COMP3530: Discrete Optimisation

Charles Christopher Hyland

Semester 2 2019

Abstract

Thank you for stopping by to read this. These are notes collated from lectures and tutorials as I took this course.

Contents

COMP3530: Discrete Optimisation

0. Graph Theory

0.1 Graph Theory

0.1.1 Graph Theory

Definition 0.1 *(Walk). A walk is a sequence of alternating vertices and edges $v_0, e_1, v_1, e_2, ..., e_k, v_k$.*

Definition 0.2 *(Closed walk). A closed walk is a walk where the starting vertex is the same as the ending vertex, that is, $v_0 = v_k$.*

Definition 0.3 *(Trail). A trail is a walk with no repeated edges.*

Definition 0.4 *(Path). A path is an open trail with no repeated vertices.*

Lemma 0.5 *All paths are open walks.*

Definition 0.6 *(Cycle). A cycle is a closed trail where there are no other repeated vertices except for the start and ending vertex.*

Definition 0.7 *(Circuit). A circuit a closed trail. That is, a circuit has no repeated edges but may have repeated vertices.*

Definition 0.8 *(Hamiltonian cycle). A graph contains a Hamiltonian cycle if the graph has a cycle that passes through each node exactly once.*

Definition 0.9 *(Vertex cutset). For a connected graph $G = (V, E)$, a vertex cutset if a subset $W \subseteq V$ the removal of W disconnects the graph G.*

Definition 0.10 *(Edge cutset). For a connected graph $G = (V, E)$, an edge cutset M is a subset $M \subseteq M$ where the removal of M disconnects the graph G.*

COMP3530: Discrete Optimisation

1. Introduction to Linear Programming and Polyhedra Theory

1.2 Polyhedra Theory

1.2.1 Introduction to Linear Programming

We are interested in minimizing a linear cost function subject to linear equality and inequality constraints.

Definition 1: General Linear Programming Problem

A general programming problem is given a cost vector $\mathbf{c} = (c_1, ..., c_n)$, we seek to minimise a linear cost function $\mathbf{cx} = \sum_{i=1}^{n} c_i x_i$ over $\mathbf{x} \in \mathbb{R}^n$ subject to linear constraints, which can be of any form. The variables $x_1, ..., x_n$ are called decision variables.

Let M_1, M_2, M_3 be finite index sets and for every i in any one of these sets, we are given a vector $a_i \in \mathbb{R}^n$ and a scalar b_i to model the i-th constraint. Let N_1, N_2, N_3 be subsets of $\{1, ..., n\}$ to indicate which decision variables x_j are constrainted to be nonnegative, nonpositive, or free.

Definition 2: Formulation of Linear Program

Let $x \in \mathbb{R}^n$ be our decision variables and $c \in \mathbb{R}^n$ our cost vector. The linear program is of the form

$$\min_{\tilde{x}} c^T x$$

$$\text{subject to} \begin{cases} a_i^T x \geq b_i & i \in M_1 \\ a_i^T x \leq b_i & i \in M_2 \\ a_i^T x = b_i & i \in M_3 \\ & \\ x_j \geq 0 & j \in N_1 \\ x_j \leq 0 & j \in N_2 \\ x_j \text{ free} & j \in N_3 \end{cases}$$

Definition 3: Feasible Solution

The vector $\mathbf{x}$ satisfying all the linear constraints is known as a feasible solution. The set of all feasible solutions is known as the **feasible region**.

Definition 1.11 *(Optimal Solution). A feasible solution* $\boldsymbol{x}^*$ *that satisfies the property* $\boldsymbol{cx}^* \leq \boldsymbol{cx}$ *for all feasible solutions* x *is known as the* ***optimal feasible solution****. The value* $\boldsymbol{cx}^*$ *is known as the optimal cost.*

Definition 1.12 *(Unbounded problem). We say that a linear program is an unbounded problem if for every*

$K \in \mathbb{R}$, there exists a feasible solution $\boldsymbol{x}$ such that

$$\boldsymbol{cx} < K.$$

The current LP formulation we have is tedious. We can reformulate it. An equality constraint $a_i^T x = b_i$ is equivalent to the two constraints $a_i^T x \leq b_i$ and $a_i^T x \geq b_i$. Furthermore, we can rewrite $a_i^T x \leq b_i$ as $(-a_i)^T x \geq -b_i$. Finally, $x_i \leq 0$ or $x_i \geq 0$ are constraints of the form $a_i^T x \geq b_i$ where a_i is the unit vector and $b_i = 0$.

Theorem 1: General form of Linear Program

Any general linear program can be written in the form

$$\min_{\mathbf{x}} \quad \mathbf{cx}$$

$$\text{subject to} \quad A\mathbf{x} \geq \mathbf{b}$$

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

Remark 1.13 *We will see later that this is the definition of a polyhedron.*

Definition 4: Standard Form Linear Program

We say that a linear programming problem is in standard form if we minimise $\mathbf{cx}$ subject to

$$\min_{\mathbf{x}} \quad \mathbf{cx}$$

$$\text{subject to} \begin{cases} A\mathbf{x} = \mathbf{b} \\ \mathbf{x} \geq \mathbf{0}. \end{cases}$$

where $A \in \mathbb{R}^{m \times n}, x \in \mathbb{R}^n, b \in \mathbb{R}^m$.

Lemma 1.14 *Any real number $K \in \mathbb{R}$ can be expressed as the difference of two nonnegative numbers $a - b$.*

Any general linear program can be transformed into standard form.

Theorem 2: Conversion of LP into Standard LP

Any generalised linear program can be reduced into a standard form linear program by

1. Eliminating any unbounded decision variable by expressing it as the difference of two new decision variables. That is $x_j = x_j^+ - x_j^-$ where $x_j^+, x_j^- \geq 0.$;
2. Eliminate inequality constraints through the addition of slack variables or the subtraction of surplus variables. That is, given an inequality constraint

$$\sum_{j=1}^{n} a_{ij}x_j \leq b_i$$

we can add the slack variable s_i so that

$$\begin{cases} \sum_{j=1}^{n} a_{ij}x_j + s_i = b_i \\ s_i \geq 0. \end{cases}$$

Likewise, if we had the constraint

$$\sum_{j=1}^{n} a_{ij}x_j \geq b_i$$

we can add the surplus variable s_i to get

$$\begin{cases} \sum_{j=1}^{n} a_{ij}x_j - s_i = b_i \\ s_i \geq 0. \end{cases}$$

Theorem 1.15 *A cost is an optimal cost to a general linear program if and only if it is an optimal cost to a standard linear program.*

Remark 1.16 *Given an optimal feasible solution $\boldsymbol{x}$, we can construct an equivalent optimal feasible solution $\boldsymbol{y}$ for the other form of the linear program.*

1.2.2 Polyhedra Theory

Definition 1.17 *(Polyhedra). A polyhedron $\mathcal{P}$ is a set $\{x \in \mathbb{R}^n : Ax \geq b\}$ defined by a matrix $A \in \mathbb{R}^{m\times n}$ and a vector $b \in \mathbb{R}^m$. A polyhedron $P \subseteq \mathbb{R}^n$ is bounded if $P \subseteq [-C, C]^n$ for some $C > 0$, and unbounded otherwise.*

Lemma 1.18 *The feasible set of any linear programming problem is a polyhedron.*

Definition 5: Polyhedron in standard form

A polyhedron in the form

$$\{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq 0\}$$

is known as a polyhedron in standard form.

Definition 1.19 *(Convex Hull). The convex hull of a set of vectors $a_1, a_2, ..., a_k \in \mathbb{R}^n$ is the set of all convex combinations of these vectors:*

$$CH(\{a_1, ..., a_k\}) = \{\sum_{i=1}^{l} \langle \lambda_i, a_i \rangle : \sum_{i=1}^{k} \lambda_i = 1 \quad and \quad \lambda_i \geq 0\}.$$

Lemma 1.20 *The convex hull is the smallest convex set containing the set of vectors in the convex hull.*

Definition 6: Hyperplane

A hyperplane is a set $\{x \in \mathbb{R}^n : \langle a, x \rangle = d\}$ for some vector $a \in \mathbb{R}^n$ and scalar $d \in \mathbb{R}$. A **half space** is the set $\{x \in \mathbb{R}^n : \langle a, x \rangle \geq d\}$.

