DISCRETE OPTIMIZATION

Doug Kang

Abstract. Streamlined notes from MATH 409: Discrete Optimization taught by Thomas Rothvoss at the University of Washington. Contains main definitions, statements, theorems for guided study.

Contents

1. Algorithms and Complexity

Discrete optimization deals with finding the best solution out of a finite number of possibilities in a computationally efficient way.

Definition. The running time of an algorithm is defined as the number of elementary operations (adding, subtracting, comparing, etc) that the algorithm makes.

Definition. If $f(s)$ and $g(s)$ are two positive real valued functions on $\mathbb{N}$, the set of non-negative integers, we say that $f(n) = O(g(n))$ if there is a constant $c > 0$ such that $f(n) \leq c \cdot g(n)$ for all n greater than some finite n_0.

Definition. A polynomial running time is of the form n^C, where $C > 0$ is a constant and n is the length of the input.

Definition. The complexity class **P** is the class of problems that admit a polynomial time algorithm.

Definition. The complexity class **NP** is the class of problems that admit a non-deterministic polynomial time algorithm.

Remark. Intuitively the complexity class P is the class of problems *solvable* in polynomial time, whereas the complexity class NP is the class of problems *verifiable* in polynomial time.

Definition. We say that a problem P $\in$ NP is **NP-complete** if with a polynomial time algorithm for P, one could solve any other problem in NP in polynomial time.

Remark. Since solving problems is harder than verifying problems, we believe that NP-complete problems cannot be solved in polynomial time (it is a millennium problem to prove that NP-complete problems do not have polynomial algorithms, i.e. prove NP $\neq$ P), and so we assume NP $\neq$ P.

2. Basic Graph Theory

2.1. Undirected Graphs.

Definition. An undirected graph $G = (V, E)$ is the pair of sets V and E where V is the set of vertices of G and E is the set of (undirected) edges of G.

An edge $e \in E$ is a set $\{i, j\}$ where $i, j \in V$. An edge $\{u, v\} \in E$ iff the vertices u and v are "connected" in the graph G. When $\{u, v\} \in E$, we say the vertices u and v are adjacent or that they are neighbors, and write $u \sim v$.

Definition. A subset $U \subseteq V$ induces a cut $\delta(U) = \{\{u, v\} \in E \mid u \in U, v \notin U\}$.

Definition. The degree of a node is the number of incident edges. We write $\deg(v) = |\delta(v)|$.

Definition. If the edges of G are precisely the $\binom{|V|}{2}$ pairs $\{i, j\}$ for every pair $i, j \in V$ (there is an edge between every two vertices) then G is the complete graph on V. The complete graph on $n = |V|$ vertices is denoted as K_n and the number of edges of G is $\binom{|V|}{2} = \frac{n(n-1)}{2}$.

Definition. A subgraph of $G = (V(G), E(G))$ is a graph $H = (V(H), E(H))$ where $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$ with the restriction that if $\{i, j\} \in E(H)$ then $i, j \in V(H)$.

Definition. If $V' \subseteq V(G)$, then the subgraph induced by V' is the graph $(V', E(V'))$ where $E(V')$ is the set of all edges in G for which both vertices are in V'.

Definition. A subgraph H of G is a spanning subgraph of G if $V(H) = V(G)$.

Definition. Given $G = (V, E)$ we can define subgraphs obtained by deletions of vertices or edges.

- If $E' \subseteq E$ then $G \setminus E' := (V, E \setminus E')$.
- If $V' \subseteq V$ then $G \setminus V' = (V \setminus V', E \setminus \{\{i,j\} \in E \mid i \in V' \text{ or } j \in V'\})$.

Definition. A path is a graph $P = (V, E)$ where $V = \{v_0, v_1, \dots, v_k\}$ and $E = \{\{v_0, v_1\}, \{v_1, v_2\}, \dots, \{v_{k-1}, v_k\}\}$ and all $v_0, \dots, v_k$ are distinct. The length of the path is the number of edges in the path which equals k. We will call this a (v_0, v_k)-path if we want to emphasize the endpoints.

Definition. A graph $G = (V, E)$ is connected if for all $u, v \in V$, G contains a (u, v)-path.

Definition. A walk in a graph $G = (V, E)$ is a sequence of vertices and edges $v_0, e_1, v_1, e_2, v_2, e_3, \dots, e_k, v_k$, such that for $i = 0, \dots, k$, $v_i \in V$, $e_i \in E$ where $e_i = \{v_{i-1}, v_i\}$. If $v_0 = v_k$ (startpoint = endpoint), then this is a closed walk.

Definition. A cycle is a graph $G = (V, E)$ where $V = \{v_0, v_1, \dots, v_{k-1}\}$ and $E = \{\{v_0, v_1\}, \{v_1, v_2\}, \dots, \{v_{k-1}, v_0\}\}$ where $v_0, \dots, v_{k-1}$ are distinct and $k \geq 3$. I.e. a cycle is a path from v_0 to v_0 (no repeated vertices).

Definition. A graph G is acyclic if it contains no cycle as subgraphs.

Definition. An acyclic graph is called a forest. A connected forest $T = (V(T), E(T))$ is a tree.

