear Functions

A linear function $f : \mathbb{R}^n \to \mathbb{R}$ is a function of the form

$$f(x) = c^T x = c_1 x_1 + ... + c_n x_n$$

where $c \in \mathbb{R}^n$

6.2 Polyhedrons

A polyhedron $P \subseteq \mathbb{R}^n$ is the set of all points $x \in \mathbb{R}^n$ that satisfy a set of linear inequalities

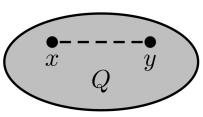
$$P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$$

where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$

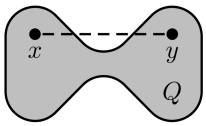
- A polytope is a bounded polyhedron, that is it can be enclosed in a ball of finite radius
- All polyhedrons are convex

6.3 Convexity

A set $Q \subseteq \mathbb{R}^n$ is convex if for any two points $x, y \in Q$, the line segment joining them is also in Q



convex

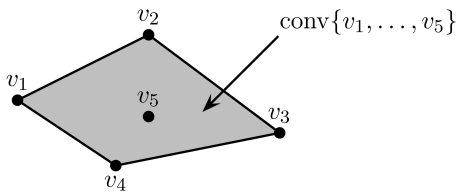


not convex

6.4 Convex Combinations

A convex combination of a finite set of points $v_1, ..., v_t \in \mathbb{R}^n$ is any vector of the form $\sum_{i=1}^{t} \lambda_i v_i$ such that $\lambda_i \in [0, 1]$ for all $i = 1, ..., t$ and $\sum_{i=1}^{t} \lambda_i = 1$

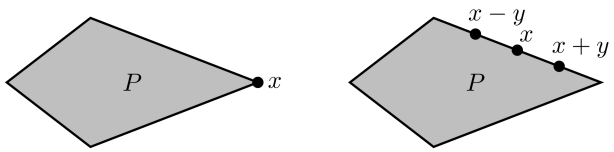
- The set of all convex combinations of $v_1, ..., v_t$ is called its convex hull
  - $\text{conv}\{v_1, ..., v_t\} = \left\{ \sum_{i=1}^{t} \lambda_i v_i \;\middle|\; \lambda_1 + ... + \lambda_t = 1,\ \lambda_i \geq 0,\ \forall i = 1, ..., t \right\}$
- Every polytope P is the convex hull of a finite number of points and vice versa



6.5 Extreme Points

A point x in a convex set P is extreme if there is no vector $y \in \mathbb{R}^n \backslash \{0\}$ with both $x + y \in P$ and $x - y \in P$

- A point $x \in P$ is extreme if it is at the vertex of the convex set P



x is extreme point $\quad$ x is NOT extreme point

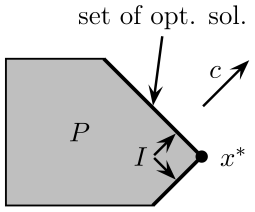
6.6 Linear Programs

A linear program is the problem of maximizing or minimizing a linear function of the form $\sum_{i=1}^{n} c_i x_i$ over all $x = (x_1, ..., x_n)$ in a polyhedron P

$$\max \left\{ \sum_{i=1}^{n} c_i x_i \;\middle|\; Ax \leq b \right\}$$

where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$

- A linear program is always maximized at an extreme point x^* where x^* is the unique solution to the system of linear equations ${A_i}^T x = b_i$ for all $i \in I$ where $I \subseteq \{1, ..., m\}$ with $|I| = n$
- The optimum point of a linear program in $\mathbb{R}^n$ is an extreme point where n linearly independent constraints are tight



6.7 Duality

Let $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^{n \times 1}$, $c \in \mathbb{R}^{n \times 1}$, and $b \in \mathbb{R}^{m \times 1}$. Then the primal and dual linear programs are

$$\begin{aligned} \text{primal}: &\quad \max \left\{ c^T x \mid Ax \leq b \right\} \\ \text{dual}: &\quad \min \left\{ b^T y \mid y^T A = c^T,\ y \geq 0 \right\} \end{aligned}$$

6.8 Weak Duality Theorem

If x and y are feasible solutions for the primal and dual problems, then $c^T x \leq b^T y$

- The Weak Duality Theorem states that any solution of the primal is less than or equal to any solution of the dual