Remark 1.21 *A polyhedron $\mathcal{P}$ (feasible solutions) is the intersection of half-spaces (constraints).*

Theorem 1.22 *(Separating hyperplane theorem). Let $C \subseteq \mathbb{R}^n$ be a non-empty, closed, and convex set. Let $y \in \mathbb{R}^n - C$. Then, there always exists a hyperplane to separate y and C. That is, there exists $a \in \mathbb{R}^n$ and $\alpha \in \mathbb{R}$ such that*

$$a^T x \leq \alpha \leq a^T y$$

for all $x \in C$.

Theorem 1.23 *Every polyhedron is a convex set.*

Definition 7: Extreme point

A vector $x \in \mathcal{P}$ is an extreme point of $\mathcal{P}$ if there does not exist vectors $y, z \in \mathcal{P} \backslash \{x\}$ such that $x \in \{\theta y + (1-\theta)z : \theta \in [0,1], \theta > 0\}$. In otherwords, x cannot be written as a convex combination of another set of vectors in $\mathcal{P}$.

Definition 8: Vertex

A vector $x \in \mathcal{P}$ is a vertex of $\mathcal{P}$ if there exists a vector $c \in \mathcal{P}$ such that

$$\langle c, x \rangle < \langle c, y \rangle$$

for all vectors $y \in \mathcal{P} \backslash \{x\}$.

Remark 1.24 *A vertex x is a point where you can construct a hyperplane $\langle c, x \rangle$ such that every point in the polyhedron is on one side of the hyperplane. Furthermore, this hyperplane only touches the polyhedron at the point x.*

Definition 1.25 *(Active Constraint). If a vector $\boldsymbol{x}^*$ satisfies $\langle a_i, \boldsymbol{x}^* \rangle = b_i$ for some constraint i, we say that the constraint i is **active** or **binding** at $\boldsymbol{x}^*$.*

Theorem 3: Characterisation of active constraints

Let $\mathbf{x}^* \in \mathbb{R}^n$. Let $I = \{i : \langle a_i, \mathbf{x}^* \rangle = b_i\}$ be the set of indices of constraints are active at $\mathbf{x}^*$. Then the following are equivalent

1. There exists n vectors in the set $\{a_i : i \in I\}$ which are linearly independent;
2. The span of the vectors $\mathbf{a}_i$ for $i \in I$ is all of $\mathbb{R}^n$;
3. The system of equations $\langle a_i, \mathbf{x} \rangle = b_i$ for $i \in I$ has an unique solution.

Definition 9: Basic Feasible Solution

Consider a polyhedron P defined by linear equality and inequality constraints and let $\mathbf{x}^* \in \mathbb{R}^n$. The vector $\mathbf{x}^*$ is a **basic solution** if

1. All equality constraints are active;
2. Out of the constraints that are active at $\mathbf{x}^*$, there are n of them that are linearly independent.

If $\mathbf{x}^*$ is a basic solution that satisfies all of the constraints, we say that it is a **basic feasible solution.**

Remark 1.26 *If the number m of constraints used to define a polyhedron is less than the number of decision variables, the number of active constraints at any given point is less than n and hence there are no basic or basic feasible solutions.*

Theorem 4: Characterisation of basic feasible solutions

Let $\mathcal{P}$ be a non-empty polyhedron and $x \in \mathcal{P}$. The following statements about x are equivalent:

1. x is a vertex;
2. x is an extreme point;
3. x is a basic feasible solution.

Theorem 1.27 *Let $\mathcal{P} \subseteq \mathbb{R}^n$ be a bounded non-empty polyhedron. Then*

$$\mathcal{P} = CH(\{x \in \mathbb{R}^n : x \textit{ is an extreme point of } \mathcal{P}\}).$$

Definition 1.28 *(Polyhedra in standard form). A polyhedron in standard form is the set*

$$\mathcal{P} = \{x \in \mathbb{R}^n : Ax = b \quad \textit{and} \quad x \geq 0\}$$

where $b \in \mathbb{R}^m$ and the matrix $A \in \mathbb{R}^{m \times n}$ is full rank.

Lemma 1.29 *Any linear program can be turned into an equivalent linear program in standard form.*

Theorem 1.30 *Consider the constraints $A\boldsymbol{x} = \boldsymbol{b}$ and $\boldsymbol{x} \geq \boldsymbol{0}$ where $A \in \mathbb{R}^{m \times n}$. Assume that A has linearly independent rows. A vector $x \in \mathbb{R}^n$ is a basic solution if and only if we have $A\boldsymbol{x} = \boldsymbol{b}$ and there exists indices $B_1, ..., B_m$ such that*

1. *The **columns** $A_{B_{(1)}}, ..., A_{B_{(m)}}$ are linearly independent;*
2. *If $i \neq B_1, ..., B_m$, then $x_i = 0$.*

Definition 1.31 *(Adjacent Basic Solutions). Two distinct basic solutions are adjacent if there are n - 1 linearly independent constraints that are active at both of them. Furthermore, two bases are adjacent if they share all but one basic column.*

According to our definition, at a basic solution, we must have n linearly independent active constraints.

Definition 10: Degenerate solution

A basic solution $x \in \mathbb{R}^n$ is said to be degenerate if more than n of the constraints are active at **x**. Alternatively, a basic feasible solution **x** is degenerate if more than n - m of the components of **x** are 0.

Hence, a degenerate basic solution is one where we have the number of active constraints to be greater than necessary.

COMP3530: Discrete Optimisation

2. Simplex

2.3 Simplex

2.3.1 Optimality Conditions

If a linear program in standard form has an optimal solution, then there exists a basic feasible solution that is optimal. The simplex method is based on this fact and searches for optimal solution by moving from one basic feasible solution to another, along the edges of the feasible set in a cost reducing direction. We assume we have the standard form

$$\min_{\mathbf{x}} \quad \mathbf{cx}$$

$$\text{subject to} \begin{cases} A\mathbf{x} = \mathbf{b} \\ \mathbf{x} \geq 0 \end{cases}$$

We let P be the corresponding feasible set.
The following theorem states why our idea of locally looking for an optimal solution is a good idea.

Theorem 5: Global Optimiality for Linear Programming

For a linear programming problem, local optimality imlpies global optimality as we are minimising a convex function over a convex set.

Suppose we are at a point $\mathbf{x} \in P$ and we are contemplating moving away from $\mathbf{x}$. The following tells us where we should move.

Definition 11: Feasible Direction

Let x be an element of the polyhedron P. A vector $d \in \mathbb{R}^n$ is said to be a feasible direction at x if there exists a positive scalar λ such that $x + \lambda d \in P$.

Definition 12: Basis

Let the LP be $\langle C^T, x\rangle$ subject to $Ax = b$ and $x \geq 0$ where $x \in \mathbb{R}^n$ and $A \in \mathbb{R}^m \times \mathbb{R}^n$. Let B be a subset of m indices from $\{1, ..., n\}$. Denote $A_B \in \mathbb{R}^m \times \mathbb{R}^m$ as the A matrix indexed by B. If A_B is nonsingular, the columns indexed by B are a **basis** of the **column space** of A. We call B a basis of the LP.

Remark 2.32 *The matrix A has at most $\binom{n}{m}$ bases.*

Definition 13: Basic Solution

The vector $x \in \mathbb{R}^n$ is a basic solution if the vectors $\{a_i : x_i \neq 0\}$ are linearly independent or A_B is nonsingular.

Definition 2.33 *(Basic Feasible Solution with Basis B). Given a basis B, it is a basic feasible solution if for all $j \notin B : x_j = 0$ or $x \geq 0$.*

We make some assumptions first on the simplex algorithm we are about to develop.

1. We are provided with an initial feasible basis.
2. Every feasible basis B is non-degenerate $x_B = A_B^{-1} b > 0$.
3. The feasible region is bounded, that is, there exists a C such that $\{x \in \mathbb{R}^n : Ax = b, x \geq 0\} \subseteq [0, C]^n$,

Definition 14: Reduced cost

Let $\mathbf{x}$ be a basic solution, let $\mathbf{B}$ be an associated basis matrix, and let $\mathbf{c}_B$ be the vector of costs of the basic variables. For each feasible direction j, we define the **reduced cost** $\bar{c}_j$ of the variable x_j according to the formula

$$\bar{c}_j = c_j - \mathbf{c}_B \mathbf{B}^{-1} \mathbf{A}_j.$$

Remark 2.34 *The first term c_j is the cost per unit increase of the variable x_j (the candidate feasible direction) whilst the second term is the cost of the compensating change in the basic variables necessitated by the constraint $\boldsymbol{Ax} = \boldsymbol{b}$.*

Theorem 6: Optimality Conditions

Consider a basic feasible solution $\mathbf{x}$ associated with a basis matrix $\mathbf{B}$, and let $\bar{\mathbf{c}}$ be the corresponding vector of reduced costs.