Definition. A subgraph $T = (V(T), E(T))$ of G is a spanning tree if T is spanning and a tree (connected and acyclic).

Definition. A Hamiltonian circuit of G is a subgraph that is a spanning cycle. I.e. visits every vertex once with no repeats.

Definition. A set $M \subseteq E$ of edges with degree ≤ 1 for each vertex is called matching.

Definition. A set $M \subseteq E$ of edges with degree exactly 1 for each vertex is called perfect matching.

2.2. **Directed Graphs.**

Definition. A directed graph $G = (V, E)$ is the pair of sets V and E where V is the set of vertices of G and E is the set of (directed) edges of G.

An edge $e \in E$ is a tuple (i, j) where $i, j \in V$ with $i \neq j$. One can imagine i as the "start node" and j as the "end node" of the edge.

In the literature, directed graphs are often called diagraphs and edges in a directed graph are called arcs.

Definition. A directed path is a directed graph $P = (V, E)$ with distinct vertices $V = \{v_0, v_1, \dots, v_k\}$ and edges $E = \{\{v_0, v_1\}, \{v_1, v_2\}, \dots, \{v_{k-1}, v_k\}\}$. We will consider the edge set E itself as the directed path.

3. The traveling salesperson problem

Traveling Salesperson Problem (TSP)
Input: We are given a complete graph $K_n = (V, E)$ with edge cost $c_{ij} \in \mathbb{R}_{\geq 0}\ \forall \{i, j\} \in E$.
Goal: Find the Hamiltonian circuit (V, C), $C \subseteq E$ that minimizes the total cost $\sum_{e \in C} c_e$.

A decent approach is the Held-Karp algorithm, which first computes the best subtours using dynamic programming. For each subset $S \subseteq V$ and points $i, j \in S$ we want to compute table entries

$$C(S, i, j) := \text{length of the shortest path from } i \text{ to } j \text{ visiting each node in } S \text{ exactly once.}$$

Now we simply compute all entries $C(S, i, j)$ with increasing size S: The number of table entries

Algorithm 1: Held-Karp algorithm

```
Input: Distances $c_{ij} \geq 0$ in a complete $n$-node graph.
Output: Cost of an optimum TSP tour.
1 $C(\{i, j\}, i, j) := c_{ij}$ for all $i \neq j$
2 for $k = 3, \dots, n$ do
3     for all sets $S \subseteq V : |S| = k$ do
4         $C(S, i, j) := \min_{\ell \in S \setminus \{i,j\}} \{C(S \setminus \{j\}, i, \ell) + c_{\ell,j}\}$
5     end for
6 end for
7 output the optimum cost $\min_{j \neq 1}\{C(V, 1, j) + c_{j1}\}$
```

$C(S, i, j)$ is upper bounded by $2^n \cdot n^2$ since there are 2^n many subsets S and at most n^2 pairs i, j. Computing a single table entry takes linear time. Evaluating each term $C(S \setminus \{j\}, i, \ell) + c_{\ell,j}$ takes $O(1)$ many elementary operations. Therefore, the overall running time of the algorithm is $O(n^3 2^n)$.

4. Minimum spanning trees

Minimum Spanning Tree
Input: Undirected graph $G = (V, E)$ with edge cost $c_{ij} \geq 0\ \forall \{i, j\} \in E$.
Goal: A spanning tree (V, T) with $T \subseteq E$ minimizing $c(T) := \sum_{e \in T} c_e$.

Recall that in a graph $G = (V, E)$ a subgraph (V', T) is a spanning tree if it is

(i) acyclic, (ii) connected (iii) spanning ($V' = V$).

Slight abuse of notation: We call a subset of edges $T \subseteq E$ a spanning tree of G if this is the case for the subgraph (V, T). Equivalent definition:

Definition. The edges $T \subseteq E$ form a spanning tree of G if and only if for each pair $u, v \in V$ of nodes there is exactly one path from u to v.

Here are some basic properties of a spanning tree.

Lemma. For $G = (V, E)$ connected with $|V| = n$ nodes. Then one has:

(a) $|E| \geq n - 1$.
(b) $\exists$ spanning tree $T \subseteq E$.
(c) G is acyclic $\iff$ $|E| = n - 1$.
(d) G is spanning tree $\iff$ $|E| = n - 1$.

Proof. Fix root $r \in V$ and form the spanning tree $T := \{\{v, p(v)\} \mid v \in V \setminus \{r\}\}$ of all edges where $p(v)$ is exactly one arbitrary neighbor node of v that is closer to r, meaning $d(p(v)) = d(v) - 1$. ☐

Observations (can you justify them all?):

(i) (V, T) is connected.
(ii) $|E| \geq |T| = n - 1$.
(iii) (V, T) is acyclic.
(iv) T is a spanning tree in G.
(v) For any edge $\{u, v\} \in E \setminus T$, we have $T \cup \{u, v\}$ contains a cycle.
(vi) d) follows from c).

Lemma. Let $T \subseteq E$ be a spanning tree in $G = (V, E)$. Take an edge $e' = \{u, v\} \in E \setminus T$ and let $P \subseteq T$ be the edges on the path from u to v. Then for any edge $e \in P$, $T' = (T \setminus \{e\}) \cup \{e'\}$ is a spanning tree.