6.9 Hyperplane Separation Theorem

Let $P, Q \subseteq \mathbb{R}^n$ be convex, closed and disjoint sets with at least one of them bounded. Then there exists $c \in \mathbb{R}^n$ and $\beta \in \mathbb{R}$ with

$$c^T x < \beta < c^T y$$

for all $x \in P$ and for all $y \in Q$

- $c^T x = \beta$ represents a hyperplane separating P and Q

6.10 Farkas' Lemma

Let $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. Then exactly one of the following is true

- There exists $x \in \mathbb{R}^n$ such that $Ax = b$ and $x \geq 0$
- There exists $y \in \mathbb{R}^m$ such that $A^T y \geq 0$ and $b^T y < 0$

6.11 Strong Duality Theorem

If x and y are optimal solutions for the primal and dual problems, then $c^T x = b^T y$

$$\max \left\{ c^T x \mid Ax \leq b \right\} = \min \left\{ b^T y \mid y^T A = c^T,\ y \geq 0 \right\}$$

- The Strong Duality Theorem states that the maximal solution of the primal is equal to the minimal solution of the dual

6.12 Complimentary Slackness Theorem

Let (x^*, y^*) be feasible solutions for the primal and dual problems. Then (x^*, y^*) are simultaneously optimal if and only if

- For all $1 \leq i \leq m$, either ${A_i}^T x^* = b_i$ or ${y_i}^* = 0$ or both hold
  - If ${A_i}^T x^* \neq b_i$, then ${y_i}^* = 0$
  - If ${y_i}^* \neq 0$, then ${A_i}^T x^* = b_i$
- For all $1 \leq j \leq n$, either ${y^*}^T A_j = c_j$ or ${x_j}^* = 0$ or both hold
  - If ${y^*}^T A_j \neq c_j$, then ${x_j}^* = 0$
  - If ${x_j}^* \neq 0$, then ${y^*}^T A_j = c_j$

6.13 Simplex Algorithm

Simplex algorithm

Input: $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m, c \in \mathbb{R}^n$ and a starting basis $I \in \binom{[m]}{n}$
Output: opt. solution x attaining $\max\{c^T x \mid Ax \leq b\}$

(1) $x = A_I^{-1} b_I$

(2) IF $y := cA_I^{-1} \geq \mathbf{0}$ THEN RETURN x is optimal

(3) select $j \in I$ and $j' \notin I$ so that for $I' := I \backslash \{j\} \cup \{j'\}$ the following 3 conditions are satisfied

(i) $\text{rank}(A_{I'}) = n$
(ii) the point $x' = A_{I'}^{-1} b_{I'}$ lies in P
(iii) $c^T x' \geq c^T x$

(4) UPDATE $I := I'$ and GOTO (1)

- $\binom{[m]}{n}$ denotes the set of all $I \subseteq \{1, ..., m\}$ with $|I| = n$
- Simplex has very fast practical performance
- Simplex has $O(2^n)$ worst-case running time, where n is the number of variables

6.14 Linear Program Reductions

Linear program reductions are a class of problems that can be reduced to a linear program and solved using the Simplex algorithm

- Bipartite Perfect Matching Problem
  Given an undirected bipartite graph, find a perfect matching maximizing $\sum_{e \in M} c_e$
  - Construct the following linear program

$$\max \left\{ \sum_{e \in E} c_e x_e \;\middle|\; \sum_{e \in \delta(v)} x_e = 1,\ x_e \geq 0,\ \forall v \in V,\ \forall e \in E \right\}$$

    where $\delta(v)$ is the set of edges incident to a node
  - Run the Simplex algorithm on the linear program
  - When x is an optimum solution, $M = \{e \in E \mid x_e = 1\}$ represents a perfect matching
    - If there does not exist a perfect matching, then the linear program is infeasible

6.15 Integer Hulls

Let $P \subseteq \mathbb{R}^n$ be a polyhedron. Then an integer hull $P_I \subseteq P$ is the convex hull of all integer points in P

$$P_I = \text{conv}\{P \cap \mathbb{Z}^n\}$$

where $\mathbb{Z}^n$ is the n-dimensional integer space

- If $P \subseteq \mathbb{R}^n$ is a polytope, then the following are equivalent
  - $P = P_I$
  - $\max\{c^T x \mid x \in P\} = \max\{c^T x \mid x \in P_I\}$ for all $c \in \mathbb{R}^n$