1. If $\bar{\mathbf{c}} > 0$ then $\mathbf{x}$ is optimal.
2. If $\mathbf{x}$ is optimal and nondegenerate, then $\bar{\mathbf{c}} \geq 0$.

Hence, for a given basic solution, we have two conditons for it to be optimal. First is that it satisfies the original LP constraints and second, the reduced cost is positive.

Theorem 7: Conditions for Optimal base

A basis matrix $\mathbf{B}$ is said to be optimal if

1. $A_B^{-1} \mathbf{b} \geq 0$
2. $\bar{c} = \mathbf{c} - \mathbf{c}_B \mathbf{B}^{-1} \mathbf{A} \geq \mathbf{0}$.

Theorem 2.35 *When moving along feasible direction, we want to move to $\boldsymbol{x} + \theta^* \boldsymbol{d}$. The formula for the scalar is*

$$\theta^* - \min_{i=1,2,\ldots,m : d_{B(i)} < 0} -\frac{x_i}{d_i}.$$

2.3.2 Simplex

Theorem 2.36 *Suppose we remove index i from the basis and added in index ℓ. The new matrix $\overline{B}$ is a basis matrix and $A_{\overline{B}}$ is linearly independent. Furthermore, the vector $\boldsymbol{y} = \boldsymbol{x} + \theta^* \boldsymbol{d}$ is a basic feasible solution associated with the basis matrix $\overline{B}$.*

COMP3530: Discrete Optimisation

3. Linear Programming Applications

3.4 Linear Programming Applications

3.4.1 Linear Programming Applications

Definition 3.37 *(Production Planning). We have demand project $d_1, ..., d_n$. We need to decide how much to produce each time period whilst minimising storage cost. Let x_i be the number of units produced at period i and s_i be the number of units needed to carry from period i to i+1. Hence,*

$$\textbf{\textit{minimise}} \sum_i s_i$$

$$\textbf{\textit{subject to}}\ s_i = s_{i-1} + x_i - d_i \quad i = 2, ..., n$$

$$s_i \geq 0 \quad i = 1, ..., n$$

$$x_i \geq m \quad i = 1, ..., n$$

$$x_i \leq M \quad i = 1, ..., n.$$

Definition 3.38 *(Line Fitting). We want to fit a line $y = ax + b$ through a set of points $(x_1, y_1), ..., (x_n, y_n)$ by adjusting a and b. The cost function is the absolute errors $\sum_i |y_i - ax_i - b|$.*

$$\textbf{\textit{minimise}} \sum_i e_i$$

$$\textbf{\textit{subject to}}\ e_i \geq y_i - ax_i - b \quad i = 1, ..., k$$

$$e_i \geq -(y_i - ax_i - b) \quad i = 1, ..., k$$

$$\textbf{\textit{a,b are free}}.$$

Definition 3.39 *(Flow Problems). Given a directed graph G = (V,E) with edge capacities $c : E \to \mathbb{R}^+$. A circulation is an edge function $f : E \to \mathbb{R}^+$ that obeys flow conversation at every vertex whilst obeying edge capacity constraints. The maximum flow problem is to find a circulation maximising the flow along an edge (t,s) called the sourrce and the sink. Let $f_{u,v}$ be the flow values for each $(u, v) \in E$. Hence,*

$$\textbf{\textit{minimise}}\ f_{t,s}$$

$$\textbf{\textit{subject to}} \sum_{(u,v)\in\delta^{in}(u)} = \sum_{(u,v)\in\delta^{out}(u)} \quad u \in V$$

$$f_{u,v} \leq c_{u,v} \quad (u, v) \in E$$

$$f_{u,v} \geq 0 \quad (u, v) \in E.$$

Definition 3.40 *(Vertex Cover). We are given an undirected graph G=(V,E) and need to select smallest subset $C \subseteq V$ such that every edge has at least one endpoint in C. Let $x_u \in \{0,1\}$ be an indicator variable where $x_u = 1$ if $u \in C$ and 0 otherwise. Hence,*

$$\textbf{\textit{minimise}} \sum_{u \in V} x_u$$

$$\textbf{\textit{subject to}}\ x_u + x_v \geq 1 \quad (u,v) \in E$$

$$x_u \in \{0,1\} \quad u \in V.$$

Definition 3.41 *(Edge cover). An edge cover of a graph G is a subset of edges of E which contains all nodes of G.*

Remark 3.42 *Edge cover is using edges to cover graph whereas vertex cover is using verticies to cover graph.*

COMP3530: Discrete Optimisation

4. Duality Theory

4.5 Duality Theory

4.5.1 Motivation

Let's start off with a LP minimisation in standard form

$$\min_x \quad c.x$$

$$\text{subject to } \begin{cases} Ax = b \\ x \geq 0. \end{cases}$$

What we can do is take the first constraint and lift it into the objective function and hence create a Lagrangian in the process.

$$\min_x \quad c.x + \lambda(b - Ax)$$

$$\text{subject to } x \geq 0$$

Now, let us denote x^* as the optimal value to the original LP and $t(\lambda)$ as the optimal value to our Lagrangian formulation of the problem.

Lemma 4.43 *The optimal to the original problem is an upper bound for the objective value of our Lagrangian*

$$t(\lambda) \leq cx^*,$$

Proof:

$$t(\lambda) = \min_{x \geq 0} \ c.x + \lambda(b - Ax) \leq cx^* + \lambda(b - Ax^*)$$

$$= cx^*$$

where the last equality follows as $Ax^* = b$. ■

Hence, we see that different values for the Lagrangian multiplier (which can be thought of as the price for violating the constraint) gives us a different value to $t(\lambda)$. As $t(\lambda)$ is lower bounded by the optimal cost of our original problem, we therefore want to maximise $t(\lambda)$ to get the tightest lower bound to our original problem. Hence, we want to solve the problem

$$\max_\lambda t(\lambda)$$

$$\text{subject to no constraint}$$

This is known as the **dual**. Now, we note that

$$t(\lambda) = \min_{x \geq 0} \ c.x + \lambda(b - Ax) = \lambda b + \min_{x \geq 0} \Big(c - \lambda A\Big)x.$$

Now looking at the last term, we have that

$$\min_{x\geq 0}\Big(c-\lambda A\Big)x=\begin{cases}0 & \text{if } c-\lambda A\geq 0\\ -\infty & \text{otherwise.}\end{cases}$$

Hence, as we are trying to maximise the dual, we do not want to consider values of λ where $c\leq \lambda A$ or else we get $-\infty$ in our objective value. Hence, we can therefore conclude our dual linear program is

$$\max_{\lambda}\lambda b$$

$$\text{subject to } \lambda A\leq c.$$

If we select the price λ properly, the optimal solution to the constrained problem is also the optimal solution to the unconstrained problem. Hence, we need to figure out the prices such that the absence/presence of constraints does not affect the optimal solution. We can find out what the prices are by solving the linear program which is the dual of the original problem.

4.5.2 Duality Theory

Given a standard primal problem to minimise a program, the aim of the dual is to find the tighest lower bound for the primal program.

Definition 4.44 *(Primal/Dual Program). Let $A\in\mathbb{R}^{m\times n}$. The primal program is*

$$min \quad \sum_{j=1}^{n}c_jx_j$$

$$\begin{aligned} s.t.(y_i)a_i^Tx\geq b_i &\quad \forall i\in M_1\\ (y_i)a_i^Tx\leq b_i &\quad \forall i\in M_2\\ (y_i)a_i^Tx=b_i &\quad \forall i\in M_3\\ x_j\geq 0 &\quad \forall i\in N_1\\ x_j\leq 0 &\quad \forall i\in N_2\\ x_j \quad free &\quad \forall i\in N_3\end{aligned}$$

We define the dual as

$$max \quad \sum_{i=1}^{m}b_iy_i$$

$$\begin{aligned} s.t.(x_j)y^TA_j\geq c_j &\quad \forall j\in N_1\\ (x_j)y^TA_j\leq c_j &\quad \forall j\in N_2\\ (x_j)y^TA_j=c_j &\quad \forall j\in N_3\\ y_i\geq 0 &\quad \forall i\in M_1\\ y_i\leq 0 &\quad \forall i\in M_2\\ y_i \quad free &\quad \forall i\in M_3\end{aligned}$$

Theorem 8: Weak Duality

Let x and y be feasible solutions to the primal (P) and the dual (D). Then,

$$\sum_{j=1}^{n} c_j x_j \geq \sum_{i=1}^{m} b_i y_i.$$

From weak duality, we get a strong test for whether does a region have feasible solutions.