Proof. Swapping edges in a cycle $P \cup \{e'\}$ leaves T connected and the number of edges is still $n-1$, hence T' is still a spanning tree. ☐

Definition. The operation to replace $e \in P$ from u to v by $e' = \{u, v\} \in E \setminus T$ is called an edge swap.

Definition. For any graph $G = (V, E)$ we can partition the nodes into $V = V_1 \cup \cdots \cup V_k$ so that a pair u, v is in the same set V_i if and only if they are connected by a path. Those sets $V_1, \ldots, V_k$ are called connected components.

Theorem. *Let T be a spanning tree in graph G with edge costs c_e for all $e \in E$. TFAE:*

(1) T is a minimum spanning tree (MST).
(2) No edge swap for T decreases the cost.

4.1. **Kruskal's algorithm.** Algorithm to find MST which considers each edge only once and makes an immediate decision whether or not to include that edge.

Algorithm 2: 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 $c_{e_1} \leq c_{e_2} \leq \cdots \leq c_{e_m}$.
2 Set $T = \varnothing$.
3 for $i = 1, \ldots, n$ do
4     If $T \cup \{e_i\}$ is acyclic then update $T := T \cup \{e_i\}$.
5 end for
```

Theorem. *The edge set T found by Kruskal's algorithm is an MST of G.*

Theorem. *Kruskal's algorithm has a running time of $O(m \log(n))$ (in the arithmetic model) where $n = |V|$ and $m = |E|$.*

5. Shortest Paths

Shortest Path Problem
Input: Directed graph $G = (V, E)$, cost c_{ij} $\forall (i,j) \in E$, nodes $s, t \in V$.
Goal: s-t path $P \subseteq E$ minimizing $c(P) := \sum_{e \in P} c_e$.

Lemma. (Bellman's principle). Let $G = (V, E)$ be a directed, weighted graph with no negative cost cycle. Let $P = \{(s, v_1), (v_1, v_2), \ldots, (v_k, t)\}$ be the shortest s-t path. Then $P_{s,i} = \{(s, v_1), (v_1, v_2), \ldots, (v_{i-1}, v_i)\}$ is a shortest s-v_i path.

Proof. If we have a s-v_i path Q that is shorter, then replacing the s-v_i part in P with Q would give a shorter s-t path. □

Note: "Shortest path" means the one minimizing the cost (not necessarily the one minimizing the number of edges).

5.1. **Dijkstra's algorithm.** Dijkstra's algorithm is the most efficient algorithm to compute shortest paths from a given vertex s to all vertices v in a diagraph G without negative cost cycles.

```
Algorithm 3: Dijkstra's Algorithm
  Input: A directed graph $G = (V, E)$ with edge costs $c_e \geq 0$ $\forall e \in E$. A source vertex $s$.
  Output: Length $\ell(v)$ of shortest $s$-$v$-paths for all $v \in V$.
1 Set $\ell(s) = 0$. $R = \{s\}$ and for all $v \in V \setminus \{s\} : \ell(v) = \begin{cases} c(s,v) & \text{if } (s,v) \in E \\ \infty & \text{otherwise.} \end{cases}$
2 while $R \neq V$ do
3     Select $v \in V \setminus R$ attaining $\min\{\ell(v) \mid v \in V \setminus R\}$.
4     for all $w \in V \setminus R$ with $(v, w) \in E$ do
5         $\ell(w) := \min\{\ell(w), \ell(v) + c(v, w)\}$.
6     end for
7     Set $R = R \cup \{v\}$.
8 end while
```

For proofs see page 24-26.

Lemma. Dijkstra's algorithm runs in $O(n^2)$ time in an n-node graph.

Proof. The algorithm has n iterations and the main work is done for updating the $\ell(w)$-values. Note that one update step $\ell(w) := \min\{\ell(w), \ell(v) + c(v, w)\}$ takes time $O(1)$ and we perform n such updates per iteration. In total the running time is $O(n^2)$. □

5.2. **Moore-Bellman-Ford Algorithm.** We now study a second algorithm for finding shortest paths in a directed graph that (unlike Dijkstra's algorithm) can deal with negative edge costs at the expense of a higher running time.

Algorithm 4: Moore-Bellman-Ford Algorithm