7 Total Unimodularity

7.1 Full Rank Matrices

Let A be an $m \times n$ matrix

- If $m < n$, then A has full rank if its rows are linearly independent
  - If $m < n$, then the columns are guaranteed to be linearly dependent
- If $m > n$, then A has full rank if its columns are linearly independent
  - If $m > n$, then the rows are guaranteed to be linearly dependent
- If $m = n$, then A has full rank if its rows and columns are linearly independent
  - A has full rank if and only if $\det(A) \neq 0$
  - The rows are linearly independent if and only if the columns are linearly independent

7.2 Extreme Points in Polyhedrons

Let $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ be a polyhedron with $A \in \mathbb{R}^{m \times n}$. For each extreme point x of P, there are row indices $I \subseteq \{1, ..., m\}$ such that A_I is an $n \times n$ matrix of full rank and x is the unique solution to $A_I x = b_I$

7.3 Cramer's Rule

Let $B = (b_{ij})_{i,j \in \{1,...,n\}}$ be an $n \times n$ square matrix of full rank. Then the system $Bx = b$ has exactly one solution x^* where

$$x_i{}^* = \frac{\det(B^i)}{\det(B)}$$

where B^i is the matrix B with the i^{th} column replaced by b

7.4 Totally Unimodular Matrices

Let A be an $m \times n$ matrix. Then A is totally unimodular (TU) if every square submatrix of A has determinant $0, 1$, or -1

- A matrix $A \in \{-1, 0, 1\}^{m \times n}$ is totally unimodular if it can be written in the form

$$A = \left(\begin{array}{c|c} B & \begin{array}{c} C_1 \\ \hline C_2 \end{array} \end{array} \right)$$

  where B has at most one non-zero entry per column, C_1 has exactly one $+1$ entry per column, and C_2 has exactly one -1 entry per column
- A matrix with the consecutive-ones property is totally unimodular
  - The consecutive-ones property holds if the ones in each column are consecutive
- The node-edge incidence matrix $A \in \{0, 1\}^{|V| \times |E|}$ of an undirected bipartite graph is totally unimodular
- The node-edge incidence matrix $A \in \{-1, 0, 1\}^{|V| \times |E|}$ of a directed graph is totally unimodular
- We can check whether a matrix $A \in \{-1, 0, 1\}^{m \times n}$ is totally unimodular in polynomial time

7.5 Preserving Total Unimodularity

Let $A \in \{-1, 0, 1\}^{m \times n}$ be a totally unimodular matrix. Then the following operations preserve the total unimodularity of A

- Multiplying a row/column with -1
- Permuting rows/columns
- Adding/removing a row/column that has at most one ± 1 entry and 0's otherwise
- Duplicating rows
- Transposing the matrix

7.6 Totally Unimodular Polyhedrons

Let $A \in \{-1, 0, 1\}^{m \times n}$ be a totally unimodular matrix, let $b \in \mathbb{Z}^m$ and let $u, v \in \mathbb{Z}^n$

- All extreme points of $\{x \in \mathbb{R}^n \mid Ax \leq b\}$ are integral
- All extreme points of $\{x \in \mathbb{R}^n \mid Ax \leq b,\ x \geq 0\}$ are integral
- All extreme points of $\{x \in \mathbb{R}^n \mid Ax = b,\ x \geq 0\}$ are integral
- All extreme points of $\{x \in \mathbb{R}^n \mid Ax \leq b,\ u \leq x \leq v\}$ are integral

7.7 Totally Unimodular Linear Programs

Let $A \in \{-1, 0, 1\}^{m \times n}$ be a totally unimodular matrix, let $b \in \mathbb{Z}^m$ and let $c \in \mathbb{Z}^n$. Then there are optimum integral solutions x^*, y^* to

$$\max\{c^T x \mid Ax \leq b,\ x \geq 0\} \qquad \text{and} \qquad \min\{b^T y \mid A^T y \geq c,\ y \geq 0\}$$

with $c^T x^* = b^T y^*$

7.8 Totally Unimodular Reductions

Totally unimodular reductions are a class of problems that can be reduced to a totally unimodular linear program and solved using the Simplex algorithm