Proposition 1: Weak Duality Corollary

1. If the primal (P) is unbounded ($-\infty$), then the dual (D) is infeasible.
2. If the dual (D) is unbounded (∞), then the primal (P) is infeasible.

We now state something 'obvious'.

Lemma 4.45 *The dual of the dual is the primal.*

We now see that when we maximise the dual and get a bounded objective value, then we are done as it will equal the minimisation of the primal. We state this more formally.

Theorem 9: Strong Duality

If the primal (P) is feasible and has bounded objective, then so does the dual (D) and they have the **same optimal value**.

Proof:(Sketch). Use the fact that the dual is bounded from above by the primal. Then, show by Farkas lemma that the optimal value to the dual is greater than the optimal value of the primal. Hence this shows that the optimal value to the dual is **equal** to the primal. ■

Theorem 4.46 *(Complementary Slackness). Let x and y be a pair of optimal solutions to the primal (P) and the dual (D). Then*

1. *$x_j = 0$ or $y^T A_j = C_j$ for all j;*
2. *$y_i = 0$ or $a_i^T x = b_i$ for all j.*

4.5.3 Farkas Lemma

We are interested in understanding when is a linear program is feasible. That is, when is the set of constraints non-empty. First, we recall Gaussian elimination. Suppose we had a system to solve $Ax = b$. The question is does there exists $x \in \mathbb{R}^n$ such that $Ax = b$? Either this is the case or else, we can multiply the system by a vector $y \in \mathbb{R}^m$ such that $y^T Ax = 0$ BUT $y^T b \neq 0$. As a result, this system cannot be solve because $y^T Ax = 0 \neq y^T b$. This vector y is called a **certificate of infeasibility.**

Theorem 4.47 *(Gauss). Let $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. Then exactly one of the following holds.*

1. *There exists $x \in \mathbb{R}^n$ such that $Ax = b$.*
2. *There exists $y \in \mathbb{R}^m$ such that $y^T A = 0$ but $y^T b \neq 0$.*

Taking this idea, we want to apply it to our standard form linear programs where $Ax = b$ and $x \geq 0$. This is what Farkas lemma does and is used for certifying infeasibility.

Theorem 10: Farka's Lemma

Let $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. Then exactly one of these holds

1. There exists $x \in \mathbb{R}^n : Ax = b$ and $x \geq 0$
2. There exists $y \in \mathbb{R}^m : A^T y \geq 0$ and $\langle b, y \rangle < 0$.

COMP3530: Discrete Optimisation

5. Linear Programming Applications II

5.6 Linear Programming Applications II

5.6.1 Playing Zero-Sum Games

We look at two-player games.

Definition 5.48 *(Zero-Sum Games). We have a tuple (S_1, S_2, P) where*

1. *$S_1 = \{a_1, a_2, ..., a_n\}$ are strategies for player I;*
2. *$S_2 = \{b_1, b_2, ..., b_m\}$ are strategies for player II;*
3. *$P \in \mathbb{R}^{n\times m}$ is a pay-off matrix.*

If player I picks a_i and player II picks b_j, we say that player I gets pay-off $P_{i,j}$ and player II gets pay-off $-P_{i,j}$.

Definition 5.49 *(Outcomes). An outcomes (a_i, b_j) is an equilibrium*

1. *$P_{ij} \geq P_{kj}$ for all $k = 1, ..., n$;*
2. *$-P_{ij} \geq -P_{ik} = P_{ij} \leq P_{ik}$ for all $k = 1, ..., m$.*

Definition 5.50 *(Mixed Stategy). A mixed strategy for player I is the probability mass function: $x_1, ..., x_n \geq 0$ and $\sum x_i = 1$. Likewise, a mixed strategy for player II is the probability mass function $y_1, ..., y_m \geq 0$ and $\sum y_j = 1$. We define the expected pay-offs as*

$$\text{Expected payoff for player I} = \sum_i \sum_j x_i y_j P_{ij} = -\text{Expected payoff for player II.}$$

We say that (x, y) is a mixed equilibrium if

$$\sum_i \sum_j x_i y_j P_{ij} \geq \sum_i \sum_j \hat{x}_i y_j P_{ij}$$

for all $\hat{x}$ where $\hat{x}_1, \hat{x}_2, ..., \hat{x}_n \geq 0$ and $\sum \hat{x}_i = 1$ and

$$\sum_i \sum_j x_i y_j P_{ij} \geq \sum_i \sum_j x_i \hat{y}_j P_{ij}$$

for all $\hat{y}$ where $\hat{y}_1, \hat{y}_2, ..., \hat{y}_n \geq 0$ and $\sum \hat{y}_i = 1$.

Theorem 5.51 *Every two-player zero-sum game has a mixed equilibrium.*

Claim 5.52 *Let (x^*, z^*) and $y^*, z^*)$ be an optimal primal-dual par. Then, (x^*, y^*) form a mixed equilibrium.*

5.6.2 Bipartite Matching

Definition 5.53 *(Matching). A matching M is a subset of edges such that no two edges are incident on the same vertex*

$$deg_M(u) \leq 1 \quad \forall u \in V.$$

Definition 15: Bipartite matching

Let $G = (V, E)$ be a bipartite graph and $w : E \to \mathbb{R}$. The maximum weight bipartite matching problem is

$$max \sum_{e \in E} w_e x_e$$

$$s.t. \begin{cases} \sum_{e \in \delta(u)} x_e \leq 1 & \forall u \in V \\ x_e \in \{0, 1\} & \forall e \in E. \end{cases}$$

We can relax this program.

Definition 5.54 *(Bipartite matching relaxation). The integer relaxation of the maximum weight bipartite matching problem is*

$$max \sum_{e \in E} w_e x_e$$

$$s.t. \begin{cases} \sum_{e \in \delta(u)} x_e \leq 1 & \forall u \in V \\ x_e \geq 0 & \forall e \in E. \end{cases}$$

Lemma 5.55 *The solution to the linear relaxation is an upper bound to the original bipartite matching problem.*

Proof: We are removing constraints and hence should be able to obtain a larger maximum. ■

We now move onto a remarkable theorem.

Theorem 11: Integrality of bipartite matching

Every basic feasible solution to the linear relaxation of bipartite matching is **integral.** Furthermore, the value of the linear relaxation of bipartite matching is **equal** to the value of the integer bipartite matching.

5.6.3 König's Theorem

We now study maximum **unweighted** bipartite matching.

$$max \sum_{e \in E} x_e$$

$$s.t. \begin{cases} \sum_{e \in \delta(u)} x_e \leq 1 & \forall u \in V \\ x_e \geq 0 & \forall e \in E. \end{cases}$$

We now take the dual

$$min \sum_{v\in V} p_v$$

$$s.t. \begin{cases} p_u + p_v \geq 1 \quad \forall (u,v) \in E \\ p_u \geq 0 \quad \forall u \in V. \end{cases}$$

We now state a theorem about the **dual**.

Theorem 5.56 *Every basic feasible solution of the dual to maximum unweighted bipartite matching is* ***integral.***

We study something similar to the dual.

Definition 16: Vertex Cover

Let G=(V,E) be a graph and $w : E \to \mathbb{R}$. The vertex cover is a subset of vertices is such that every edge in the graph has at least one vertex in the cover.

$$min \sum_{v\in V} w_v x_v$$

$$s.t. \begin{cases} x_u + x_v \geq 1 \quad \forall (u,v) \in E \\ x_v \in \{0,1\} \quad \forall v \in V. \end{cases}$$

Theorem 12: Koenig's Theorem

In every bipartite graph, the cardinality of the largest matching equals the cardinality of the smallest vertex cover. This is equal to $|V|$.

COMP3530: Discrete Optimisation

6. Integral Polyhedra

6.7 Integral Polyhedra

6.7.1 Totally Unimodular Matrices

We are now interested in what constraints do we need on the constraint matrix such that every basic feasible solution is integral. That is, given a integer program that we perform linear relaxation on, what constraints do we need on the constraint matrix A and vector b such that we still always get an integral solution.