```
  Input: A directed graph $G = (V, E)$; edge cost $c(e) \in \mathbb{R}\ \forall e \in E$; a source node $s \in V$.
1 Assumption: There are no negative cost cycles in $G$.
  Output: The length $\ell(v)$ of the shortest $s - v$ path, for all $v \in V$.
2 Set $\ell(s) := 0$ and $\ell(v) = \infty$ for all $v \in V \setminus \{s\}$.
3 for $i = 1, \ldots, n-1$ do
4     for each $(v, w) \in E$ do
5         If $\ell(w) > \ell(v) + c_{vw}$ then set $\ell(w) = \ell(v) + c_{vw}$ ("update edge $(v, w)$")
6     end for
7 end for
```

Theorem. *The Moore-Bellman-Ford Algorithm runs in $O(mn)$-time.*

Proof. The outer loop takes $O(n)$ time since there are $n - 1$ iterations. The inner loop takes $O(m)$ time as it checks the l-value at the head vertex of every edge in the graph. This gives an $O(nm)$-algorithm. □

5.3. **Detecting negative cycles.** Algorithms like Dijkstra's and Moore-Bellman-Ford to compute the shortest (s, v)-paths in a directed graph G require no negative cost cycles, in which case we say that the vector of costs, c, is conservative.

Definition. Let G be a diagraph with costs $c_e \in \mathbb{R}$ for all $e \in E$.
Let $\pi : V \to \mathbb{R}$ be a function that assigns a real number $\pi(v)$ to every vertex $v \in V$.

(1) For an edge $(i, j) \in E$, define the reduced cost with respect to π to be $c_{ij} + \pi(i) - \pi(j)$.
(2) If the reduced costs of all edges of G are nonnegative, then π is a feasible potential on G.

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

Lemma. Given (G, c), in $O(nm)$ time we can find either a feasible potential on G or a negative cost cycle in G.

6. Network flows

Definition. A network (G, u, s, t) consists of the following data:

- A directed graph $G = (V, E)$,
- edge capacities $u_e \in \mathbb{Z}_{\geq 0}$ for all $e \in E$, and
- two specified nodes s (source) and t (sink) in V.

Definition. A s-t flow is a function $f : E \to \mathbb{R}_{\geq 0}$ with the following properties:

- The flow respects the capacities, i.e. $0 \leq f(e) \leq u_e \; \forall e \in E$.
- It satisfies flow conservation at each vertex $v \in V \setminus \{s, t\}$:

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

where $\delta^-(v) = \{(w, v) \in E\} = \{\text{edges entering } v\}$ and $\delta^+(v) = \{(v, w) \in E\} = \{\text{edges leaving } v\}$.

The value of an s-t flow f is

$$\text{value}(f) := \sum_{e \in \delta^+(s)} f(e) - \sum_{e \in \delta^-(s)} f(e)$$

which is the total flow leaving the source node s minus the total flow entering s.

Maximum Flow Problem
Input: A network (G, u, s, t).
Find: An s-t flow in the network of maximum value.

6.1. **Ford-Fulkerson algorithm.**

Definition. Given a network $(G = (V, E), u, s, t)$ with capacities u and an s-t flow f. The residual graph $G_f = (V, E_f)$ has edges

$$E_f = \underbrace{\{(i,j) \mid (i,j) \in E \text{ and } u(i,j) - f(i,j) > 0\}}_{\text{forward edges / residual capacity}} \cup \underbrace{\{(j,i) \mid (i,j) \in E \text{ and } f(i,j) > 0\}}_{\text{backward edges / current flow}}.$$

An edge $(i, j) \in E_f$ has residual capacity

$$u_f(i,j) = \begin{cases} u(i,j) - f(i,j) & \text{if } (i,j) \text{ is a forward edge.} \\ f(j,i) & \text{if } (i,j) \text{ is a backward edge.} \end{cases}$$

An s-t path in G_f is called an f augmenting path. Given a flow f and an s-t path P in G_f, to augment f along P by γ means to do the following for each $e \in P$: Increase $f(e)$ by γ on all the forward edges *of the path* and decrease $f(e)$ by γ on all the backward edges of P.

Definition. For a flow f we define the excess of a node as

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

Lemma. Let (G, u, s, t) be a network with an s-t flow f. Let P be an f-augmenting path so that $u_f(e) \geq \gamma \; \forall e \in P$ for some $\gamma \geq 0$. Let us augment f along P by γ and call the result f'. Then f' is a feasible s-t flow of value value(f') =value$(f) + \gamma$.

Algorithm 5: 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 while There exists an $f$-augmenting path $P$ do
3     Compute $\gamma := \min\{u_f(e) \mid e \in P\}$.
4     Augment $f$ along $P$ by $\gamma$.
5 end while
```

6.2. **Min Cut problem.**

s-*t* Min Cut problem
Input: A network (G, u, s, t).
Find: A set $S \subseteq V$ with $s \in S$, $t \notin S$ (i.e. an s-t cut) minimizing

$$\underbrace{u(\gamma^+(S)) = \sum_{(i,j)\in E: i\in S, j\notin S} u(i,j)}_{\text{The capacity of edges leaving } S}.$$

Lemma. Let (G, u, s, t) be a network with an s-t flow f and an s-t cut $S \subseteq V$. Then value$(f) \leq u(\delta^+(S))$. In other words, the s-t min cut gives an upper bound on the max flow.

Corollary. In the last proof we have value$(f) = u(\delta^+(S))$ if and only if both conditions are satisfied:

- All outgoing edges from S are saturated: $f(e) = u(e)\ \forall e \in \delta^+(S)$.
- All incoming edges into S have 0 flow: $f(e) = 0\ \forall e \in \delta^-(S)$.

Now we can prove correctness of Ford and Fulkerson's algorithm.

Theorem. *An (s-t)-flow f is optimal if and only if there does not exist an f-augmenting path in G_f.*

Proof. Pg 37. □

Theorem. *(MaxFlow – MinCut). The max value of an s-t flow in the network (G, u, s, t) equals the min capacity of a s-t cut in the network.*

Remark. The Ford-Fulkerson alg. not only find a max flow but also the min cut in the network.