- Bipartite Matching Problem
  Given an undirected bipartite graph, find a perfect matching maximizing $\sum_{e \in M} c_e$
  - Construct the following totally unimodular linear program

$$\max \left\{ \sum_{e \in E} c_e x_e \;\middle|\; Ax = 1,\ x \geq 0 \right\}$$

    where $A_{v,e} = \begin{cases} 1 & v \text{ incident to } e \\ 0 & \text{otherwise} \end{cases}$ such that A is totally unimodular
  - Run the Simplex algorithm on the linear program
  - When x is an optimum solution, $M = \{e \in E \mid x_e = 1\}$ represents a perfect matching
    - If there does not exist a perfect matching, then the linear program is infeasible
- Minimum Cost Integer Circulation Problem
  Given a directed weighted graph with integral edge capacities and real edge costs, find the cheapest s-t flow of value k
  - Construct the following totally unimodular linear program

$$\min \left\{ c^T f \;\middle|\; Af = b,\ 0 \leq f(e) \leq u(e),\ \forall e \in E \right\}$$

    where $A_{v,e} = \begin{cases} 1 & e \in \delta^-(v) \\ -1 & e \in \delta^+(v) \\ 0 & \text{otherwise} \end{cases}$ such that A is totally unimodular and $b_v = \begin{cases} -k & v = s \\ k & v = t \\ 0 & \text{otherwise} \end{cases}$
  - Run the Simplex algorithm on the linear program
  - When f is an optimal solution, f_e represents an optimal flow through edge e
- Interval Scheduling
  Given a list of intervals $I_1, ..., I_n \subseteq \mathbb{R}$ with profits $c_1, ..., c_n$, find a disjoint subset of intervals that maximizes the sum of profits
  - Construct the following totally unimodular linear program

$$\max \left\{ c^T x \;\middle|\; Ax \leq 1,\ 0 \leq x \leq 1 \right\}$$

    where $A_{ji} = \begin{cases} 1 & t_j \in I_i \\ 0 & \text{otherwise} \end{cases}$ such that A satisfies the consecutive-ones property and is totally unimodular
  - Run the Simplex algorithm on the linear program
  - When x is an optimum solution, $M = \{I_i \subseteq \mathbb{R} \mid x_i = 1\}$ represents an optimal schedule

8 Branch & Bound

8.1 Branch & Bound Algorithm

Branch & Bound algorithm

Input: $c \in \mathbb{R}^n, A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$
Output: An optimum solution to $\max\{c^T x \mid Ax \leq b;\ x \in \mathbb{Z}^n\}$

(1) Set $x^* :=$ `UNDEFINED` *(best solution found so far)*
(2) Put problem $P := \{x \mid Ax \leq b\}$ on the stack
(3) WHILE stack is non-empty DO
  (4) Select a polytope P' from the stack
  (5) Solve $\max\{c^T x \mid x \in P'\}$ and denote LP solution by $\tilde{x}$
  (6) IF $P' = \emptyset$ THEN goto (3) *("prune by infeasibility")*
  (7) IF $c^T x^* \geq c^T \tilde{x}$ THEN goto (3) *("prune by bound")*
  (8) IF $c^T x^* < c^T \tilde{x}$ and $\tilde{x} \in \mathbb{Z}^n$ THEN *(let $c^T x^* = -\infty$ if $x^* =$ `UNDEFINED`*
    update $x^* := \tilde{x}$ and goto (3) *("prune by optimality")*
  (9) Otherwise (i.e. $\tilde{x} \notin \mathbb{Z}^n, c^T x^* < c^T \tilde{x}$) do *("branch")*
    (10) Select coordinate i with $\tilde{x}_i \notin \mathbb{Z}$ that maximizes $\min\{\tilde{x}_i - \lfloor \tilde{x}_i \rfloor, \lceil \tilde{x}_i \rceil - \tilde{x}_i\}$
    (11) Add problems $P' \cap \{x \mid x_i \leq \lfloor \tilde{x}_i \rfloor\}$ and $P' \cap \{x \mid x_i \geq \lceil \tilde{x}_i \rceil\}$ to stack

- The stack is either a LIFO stack or max-priority queue ordered on $\tilde{x} = \max\{c^T x \mid x \in P'\}$
- Branch & bound has $O(2^{\frac{n}{3}})$ worst-case running time, where n is the number of leafs in the search tree

8.2 Pathological Instances

Let $n \geq 4$ be an even integer. Then the search tree for the following integer linear program has at least $2^{\frac{n}{3}}$ leafs

$$\max \left\{ x_0 \;\middle|\; \frac{1}{2} x_0 + \sum_{i=1}^{n} x_i = \frac{1}{2} n,\ x \in \{0,1\}^{n+1} \right\}$$