Definition 6.57 *(Integral Polyhedra). We say that a polyhedron P is integral if all of its extreme points are integral. In other words, the integer program*

$$\begin{gathered} min \quad \langle c, x\rangle \\ such\ that\ Ax = b \\ x \in \mathbb{Z}^n_+ \end{gathered}$$

is equivalent to

$$\begin{gathered} min \quad \langle c, x\rangle \\ such\ that\ Ax = b \\ x \geq 0. \end{gathered}$$

Remark 6.58 *Hence, we are interested for when can we perform linear relaxation on the integer program and still get an integer solution.*

If we had a bounded objective function then there exists a basic feasible solution x that is optimal. We denote the basis of this by B. Then we are interested for when $x_B = A_B^{-1}b$ is integral. Clearly, x_B is integral if A_B^{-1} is integral.

Definition 17: Unimodular

We say that a full-row rank matrix $A \in \mathbb{Z}^{m\times n}$ is unimodular if for all basis B of A, then $det(A_B) \in \{1, -1\}$.

Corollary 6.59 *Unimodularity of a matrix is a sufficient condition for the integrality of A_B^{-1}.*

Definition 6.60 *(Cramer's Rule). Let M be an invertible matrix. Then, Cramer's rule states that*

$$M^{-1} = \frac{Adj(M)}{det(M)}.$$

Lemma 6.61 *If M is integral, then Adj(M) is integral.*

Theorem 13: Integrality of Standard Linear Programs

A full row-rank matrix $A \in \mathbb{R}^{m\times n}$ is unimodular if and only if all extreme points of the polyhedron $P = \{x \in \mathbb{R}^n : Ax = b, x \geq 0\}$ are integral for all $b \in \mathbb{Z}^m$.

The above theorem only holds for linear programs in standard form. However, we can generalise the above theorem for LPs not in standard form by imposing additional conditions on the constraint matrix.

Definition 18: Totally unimodular

We say a matrix $A \in \mathbb{Z}^{m\times n}$ is totally unimodular if for every square submatrix A' of A, we have that

$$det(A') \in \{-1, 0, 1\}.$$

Hence, we can now generalise the theorem on integrality of linear programs.

Theorem 14: Integrality of Linear Programs

A matrix A is totally unimodular if and only if the polyhedron's extreme points

$$P = \{x \in \mathbb{R}^n : Ax \leq b, x \geq 0\}$$

is integral for all $b \in \mathbb{Z}^m$.

6.7.2 Properties of Totally Unimodular Matrices

We now analyse furhter properties of totally unimodular matrices.

Theorem 6.62 *Let A be a totally unimodular matrix. The following properties hold*

1. *Entries of $A \in \{-1, 0, 1\}$;*
2. *A^T is a totally unimodular matrix;*
3. *The matrix $\begin{bmatrix} A & I \end{bmatrix}^T$ is a totally unimodular matrix;*
4. *The matrix $\begin{bmatrix} A & -A \end{bmatrix}^T$ is a totally unimodular matrix.*

Corollary 6.63 *If A is totally unimodular matrix, the extreme points of the polyhedron*

$$P = \{x \in \mathbb{R}^n : Ax \leq b, Ax \geq d, x \leq ub, x \geq lb\}$$

is integral as well where ub and lb are integral upper and lower bounds for the vector x.

Remark 6.64 *If we have a polyhedron P that is defined by a totally unimodular matrix and define a restriction of x by $Q = [lb_1, ub_1] \times ... \times [lb_n, ub_n]$ where each endpoint is integral, then $P \cap Q$ is also integral.*

Corollary 6.65 *Let the constraint matrix A be totally unimodular. Let c and b be integral. Then both the primal program $\min\{\langle c, x\rangle : Ax \geq b, x \geq 0\}$ and the dual program $\max\{\langle b, y\rangle : A^T y \leq c, y \geq 0\}$ are integral.*

6.7.3 Alternative Characterisation of Totally Unimodular Matrices

We look at alternative characterisations of totally unimodular matrices to make our lives easier.

Definition 19: Equitable Bi-Colouring

A columm bi-colouring of a matrix M is a partition of its columns into red and blue columns. We call the bi-colouring **equitable** if for every row

$$|\text{sum of blue entries} - \text{sum of red entries}| \leq 1.$$

We now have a new way to characterise totally unimodular matrices and hence linear relaxations which has integral solutions.

Theorem 15: Equitable bi-colouring characterisation of integrality

A matrix $A \in \mathbb{Z}^{m\times n}$ is totally unimodular if and only if every column submatrix of A admits an equitable column bi-colouring.

We can now show that certain matrices are totally unimodular.

Theorem 16: Bipartite matching is totally unimodular

The constraint matrix of a bipartite matching problem is totally unimodular. That is, the polyhedron

$$P = \{x \in \mathbb{R}^{|E|} : \sum_{e\in\delta(u)} x_e \leq 1, x_e \geq 0\} = \{x \in \mathbb{R}^{|E|} : Ax \leq 1, x \geq 0\}$$

is totally unimodular.

Theorem 17: Circulation problem is totally unimodular

The constraint matrix associated with the circulation problem polyhedron

$$P = \{f \in \mathbb{R}_+^{|E|} : \sum_{e\in\delta^{in}(u)} f_e = \sum_{e\in\delta^{out}(u)} f_e \quad \forall u \in V\}$$

is totally unimodular.

6.7.4 Subset Selection Problems

We now look at the broad problem of subset selection problems.

Definition 6.66 *(Feasible Subsets). Let U be a universal set of elements and $I \subseteq 2^U$ be*

$$I = \{S \subseteq U : \textit{vectors in } S \textit{ are linearly independent}\}.$$

I is referred to as a collection of feasible subsets.

Definition 20: Subset System

Let (U, I) be the pair where U is the universal set and $I \subseteq 2^U$ is a collection of feasible sets of U. The pair is a subset system if $I \neq \emptyset$ and for any $S \subset T \subseteq U$, it holds that $T \in I$ implies that $S \in I$.

Definition 6.67 *(Subset Selection Problem). Given a cost function $c : U \to \mathbb{R}$, define the canonical optimisation problem for (U, I). Find a subset $A \subseteq U$ such that $A \in I$ that maximizes $c(A) = \sum_{x \in A} c(x)$.*

Definition 21: Rank function

The rank function of (U, I) is $r : 2^u \to \mathbb{Z}_+$ where

$$r(S) = \max_{A \subseteq S, A \in I} |A|.$$

Remark 6.68 *The rank function looks for the largest feasible subset of a given set S.*

We can now express the integer formulation for the canonical optimisation.

$$\max_x \sum_{j \in U} c_j x_j$$

$$s.t. \quad \sum_{j \in S} x_j \leq r(S) \quad S \subseteq U$$

$$x_j \in \{0, 1\} \quad \forall j \in U.$$

Definition 22: Matroid Exchange Axiom

A system (U, I) satisfies the matroid exchange axiom if for all subsets $S, T \in I$, if $|S| < |T|$, then there exists an element $j \in T \backslash$S such that $S \cup \{j\} \in I$.

Definition 6.69 *(Matroid). The subset system (U, I) is called a matroid if it satisfies the matroid exchange axiom.*

Theorem 6.70 *For every subset $S \subset U$, all maximal independent sets in S have equal size.*

Theorem 18: Integrality of matroid

Let (U, I) be a matroid, then the following polyhedron is integral

$$\{x \in \mathbb{R}_+^{|u|} : x(S) \leq r(S) \quad \forall S \subseteq U\}.$$

Theorem 6.71 *(LP) is integral for acyclic subgraphs. (LP) is integral if (U, I) has the matroid property.*

Definition 23: Independence Oracle

An independence oracle is an algorithm which, when given a set $S \subseteq U$, tells us whether $S \in I$.

Remark 6.72 *The independence oracle is easy to implement for forests in graphs (check S for cycles) and for linearly independence sets of vectors (use Gaussian elimination).*

Theorem 19: Optimality of Greedy algorithm under matroid system

Suppose that $M = (U, I)$ is a subset system and that we have an oracle to determine whether a given set is dependent. Define the Greedy algorithm to iteratively adds the cheapest element of U that maintains independence. Then, the Greedy algorithm produces a maximally independent set S of minimal cost for every nonnegative cost function on U if and only if M is a matroid.