Lemma. For the network (G, u, s, t), let $U := \max\{u(e) \mid e \in E\}$ be the maximum capacity. Then the Ford-Fulkerson algorithm runs in time $O(n \cdot m \cdot U)$.

6.3. **The Edmonds-Karp algorithm.** We modify the Ford-Fulkerson algorithm by selecting the shortest augmenting path in the residual graph G_f.

Algorithm 6: The Edmonds-Karp algorithm

```
  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 while There exists an $f$-augmenting path $P$ do
3     Find a shortest $f$-augmenting path $P$.
4     Compute $\gamma := \min\{u_f(e) \mid e \in P\}$.
5     Augment $f$ along $P$ by $\gamma$.
6 end while
```

Theorem. *The Edmonds-Karp algorithm requires at most $m \cdot n$ iterations and has runnint time $O(m^2 n)$.*

Proof. Pg 38-39. □

Corollary. In a network (G, u, s, t) one can find an optimum s-t MinCut in time $O(m^2 n)$.

Proof. Run the Edmonds-Karp algorithm to compute a maximum flow f. As in the MaxCut – MinCut Theorem, we know that the set S of nodes that are reachable from s in the residual graph G_f will be a MinCut!. □

Corollary. In a network (G, u, s, t) with integral capacities (i.e.$u(e) \in Z_{\geq 0}$ $\forall e \in E$) there is always a maximum flow f that is integral.

Proof. If the capacities $u(e)$ are itnegral, then the increment γ that is used in either Ford-Fulkerson or in Edmonds-Karp is integral as well. Then also the resulting flow is integral. □

6.4. **Applications to bipartite matching.**

Definition. A bipartite graph is a graph in which the vertices can be partitioned into two disjoint sets V_1 and V_2 such that each edge is an edge between a vertex in V_1 and a vertex in V_2.

Definition. Recall that matching $M \subseteq E$ in an undirected graph $G = (V, E)$ is a set of edges so that each node is incident to at most one of those edges. A Maximum matching is a matching that maximizes the number $|M|$ of edges.

Let $G = (V_1 \cup V_2, E)$ be a bipartite graph. Now make all the edges directed from V_1 to V_2 and and add new nodes s, t. Set capacities u as 1 if its an edge from/to s/t and ∞ otherwise.

Lemma. Let $G = (V_1 \cup V_2, E)$ be a bipartite graph and (G', u, s, t) be the network constructed as above. Then

$$\max\{|M| \mid M \subseteq E \text{ matching}\} = \max\{\text{value}(f) \mid f \text{ is } s\text{-}t \text{ flow in } G'\}$$

Proof. pg. 40 □

Lemma. Maximum matching in bipartite graphs can be solved in $O(n \cdot m)$ time ($|V| = n, |E| = m$).

6.5. **Konig's Theorem.**

Theorem. *(Konig). For a bipartite graph $G = (V = V_1 \cup V_2, E)$ one has*

$$\max\{|M| \mid M \subseteq E \textit{ matching}\} = \min\{|U| \mid U \subseteq V_1 \cup V_2 \textit{ is vertex cover.}\}$$

Proof. pg. 41-42 □

6.6. **Hall's Theorem.**

Definition. For a bipartite graph $G = (V_1 \cup V_2, E)$ and nodes $U \subseteq V_1$, let $N(U) = \{v \in V_2 \mid \exists(u, v) \in E\}$ be the neighborhood of U.

Theorem. *(Hall's Theorem). A bipartite graph $G = (V_1 \cup V_2, E)$ with $|V_1| = |V_2| = n$ has a perfect matching if and only if $|N(U)| \geq |U|$ for all $U \subseteq V_1$.*

7. Linear programming

Definition. A vector $c \in \mathbb{R}^n$ can be used to define a linear function $f : \mathbb{R}^n \to \mathbb{R}$ by setting

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

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

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

for some matrix $A \in \mathbb{R}^{m \times n}$ and a vector $b \in \mathbb{R}^m$.

Definition. A polyhedron is called a polytope if it is bounded, i.e., can be enclosed in a ball of finite radius.

Definition. A set $Q \subseteq \mathbb{R}^n$ is convex if for any two points x and y in Q, the line segment joining them is also in Q. Mathematically, for every pair of points $x, y \in Q$, the convex combination $\lambda x + (1 - \lambda) y \in Q$ for every λ such that $0 \leq \lambda \leq 1$. E.g. Polyhedra are convex.

Definition. A convex combination of a finite set of points $v_1, \ldots, v_t \in \mathbb{R}^n$, is any vector of the form $\sum_{i=1}^t \lambda_i v_i$ such that $0 \leq \lambda_i \leq 1$ for all $i = 1, \ldots, t$ and $\sum_{i=1}^t \lambda_i = 1$. The set of all convex combinations of $v_1, \ldots, v_t$ is called the convex hull of $v_1, \ldots, v_t$. We denote it by

$$\mathrm{conv}\{v_1, \ldots, v_t\} = \left\{ \sum_{i=1}^t \lambda_i v_i \mid \lambda_1 + \cdots + \lambda_t = 1; \lambda_i \geq 0 \; \forall i = 1, \ldots, t \right\}.$$

Theorem. *Every polytope P is the convex hull of a finite number of points (and vice versa).*

Definition. For a convex set $P \subseteq \mathbb{R}^n$ (such as polytopes of polyhedra) we call a point $x \in P$ an extreme point / vertex of P if there is no vector $y \in \mathbb{R}^n \setminus \{0\}$ with both $x + y \in P$ and $x - y \in P$.