9 Non-Bipartite Matching

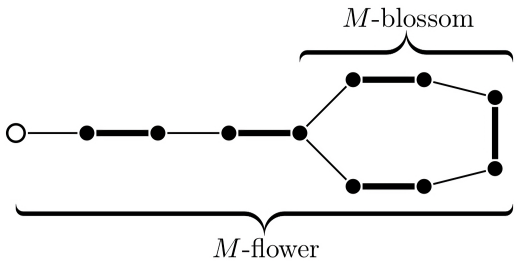
9.1 Maximum Cardinality Matching Problem

Given an undirected graph $G = (V, E)$, find a matching $M \subseteq E$ of maximum size $|M|$

9.2 Maximum Cardinality Matching Terminology

Let M be a matching in an undirected graph

- An M-exposed node is a node that is not incident to any edge in M
- An M-alternating walk is a walk whose edges are alternately in M and not in M
  - An M-alternating walk may have repeating vertices and edges
- An M-augmenting path is an M-alternating walk that starts and ends at different M-exposed vertices
  - An M-augmenting path must have distinct vertices and edges
- An M-flower is an M-alternating walk that satisfies the following properties
  - Starts at an M-exposed vertex u
  - Revisits exactly one vertex v once and ends at v
  - The cycle that is closed at v has an odd number of edges
    - This cycle is called an M-blossom
  - The path between u and v has an even number of edges
    - This path is called the stem



9.3 Augmenting Path Theorem for Matchings

A matching M in a graph is maximum if and only if there is no M-augmenting path in the graph

- If there exists an M-augmenting path $P = E \cup F$ in the graph where $E \subseteq M$ and $F \cap M = \varnothing$, then we can generate a new matching $M' = (M \backslash E) \cup F$
  - Every M-augmenting path has 1 more edge not in M than in M

9.4 Computing Augmenting Paths

Given an undirected graph $G = (V, E)$ and matching M, construct a directed bipartite graph as follows

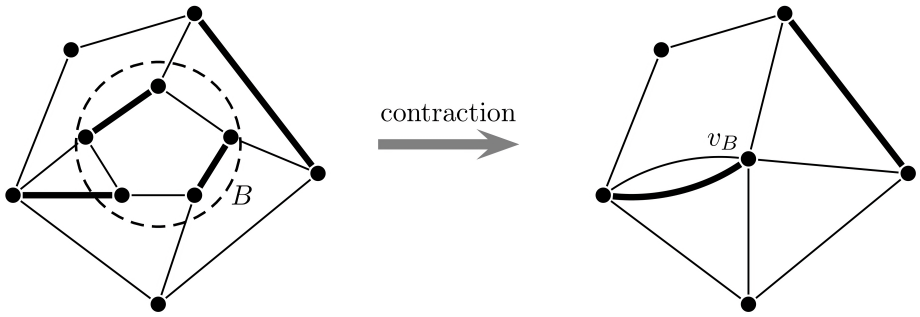
- For all vertices $v \in V$, create a vertex v in the left part and a vertex v' in the right part
- For all edges $(u, v) \in E \backslash M$, create two forwards directed edges (u, v') and (v, u')
- For all edges $(u, v) \in M$, create two backwards directed edges (u', v) and (v', u)

Observe that any path in the constructed bipartite graph corresponds to an M-alternating walk

- Run Dijkstra's shortest path algorithm on the constructed bipartite graph from each M-exposed vertex to find the shortest M-alternating walk starting and ending at an exposed vertex
  - This has $O(nm + n^2 \log n)$ running time, where $n = |V|$ and $m = |E|$
- If Dijkstra finds an M-alternating walk, then it is either an M-augmenting path or an M-flower
- If Dijkstra cannot find an M-alternating walk, then there is no M-augmenting path

9.5 Contracting Blossoms

Let $G = (V, E)$ be an undirected graph and let $B \subseteq E$ be the edges of the blossom. Then $G \times B$ represents the graph obtained by contracting all the nodes in B into a single super-node v_B



graph G with blossom B contracted graph $G \times B$

- Let M be a matching and B be an M-blossom. Then $M \backslash B$ is a matching in $G \times B$ and any $M \backslash B$ augmenting path in $G \times B$ can be extended to an M-augmenting path in G
- Let M be a matching and B be an M-blossom. If there is an M-augmenting path in G, then there is an $M \backslash B$ augmenting path in $G \times B$

9.6 Blossom Algorithm for Maximum Matching

Blossom algorithm for maximum matching (Edmonds 1965)

Input: Undirected graph $G = (V, E)$
Output: Maximum matching $M \subseteq E$.