Theorem 6.73 *Let F be the independent set returns by the greedy algorithm. Then, $w(F) \geq w(F')$ for all $F' \in I$.*

COMP3530: Discrete Optimisation

7. Integer Programming

7.8 Integer Programming

7.8.1 Introduction to Integer Programs

Integer-valued variables can represent quantities that cannot be subdivided, model yes-no decisions, combinatorial constraints, and much more. An integer linear program is an optimisation problem that has the form of a linear program with the additional restriction that the values of all variables must be integers.

Definition 24: Integer Linear Program

An integer linear program has the form

$$\text{minimize } \langle c, x\rangle$$

$$\text{subject to } Ax \geq b$$
$$x \in \mathbb{Z}^n_+.$$

Remark 7.74 *The* ***linear relaxtion*** *of an integer linear program is when we let $x \in \mathbb{R}^n_+$. That is, we remove the restriction that our variables must be integers.*

Theorem 20: Integer program as a subset of linear relaxation

Let us define the feasible region of the integer program as

$$IPF = \{x \in \mathbb{Z}^n_+ : Ax \geq b\}.$$

Let us also define the feasible region of the linear relaxation as

$$LPF = \{x \in \mathbb{R}^n_+ : Ax \geq b\}.$$

Then clearly

$$IPF \subseteq LPF.$$

Corollary 7.75 *As $IPF \subseteq LPF$, then it follows that the value of the integer program is less than or equal to the value of its relaxation.*

It is worth noting that you can't simply run a linear program and then round to the nearest integer to solve an integer linear program. There are many instances in which the optimal solution to the integer linear program is completely different to the integer rounding of the optimal solutions of a linear program.

The trick to modelling integer programs is to decompose your problem into yes-no decisions, which are represented by $\{0, 1\}$ variables or into quantities, which are represented by non-integer variables.

7.8.2 Branch and Bound Framework

Proposition 7.76 *Every feasible solution to the Integer Program is also a feasible solution for the Linear Program relaxtion.*

Proof: The LP relaxtion only **removes** restrictions and hence every solution to the Integer Program should be a solution to the Linear Program relaxtion. ■

Corollary 7.77 *The optimal solution to the Linear Program relaxation cannot be worse than the optimal solution to the Integer Program.*

If the optimal solution to the Linear Program relaxation happens to satisfy the integer domain restrictions to the Integer Program, then that **is** the optimal Integer Program solution. Otherwise, the optimal objective value for the Linear Program relaxtion gives a **bound** for the optimal objective value of the Integer Program.

Proposition 7.78 *For a maximisation (minimisation) Integer Program, the optimal objective value of the Linear Program relaxation gives an upper (lower) bound.*

Definition 25: Branch and BoundFramework

Let S be our polyhedron $S = S_1 \cup S_2 \cup ... \cup S_k$. The branch and bound framework is to optimise over each subset S_i separately by computing $z_i = \min\{\langle c, x\rangle : x \in S_i\}$ and take the minimum of the minimums of

$$z = \min_i z_i.$$

We now need to bound the branch of our problem. That is, for each z_i, we let $\underline{z}_i$ be the lower bound and $\overline{z}_i$ be the upper bound. Hence, for each decomposition $S = S_1 \cup ... \cup S_k$, we are interested in computing the bounds

$$\underline{z}_i \leq \min\{\langle c, x\rangle : x \in S_i\} \leq \overline{z}_i.$$

In particular, we use linear relaxation to derive the lower bound. Meanwhile, we use heuristics to derive the upper bound. If for a particular S_i, the problem is infeasible ($S_i = \emptyset$), then we set $\underline{z}_i = \infty$. If we can't find an upper bound, then we set $\overline{z}_i = \infty$.
We can now define $\underline{z} = \min_i \underline{z}_i$ and $\overline{z} = \min_i \overline{z}_i$. Hence, we get

$$\underline{z} \leq z \leq \overline{z}.$$

Theorem 21: Rules for pruning subproblem

We have three rules for pruning the subproblems in the decomposition.

1. Prune by optimality. If we know that $\underline{z}_i = \overline{z}_i$, then we know z_i.
2. Prune by infeasibility. If $S_i = \emptyset$, then $(\underline{z}_i = \infty)$ as we shouldn't prune it.
3. Prune by bound. If $\underline{z}_i \geq \overline{z}$, that is, if we have already found a solution with cost less than z_i, then there is no point in further decomposing S_i.

7.8.3 Branch and Bound with Linear Programming

The branch and bound framwork works well when we apply linear programming as we can compute the lower bounds quite easily. However, we need to resort to heuristics for computing the upper bounds. We have 4 things we need to specify.

1. Decide do we want to decompose subproblems in a depth-first or breadth-first fashion. The latter requires more memory but allows for more pruning and tighter upper and lower bounds.
2. We need to figure out how do we actually want to decompose S into subproblems.
3. If we solve the linear relaxation, it may be the case that adding constraints back may cause a violation of our problems. However, deciding to optimise the dual means that we only gain new variables as we add more constraints back into the primal.

7.8.4 Strengthening Linear Relaxations

To ensure our linear-programming-based branch and bound scheme works well, we need to make sure we have a **strong linear relaxation**.

Definition 26: Tighter Linear Relaxtion

Let S be a discrete set of feasible solutions. Let Q and P be polyhedra, that is, different formulations of S. Suppose $Q \subset P$. Then, we say that Q is a **tighter linear relaxtion of S**. We then write $S = Q \cap \mathbb{Z}^n$.

We want to work with **tighter linear relaxation** for our subproblems as that will give us better lowerbounds. However, the tradeoff is that it means we may impose more constraints and hence solving the relaxation is harder.

Proposition 2: Test for tighter linear relaxtion

Suppose we had two polyhedrons P and Q. To show that Q is a tighter linear relaxation than P, we need to find a point x such that $x \in P$ and $x \notin Q$.

COMP3530: Discrete Optimisation

8. Large Scale Optimisation

8.9 Large Scale Optimisation

8.9.1 TSP

First, recall a useful definition.

Definition 8.79 *(Hamiltonian cycle). A Hamiltonian cycle is a cycle in an undirected graph that passes through each node exactly once.*

Definition 27: Travelling Salesman Problem

Given an undirected complete weighted graph, the TSP is the problem of finding a minimum cost Hamiltonian cycle.

Proposition 3: Showing a problem is hard

To show problem X is hard, take a known NP-hard problem Y and show a polynomial time algorithm that converts problem Y into X.

If we can take a known hard problem and reduce it to our problem in polynomial time and solve our problem, this means that it is easy to solve the known hard problem which is a contradiction.

8.9.2 Large Number of Constraints

The TSP is an example of a problem with a large number of constraints.

Definition 8.80 *(Travelling-Salesman Problem). We define the TSP as*

$$min \sum_{e \in E} w_e x_e$$

$$subject\ to \sum_{e \in \delta(u)} x_e = 2 \quad \forall u \in v$$

$$\sum_{e \in \delta(s)} x_e \geq 2 \quad \forall \emptyset \subset s \subset v$$

$$x_e \geq 0 \quad \forall e \in E$$

where ϕ is a subset of constraint vertices.

Definition 8.81 *(TSP Polyhedron and Relaxtion). We define the TSP polyhedron P to be*

$$P = \{x \in \mathbb{R}^{|E|} : x(\delta(u)) = 2 \quad \forall u \in V, x(\delta(s)) \geq 2 \quad \forall \phi \subset s \subset v\}.$$

We define the relaxtion of the TSP polyhedron Q to be

$$P = \{x \in \mathbb{R}^{|E|} : x(\delta(u)) = 2 \quad \forall u \in V\}.$$

We can optimise over Q instead in hopes of getting an optimal solution in P.

Definition 28: Separation Oracle

A separation oracle for P takes as input a solution x and either says $x \in P$ or $x \notin P$ and produces a violated constraint.

8.9.3 Large Number of Variables

The edge colouring problem is an example of a problem with a large number of variables. To solve problems with large number of variables, we can take the dual of the problem in order to turn the large number of variables into a large number of constraints and utilise the same approach as the last section.

First, we recall some things.

Definition 29: Matching

A **matching** M on a graph G is a subset of E which does not contain any edges with a node in common. A **maximum matching** is a matching on the graph G with the highest possible cardinality. A **perfect matching** consists of edges which cover all the nodes of a graph.

Lemma 8.82 *A perfect matching has cardinality* $\frac{|V|}{2}$.