Definition. 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 $x = (x_1, \ldots, x_n)$ in a polyhedron P. Mathematically, it is the problem

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

for some matrix A and vector b (alternatively min instead of amx).

Lemma. Suppose $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ is bounded with $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$ and let $c \in \mathbb{R}^n$. Then LP $\max\{c^T x \mid x \in P\}$ is maximized at an extreme point x^* and moreover, x^* is the unique solution to the system of linear equations $A_i^T x = b_i \; \forall i \in I$ where $I \subseteq [m]$ with $|I| = n$.

Remark. So while LPs are continuous optimization problems, the lemma tells us that optimum solutions are always taken from a finite set of extreme points.

7.1. **Duality.**

Remark. Consider the following pair of LPs:

$$\textbf{primal } (P) : \max\{c^T x \mid Ax \leq b\}$$
$$\textbf{dual } (D) : \min\{b^T y \mid y^T A = c^T, y \geq \mathbf{0}\}$$

The dual LP is searching for the best upper bound for the primal LP.

Theorem. *(Weak duality Theorem). One has $(P) \leq (D)$. More precisely if we have (x, y) with $Ax \leq b, y^T A = c^T$ and $y \geq \mathbf{0}$, then $c^T x \leq b^T y$.*

Proof. One has

$$\underbrace{c^T}_{=y^T A} x = \underbrace{y^T}_{\geq \mathbf{0}} \underbrace{Ax}_{\leq b} \leq y^T b.$$

□

Next we want to show that always $(P) = (D)$, which means we can always combine a feasible inequality that will give a tight upper bound.

Theorem. *(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 a hyperplane $c^T x = \beta$ with*

$$c^T x < \beta < c^T y \ \ \forall x \in P \, \forall y \in Q.$$

Proof. Let $(x^*, y^*) \in P \times Q$ be the pair minimizing the distance $\|x^* - y^*\|_2$. Then the hyperplane through $\frac{1}{2}(x^* + y^*)$ with normal vector $c = y^* - x^*$ separates P and Q. □

Lemma. (Farkas Lemma). One has $(\exists x \geq \mathbf{0} \mid Ax = b) \vee (\exists y \mid y^T A \geq 0 \text{ and } y^T b < 0)$.

Proof. pg. 49 □

Theorem. *(Strong duality Theorem). One has $(P) = (D)$. More precisely, one has*

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

given that both systems have a solution.

Note moreover, if (P) is unbounded,then (D) is infeasible. If (D) is unbounded then (P) is infeasible. On the other hand, it is possible that (P) and (D) are both infeasible.

Theorem. *(Complementary slackness). Let (x^*, y^*) be feasible solutions for*

$$(P) : \max\{c^T x \mid Ax \leq b\} \text{ and } (D) : \min\{b^T y \mid y^T A = c^T ; y \geq \mathbf{0}\}.$$