(1) $M := \emptyset$
(2) REPEAT
  (3) Call Subroutine to find M-augmenting path P
  (4) IF $P =$ `FAIL` THEN Return M ELSE Update $M := M \Delta P$

SUBROUTINE:
Input: Undirected graph $G = (V, E)$, matching $M \subseteq E$
Output: M-augmenting path.

(1) Construct bipartite graph G' and compute shortest path P from exposed node to exposed node
(2) IF no such path exists THEN return `FAIL`
(3) IF P is M-augmenting path THEN return P
(4) IF P corresponds to M-flower with blossom B THEN
  (5) Call subroutine on $G \times B$ and $M \backslash B$
  (6) Extend the returned path by edges in B

- Blossom has $O(n^3 m + n^4 \log n)$ running time, where $n = |V|$ and $m = |E|$
  - We need to find an M-augmenting path at most n times
  - While we try to find an M-augmenting path, we may need to contract n times
  - Before contracting, we need to find the shortest M-alternating walk

10 The Knapsack Problem

10.1 Knapsack Problem

Given a set of items $\{1, ..., n\}$ with weights $w_1, ..., w_n \geq 0$, profits $c_1, ..., c_n \geq 0$, and overall capacity W, find a subset $S \subseteq \{1, ..., n\}$ that maximizes the profit $\sum_{i \in S} c_i$ while maintaining $\sum_{i \in S} w_i \leq W$

10.2 Knapsack Algorithm

Dynamic Programming Knapsack Algorithm

Input: Profits $c_1, \ldots, c_n \in \mathbb{Z}_{\geq 0}$, weights $w_1, \ldots, w_n \in \mathbb{R}_{\geq 0}$, budget $W \in \mathbb{R}_{\geq 0}$
Output: Value of optimum Knapsack solution in time $O(n \cdot C)$

(1) Set $C := \sum_{j=1}^{n} c_j$; $T(0,0) := 0$, $T(0, C') := \infty \; \forall C' \in \{1, \ldots, C\}$
(2) FOR $j = 1$ TO n DO
$T(j, C') := \min\{T(j-1, C' - c_j) + w_j, T(j-1, C')\}$ for $C' \in \{0, \ldots, C\}$
(3) Return the max C^* so that $T(n, C^*) \leq W$

- $T(j, C')$ is minimum weight of items $S \subseteq \{1, ..., j\}$ that has profit C'
- The Knapsack algorithm outputs the maximum profit that maintains the weight constraint
- The subset of items that correspond to the optimal solution can be extracted by backtracking through the array entries that make up $T(n, C^*)$
- The Knapsack algorithm has $O(nC)$ running time, where C is the total cost of all items
  - This runtime holds if and only if the profits $c_1, ..., c_n$ are integral
  - If the weights are integral, then we can modify the algorithm to run in $O(nW)$ time, where W is the total weight of all items

10.3 Approximate Knapsack Algorithm

Approximation algorithm for Knapsack

Input: Profits $c_1, \ldots, c_n \in \mathbb{R}_{\geq 0}$, weights $w_1, \ldots, w_n \in \mathbb{R}_{\geq 0}$, budget $W \in \mathbb{R}_{\geq 0}$
Output: Solution S with profit $c(S) \geq (1 - \varepsilon)OPT$ where OPT denotes the value of the optimum solution.

(1) Scale all profits so that $\max_{i=1,\ldots,n}\{c_i\} = \frac{n}{\varepsilon}$
(2) Round $c'_i := \lfloor c_i \rfloor$, $C' := \sum_{i=1}^{n} c'_i$.
(3) Compute an optimum solution S for instance $((c'_i)_{i \in [n]}, (w_i)_{i \in [n]}, W)$ using the dynamic program.
(4) Return S

- The approximate Knapsack algorithm finds a feasible solution S with $(1 - \varepsilon)\mathrm{OPT} \leq c(S) \leq \mathrm{OPT}$
- The approximate Knapsack algorithm has $O(\frac{n^3}{\varepsilon})$ running time, where ε is the scaling factor