Definition 8.83 *(Edge Colouring). Given $G = (V, E)$, we want to partition E into matching $M_1, M_2, ..., M_k$ where the objective is to minimise the number of matchings used (k). A basis is defined by $|E|$ matching $B = \{M_{b_1}, M_{b_2}, ..., M_{b_{|E|}}\}$. Hence, we have*

$$\begin{aligned} &min \sum x_M \\ &subject\ to \sum_{M_i \in M} x_M = 1 \quad \forall e \in E \\ &x_M \geq 0 \quad \forall matching\ M. \end{aligned}$$

8.9.4 Ellipsoid Algorithm

The ellipsoid algorithm is an iterative method for solving feasible lienar optimisation problems whereby it finds an optimal solution within a finite number of steps. It does this by the fact that the ellipsoid serves as a test of feasibility of a solution.

Definition 8.84 *(Unit ball). The n-dimensional unit ball is defined as*

$$B^n = \{x \in \mathbb{R}^n : x^T x \leq 1\}.$$

Definition 30: Affine Transformation

Let $f : \mathbb{R}^n \to \mathbb{R}^n$ where

$$f(x) = Mx + s$$

where $M \in \mathbb{R}^{n\times n}$ matrix, $s \in \mathbb{R}^n$, and $x \in \mathbb{R}^n$.

Definition 31: Ellipsoid

An n-dimensional ellipsoid E is the image of the n-dimensional unit ball under an affine transformation $f(x) = Mx + s$; that is

$$E = \{Mx + s : x \in B^n\}.$$

Algorithm 1: Ellipsoid Algorithm

```
Input: $\mathcal{P}$ is a polyhedra, $\epsilon$ size, and radius R
Output: Point in $\mathcal{P}$ or $\emptyset$
E = $B^n(R)$
point s = 0
while $s \notin \mathcal{P}$ do
    if iteration count $> n(2n+2)ln\frac{R}{\epsilon}$ then
        return $\mathcal{P}$ is empty
    else
        $\langle a, x\rangle \le b \leftarrow$ constraint of $\mathcal{P}$ that s violates
        $H \leftarrow \{x : \langle a, x\rangle \le \langle a, s\rangle\}$
        E $\leftarrow$ ellipsoid E‘ from assumption 3 w.r.t. E and H
        s $\leftarrow$ center of new ellipsoid E
return $s$
```

Lemma 8.85 *We make the following assumptions for the ellipsoid algorithm.*

1. *We can fit the polyhedron P inside a ball of* ***radius R****;*
2. *Either $P = \emptyset$ or it contains a ball of radius ϵ;*
3. *Given an ellipsoid E centered at s and a half-space $H = \{x \in \mathbb{R}^n : \langle a, x\rangle \ge \langle a, s\rangle\}$, then we can find another ellipsoid E' where $E' \supset E \cap H$ and*

$$vol(E') < \frac{vol(E)}{e^{\frac{1}{2n+2}}}.$$

Theorem 8.86 *The ellipsoid algorithm either returns a point inside the polyhedron P or correctly identifies that $P = \emptyset$.*

Theorem 22: Convergence of ellipsoid algorithm

After k $= n(2n+2)ln\frac{R}{\epsilon}$ iterations, we must have $P = \emptyset$.

8.9.5 Edge Colouring

Definition 32: Vizing's Theorem

Every graph can be coloured by at most $\Delta G + 1$ colours.

COMP3530: Discrete Optimisation

9. Lagrangian Relaxation

9.10 Lagrangian Relaxation

9.10.1 An introduction to Lagrangian Relaxation

The main premise behind lagrangian relaxation involves relaxing certain constraints and moving them into the objective value. Here, we penalise the objective value if a constraint is not being met.

Suppose we had a minimisation problem involving an integer program. We now want to run a branch and bound algorithm for our integer program. To get a good lower bound, we can use the notion of lagrangians.

Let z be the value of the following **integer program**

$$\min\langle c, x\rangle$$

$$\text{subject to } \begin{cases} Ax \geq b \\ Dx \geq d \\ x \in \mathbb{Z}^n. \end{cases}$$

Now suppose we apply lagrangian relaxation to the first set of constraints. We obtain the new **integer program**.

$$\min\langle c, x\rangle + \lambda(b - Ax)$$

$$\text{subject to } \begin{cases} Dx \geq d \\ x \in \mathbb{Z}^n. \end{cases}$$

where $A \in \mathbb{R}^{m \times n}$ and therefore $\lambda \in \mathbb{R}^m_+$. We let $t(\lambda)$ be the value of this program for a given choice of λ. Now, let us denote the polyhedron

$$\ell = \Big\{x \in \mathbb{Z}^n : Dx \geq d\Big\}.$$

We can now express $t(\lambda)$ as

$$t(\lambda) = \min_{x \in \ell} \Big[c \cdot x + \lambda \cdot (b - Ax)\Big].$$

Theorem 23: Lagrangian as a lower bound

Let z be the optimal cost of the integer program. Additionally, for a fixed λ, we denote $t(\lambda)$ as the optimal cost of the lagrangian of the IP for a given λ. Then, we have that for any $\lambda \geq 0$

$$t(\lambda) \leq z.$$

Theorem 24: Polynomial evaluation

For any $\lambda \geq 0$, we can evaluate $t(\lambda)$ in polynomial time.

Theorem 25

The lagrangian function $t : \mathbb{R}_+ \to \mathbb{R}$ is piece-wise linear, continuous, and concave.

Theorem 9.87 *The value $t = \max_{\lambda \in \mathbb{R}^n_+} t(\lambda)$ is given by the following program.*

$$min\langle c, x\rangle$$

$$subject\ to \begin{cases} Ax \geq b \\ x \in CH(\ell) \end{cases}$$

*where CH(ℓ) is the **convex hull** of the polyhedron ℓ.*

Corollary 9.88 *If the polyhedron ℓ is integral, then t is the value of the original linear relaxation.*

9.10.2 Unconstrainted Optimisation

We are interested in maximising the concave function $f : \mathbb{R} \to \mathbb{R}$.

Definition 9.89 *(Steepest ascent). The steepest ascent algorithm is described as follows. Start with an arbitrary solution $x^{(0)}$. We then repeat the solution according to the following rule*

$$x^{(k+1)} = x^{(k)} + \alpha^{(k)} f'(x^{(k)})$$

where $\alpha^{(k)} \geq 0$ is a parameter that determines the update step size.

Theorem 26: Guarantee of convergence

Let $f : \mathbb{R} \to \mathbb{R}$ be a continuous, differentiable, and concave function. The following criteria guarantees the convergence of $x^{(k)}$ to a maximizer of f as k tends to infinity.

$$\begin{cases} \lim_{k\to\infty} \alpha^{(k)} = 0 \quad \text{and } \lim_{k\to\infty} \sum_{i=1}^{k} \alpha^{(i)} = \infty \\ \alpha^{(k)} = \rho\beta^{(k)} \quad \text{for large } \rho \text{ and } \beta \approx 1. \end{cases}$$

We can now generalise the steepest ascent algorithm.

Theorem 27: Generalised steepest ascent

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a continuous, differentiable, and concave function we wish to maximise. Let $x^{(0)}$ be an arbitrary solution. Then, the steepest ascent update rule is

$$x^{(k+1)} = x^{(k)} + \alpha^{(k)} \nabla f(x^{(k)})$$

where $\alpha^{(k)} \geq 0$ is a parameter that determines the update step size and $\nabla f(x) = \left(\frac{\partial f(x)}{\partial x_1}, ..., \frac{\partial f(x)}{\partial x_n}\right)$ is the gradient of f at x.

Definition 9.90 *(Sub-gradient). A vector $s \in \mathbb{R}^n$ is a sub-gradient of f at x if*

$$f(y) \leq f(x) + \langle s, x - y \rangle$$

for all $y \in \mathbb{R}^n$.

Lemma 9.91 *If $f : \mathbb{R}^n \to \mathbb{R}$ is not differentiable, we can still update the solution in the direction of a sub-gradient to guarantee convergence.*

COMP3530: Discrete Optimisation

10. Maximum Submodular Coverage

10.11 Maximum Submodular Coverage

10.11.1 Maximum Coverage

Definition 33: p-approximation algorithm

For a maximisation problem, a p-approximation is an algorithm that runs in polynomial time and returns a solution whose cost is at least p times the cost of an optimal solution, where $p \leq 1$.