Then (x^*, y^*) are both optimal if and only if

$$(A_i^T x^* = b_i \vee y_i^* = 0) \ \ \forall i.$$

7.2. **Algorithms for linear programming.** Simplex

Algorithm 7: 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: optimal 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
3     return $x$ is optimal
4 end if
5 select $j \in I$ and $j' \notin I$ so that for $I' := I \setminus \{j\} \cup \{j'\}$ the following 3 conditions are satisfied
      (1) rank$(A_{I'}) = n$.
      (2) the point $x' = A_{I'}^{-1} b_{I'}$ lies in $P$.
      (3) $c^T x' \geq c^T x$.
  UPDATE $I := I'$ and GOTO (1).
```

Let $I \subseteq [m]$ be a set of row indices and let A_I be the submatrix of A that contains all the rows with index in I. At each iteration the maintained set I of indices is usually called a **basis** and the maintained solution x is always a **vertex** of P. Simplex method moves from vertex to vertex so objective always improves (or at least does not get worse). Not a polynomial time algorithm because of worst case!

Simplex: practical: very fast, worst case: exponential, works for LPs.
Interior point: practical: fast, worst case: $O(n^{3.5}L)$ arithmetic operations, works for LPs + SDPs.
Ellipsoid method: practical: slow, worst case: $O(n^6 L)$, works for any convex set. L is the number of bits necessary to represent the input.

7.3. **Connection to discrete optimization.** How can we use efficient LP algorithms to solve discrete optimization problems? Case study: Perfect Matching Problem.

Max Weight Perfect Matching
Input: Undirected graph $G = (V, E)$, weights $c_{ij} \geq 0$.
Goal: A perfect matching $M \subseteq E$ maximizing $\sum_{e \in M} c_e$.

Recall that a perfect matching $M \subseteq E$ is a set of edges that has degree exactly 1 for each node in the graph.

Lemma. Let $G = (V, E)$ be a bipartite graph and x be an optimum extreme point solution to the matching LP. Then x is integral and $M = \{e \in E \mid x_e = 1\}$ is an optimum perfect matching.

Proof. Skipped proof pg. 58 □

7.4. **Integer programs and integer hull.**

Remark. In general, for any polyhedron $P \subseteq \mathbb{R}^n$, the LP $\max\{c^T x \mid x \in P\}$ is associated with the integer program $\max\{c^T x \mid x \in P \cap \mathbb{Z}^n\}$. The LP is also called an LP relaxation of its integer program. Not difficult to construct an example where the LP is feasible while the integer program is infeasible. Note that always

$$\max\{c^T x \mid x \in P\} \geq \max\{c^T x \mid x \in P \cap \mathbb{Z}^n\}$$

(given that both systems have a finite value) since the left hand maximum is over a larger set. Also note that the same integer program could have several different LP relaxations.

Definition. For a polyhedron $P \subseteq \mathbb{R}^n$, we call

$$P_I = \operatorname{conv}(P \cap \mathbb{Z}^n)$$

the integer hull. The integer hull P_I is the smallest polyhedron that contains all the integral points in P. That means P_I is the perfect relaxation for the integer program. Optimizing any function over it gives the same value as the integer program.

Lemma. Let $P \subseteq \mathbb{R}^n$ be a polytope. TFAE:

- $P = P_I$ (i.e. the LP is the same as the integer program).
- For each $c \in \mathbb{R}^n$, $\max\{c^T x \mid x \in P\} = \max\{c^T x \mid x \in P \cap \mathbb{Z}^n\}$.

Ideally we want to solve integer programs since every discrete optimization problem can be modelled as an integer program. But integer programming is NP-hard (assuming NP $\neq$ P, no polynomial time algorithm for such programs in general).

8. Total unimodularity

Motivation is to investigate integer programs we can solve in polynomial time by computing an extreme point solution to the linear relaxation. Recall the following lemma from last chapter:

Lemma. 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 $I \subseteq \{1, \dots, m\}$ so that A_I is an $n \times n$ matrix of full rank and x is the unique solution to $A_I x = b_I$.

Here A_I denotes the rows of A that have their index in I. When can we guarantee that the solution to $A_I x = b_I$ is integral? Recall the following lemma from linear algebra:

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

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

Here B^i is the matrix B where the ith column is replaced by b.

Definition. A matrix A is totally unimodular (TU) if every square submatrix of A has determinant $0, 1$ or -1.

Note that A itself does not have to be square. If A is totally unimodular, then every entry of A has to be $0, \pm 1$ since every entry is a 1×1 submatrix of A.

Lemma. Let $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$. If A is TU and $b \in \mathbb{Z}^m$, then all extreme points of P are integral.

Proof. pg. 64 □

Lemma. Let $A \in \{-1, 0, 1\}^{m\times n}$. Then the following operations do not change whether or not A is TU:

(a) Multiplying a row/column with -1
(b) Permuting rows/columns
(c) Adding/removing a row/column that has at most ± 1 entry and zeros otherwise
(d) Duplicating rows
(e) Transposing the matrix.

Proof. pg. 64 □

Corollary 1. Let $A \in \{-1, 0, 1\}^{m\times n}$ be a TU matrix with $b \in \mathbb{Z}^m$ and $\ell, u, \in \mathbb{Z}^n$. Then

(a) All extreme points of $\{x \in \mathbb{R}^n \mid Ax \leq b; x \geq \mathbf{0}\}$ are integral.
(b) All extreme points of $\{x \in \mathbb{R}^n \mid Ax = b; x \geq \mathbf{0}\}$ are integral.
(c) All extreme points of $\{x \in \mathbb{R}^n \mid Ax \leq b; \ell \leq x \leq u\}$ are integral.

Proof. From the prev. lemma we know that also the matrices

$$\begin{pmatrix} A \\ -I \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} A \\ -A \\ -I \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} A \\ -I \\ I \end{pmatrix}$$

that define the 3 systems are TU. Then all the claims follow. □

Remark. Remember that not all polyhedra have extreme points and the previous Lemmas are useless in that case. But remember that all polyhedra that have non-negativity constraints must have extreme points.

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

$$A =?$$

so that the following conditions hold:

- B has at most one non-zero entry per column
- C_1 has exactly one $+1$ entry per column
- C_2 has exactly one -1 entry per column.

Proof. See pg. 65. □

Remember that going to the dual LP means transposing the constraint matrix. Since transposing a matrix does not destroy TU we immediately get:

Lemma. Suppose A is TU and both b and c are integral. Then there are optimum integral solutions x^*, y^* to

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

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

And a surprising fact:

Theorem. *(Seymour '80, Truemper '82). For a matrix $A \in \{-1, 0, 1\}^{m \times n}$ one can test in polynomial time whether A is TU.*

8.1. **Application to bipartite matching.**

Let $G = (V, E)$ be an undirected graph and consider the matching polytope

$$P_M = \left\{ x \in \mathbb{R}^E \;\middle|\; \sum_{e \in \delta(v)} x_e = 1 \ \forall v \in V; x_e \geq 0 \ \forall e \in E \right\}.$$

As we saw before, for bipartite graphs all extreme points of P_M are integral. We want to understand why.

Lemma. If G is bipartite, then all extreme points x of P_M satsify $x \in \{0, 1\}^E$.

Lemma. The incidence matrix of an undirected graph G is TU if and only if G is bipartite.

8.2. **Application to flows.**

Lemma. The node edge incidence matrix $A \in \{-1, 0, 1\}^{V \times E}$ of a directed graph is TU.

Corollary. For any integer k, the polytope P_{k-flow} of value k s-t flows has only integral extreme points.

Lemma. Minimum Cost Integer Circulation Problem can be solved in polynomial time.

8.3. **Application to interval scheduling.**

Lemma. Minimum Cost Integer Circulation Problem can be solved in polynomial time.

Interval Scheduling
Input: A list of intervals $I_1, \ldots, I_n \subseteq \mathbb{R}$ with profits c_i for interval I_i. **Goal:** Select a disjoint subset of intervals that maximize the sum of profits.

Observe that the matrix $A \in \{0,1\}^{m \times n}$ has the consecutive-ones property, that means the ones in each column are consecutive.

Lemma. A matrix $A \in \{0,1\}^{m \times n}$ with consecutive ones property is TU.

Theorem. *The interval scheduling problem can be solved in polynomial time.*

9. Branch & Bound

Algorithm 8: 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 $\max\{c^T x \mid Ax \leq b; x \in \mathbb{Z}^n\}$ and denote LP solution by $\tilde{x}$.
6      if $P' = \varnothing$ then
7          goto (3)
8      end if
9      if $c^T x^* \geq c^T \tilde{x}$ then
10         goto (3)
11     end if
12     if $c^T x^* < c^T \tilde{x}$ then
13         update $x^* := \tilde{x}$ and goto (3)
14     end if
15     else
16         (i.e. $\tilde{x} \notin \mathbb{Z}^n, c^T x^* < c^T \tilde{x}$)
17         Select coordinate $i$ with $\tilde{x}_i \notin \mathbb{Z}$.
18         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.
19     end if
20 end while
```

- On which variable should one branch in (line 17)?
  Branch on the most fractional variable. For example, in $\tilde{x} = (0.99, 0.5)$ one would branch on x_2 instead of x_1.
- Which problem P' should be selected from the stack? There are two popular strategies:
  Depth First Search: Always select the last added problem from the stack, corresponds to DFS.
  Best bound: Select the problem that has the highest LP maximum. The hope is that there is also a good integral solution hiding.

10. Non-bipartite matching

Apart from **NP**-complete problems, the only problem we have mentioned before for which we have not seen a polynomial time algorithm yet is the matching problem on general, non-bipartite graphs.

Recall that in an undirected graph $G = (V, E)$, a matching is a subset of disjoint edges $M \subseteq E$.

Maximum Cardinality Matching
Input: An undirected graph $G = (V, E)$.
Goal: Find a matching M of maximum size $|M|$.

10.1. Augmenting paths.

Definition. Assume we have a matching M.

- A node that is not contained in the matching is called **M-exposed**.
- A walk P in the graph is **M-alternating** if its edges are alternately in M and not in M.
- A walk P in the graph is **M-augmenting** if it is an **M-alternating** walk that starts and ends in different, **M-exposed** vertices and that does not revisit any nodes or edges (P is a path and not just a walk).

Theorem. *(Augmenting Path Theorem for Matchings). A matching M in a graph $G = (V, E)$ is maximum if and only if there is no M-augmenting path.*

Proof. Pg. 78. □

10.2. Computing augmenting paths.

We will demonstrate how to find an augmenting path given $G = (V, E)$.

(1) Create a bipartite directed graph G' which for all $v \in V$ has a node v in the left part and a mirror node v' in the right part.
(2) For each non-edge $\{u, v\} \in E \setminus M$ we create two directed edges (u, v') and (v, u').
(3) For each matching edge $\{u, v\} \in M$ we create backward edges (u', v) and (v', u).

Observe that a path in this directed bipartite graph corresponds to an M-alternating walk.

Lemma. In time $O(mn + n^2 \log n)$ we can compute the shortest M-alternating walk starting and ending in an exposed node.

Proof. We run Dijkstra's algorithm in G' for each node as source to compute shortest path distances. □

Definition. Assume we have a matching M.

- We define an **M-flower** as an M-alternating walk in $G = (V, E)$ that (i) starts at an M-exposed node u (ii) revisits exactly one node v once which is the other end point of the path and (iii) the cycle that is closed at v has an odd number of edges.
- That odd cycle $B \subseteq E$ that contains exactly $|M \cap B| = \frac{1}{2}(|B| - 1)$ edges is called an **M-blossom**. Note that it is possible that u and v are identical.

Lemma. In time $O(mn + n^2 \log n)$ we can compute (i) either an M-augmenting path P or (ii) find an M-flower or (iii) decide that there is no M-augmenting path.

It remains to discuss what to do if we find an M-flower.

10.3. **Contracting odd cycles.**

We contract the blossom. Let B be the edges of the blossom, then we write $G \times B$ as the graph that we obtain by contracting all the nodes on the odd cycle into a single super-node v_B. Why contract an odd cycle? Because the contracted graph is smaller and it suffices to find an augmenting path in the contracted graph.