Definition 34: Maximum Coverage Problem

Let U be a ground set. Let $\mathcal{S} = \{S_1, ..., S_n\}$ be a collection of subsets of U. We want to pick k subsets, denoted by C, whose union has maximum cardinality

$$\max |\bigcup_{T \in C} T|$$
$$s.t. \quad C \subseteq \mathcal{S}$$
$$|C| = k.$$

Remark 10.92 *This problem is known to be NP-hard.*

We now specify the Greedy-algorithm for this problem.

Algorithm 2: GREEDY Algorithm for Maximum Coverage

```
Input: k, U, $\mathcal{S}$
Output: A collection of sets $C \subseteq \mathcal{S}$
C ← ∅
while $|C| < k$ do
    $S \in \mathcal{S}$ ← subset maximising $|S \setminus \cup_{T \in C} T|$
    $C \leftarrow C \cup S$
return $C$
```

Theorem 28: Approximation of GREEDY algorithm

The GREEDY algorithm is a $(1 - \frac{1}{e})$-approximation for the maximum coverage problem.

10.11.2 Maximum Submodular Coverage

Definition 35: Monotone function

A function $f : 2^U \to \mathbb{R}$ is monotone non-decreasing if

$$f(A) \leq f(B)$$

for all $A \subseteq B \subseteq U$.

Definition 36: Submodular function

A function $f : 2^U \to \mathbb{R}$ is submodular if

$$f(A \cup \{x\}) - f(A) \geq f(B \cup \{x\}) - f(B)$$

for all $A \subseteq B \subseteq U$ and $x \in U \backslash$B.

Definition 37: Normalized function

A function $f : 2^U \to \mathbb{R}$ is normalized if

$$f(\emptyset) = 0.$$

Definition 38: Maximum submodular coverage

We define a subset function $f : 2^U \to \mathbb{R}$ which is defined on the power set of a ground set U. We assume that f is a monotone non-decreasing, submodular, and normalized function. We are interested in finding a subset C of cardinality k maximising f.

$$\max f(C)$$

$$s.t. \quad C \subseteq U$$

$$|C| \leq k.$$

We now define the greedy algorithm to solving this problem.

Algorithm 3: GREEDY ALGORITHM FOR SUBMODULAR COVERAGE

```
Input: k, U, $\mathcal{S}$
Output: A collection of sets $C \subseteq \mathcal{S}$
C ← $\emptyset$
while $|C| < k$ do
    $x \in \mathcal{S} \leftarrow$ element maximising $f(C \cup x) - f(C)$
    $C \leftarrow C \cup x$
return $C$
```

Theorem 29: Approximation of GREEDY-Submodular algorithm

The GREEDY-Submodular algorithm is a $(1 - \frac{1}{e})$-approximation for the maximum coverage problem provided we can find an element x efficiently.

COMP3530: Discrete Optimisation

11. Minimum Weight Submodular Cover

11.12 Minimum Weight Submodular Cover

11.12.1 Minimum Set Cover

We first define the ILP of set cover.

Definition 39: Minimum Set Cover

Let U be our ground set and $S_1, ..., S_n$ be our sets. Furthermore, we assign the weight w_i to every set S_i. We let the variable x_i be 1 if we select set S_i and $x_i = 0$ otherwise.

$$min \sum_{i=1}^{n} w_i x_i$$

$$s.t. \begin{cases} \sum_{i:v\in S_i} x_i \geq 1 \quad \forall v \in U \\ x_i \leq 1 \quad \forall i \in \{1, ..., n\} \\ x_i \in \mathbb{N} \quad \forall i \in \{1, ..., n\}. \end{cases}$$

The linear program relaxation is when we set the last case to be $x_i \geq 0 \quad i \in \{1, ..., n\}$.

Definition 40: Minimum Set Cover

Let U be a ground set and $\mathcal{S} = \{S_1, ..., S_m\}$ be a collection of subsets. Define $w : \mathcal{S} \rightarrow \mathbb{R}_+$ to be a weight function. The goal is to pick a minimum collection of subsets $C \subseteq \mathcal{S}$ whose union equals U

$$\min_{C} \quad W(C)$$

$$s.t. \bigcup_{T\in C} T = U.$$

First, we analyse the case where $W(S_i) = 1$ for all $S_i \in S$. Hence, we seek to maximise the cardinality of the collection C.

Algorithm 4: GREEDY ALGORITHM FOR CARDINALITY

```
Input: k, U, $\mathcal{S}$
Output: A collection of sets $C \subseteq \mathcal{S}$
C ← ∅
while $|C| < |U|$ do
    $S \in \mathcal{S} \leftarrow$ subset maximising $|S \backslash \cup_{T\in C} T|$
    $C \leftarrow C \cup S$
return $C$
```

Lemma 11.93 *Let X_i be the set of elements covered by GREEDY-CARDINALITY after i iterations. Then*

$$|X_i| \geq |U|.\Big(1 - \Big(1 - \frac{1}{OPT}\Big)^i\Big).$$

Theorem 11.94 *The GREEDY-CARDINALITY algorithm is a $ln|U|$-approximation algorithm for the set cover problem.*

11.12.2 Minimum Weight Set Cover

We now extend upon the algorithm of the last version by now taking the set weights into account when building the solution. We now look at the cost pay per newly covered element.

Algorithm 5: WEIGHTED GREEDY ALGORITHM

```
Input: w, U, $\mathcal{S}$
Output: A collection of sets $C \subseteq \mathcal{S}$
C $\leftarrow \emptyset$
while $|C| < |U|$ do
    $S \in \mathcal{S} \leftarrow$ subset maximising $\frac{w(S)}{|S - \cup_{T \in C} T|}$
    $C \leftarrow C \cup S$
return $C$
```

Theorem 30: Weighted Greedy Algorithm Approximation

The WEIGHTED-GREEDY algorithm is a H_Δ-approximation algorithm for the weighted set cover problem, where $\Delta = \max_{S \in \mathcal{S}} |S|$.

Recall that H_n is the nth harmonic number.

11.12.3 Minimum Weight Submodular Cover

First, we let U be a ground set. We define the function $f : 2^U \to \mathbb{R}$ and weight $w : U \to \mathbb{R}_+$. The goal is to pick $C \subseteq U$ such that $f(C) = f(U)$ and minimising $w(C) = \sum_{j \in C} w(j)$.

Algorithm 6: GREEDY SUBMODULAR ALGORITHM

```
Input: w, U, $\mathcal{S}$
Output: A collection of sets $C \subseteq \mathcal{S}$
C $\leftarrow \emptyset$
while $f(C) < f(U)$ do
    $j \in U$ be an element minimising $\frac{w_j}{f(C+j) - f(C)}$
    $C \leftarrow C \cup j$
return $C$
```

We now analyse the performance of this algorithm, which **depends on the function f.**

Theorem 31: Integer submodular approximation

GREEDY-SUBMODULAR algorithm is a $H_\Delta-$approximation where

$$\Delta = \max_{j \in U} \Big(f(\{j\}) - f(\emptyset) \Big)$$

when f is

1. integer-valued;
2. submodular;
3. monotone non-decreasing.

Theorem 32: Non-integer valued submodular approximation

GREEDY-SUBMODULAR is a $(1 + ln\Delta)$-approximation where

$$\Delta = \max_{j \in U} \frac{p_j}{\min S : p_j(S) > 0}$$

and $p_j(S) = f(S + j) - f(S)$.

11.12.4 Minimum Spanning Tree

Let $G = (V, E)$ be a connected graph with edge weights $w : E \to \mathbb{R}_+$. We denote C(H) the collection of connected components of some graph H. For any subset of edges $S \subseteq E$, we define

$$f(S) = \sum_{A \in C(V,S)} \Big(|A| - 1 \Big).$$

Note that $f(E) = n-1$. Furthermore, for any subset $S \subseteq E$ such that $f(S) = n-1$, this spans the vertex set.

Lemma 11.95 *The function f defined above is submodular and monotone non decreasing.*

The GREEDY-SUBMOULDAR algorithm is actually a generalisation of Kruskals' algorithm!

Theorem 33: GREEDY-SUBMODULAR for MST

The GREEDY-SUBMODULAR algorithm is a 1-approximation for the minimum weight spanning subgraph problem.

COMP3530: Discrete Optimisation

12. Linear Programming Rounding

12.13 Linear Programming Rounding

12.13.1 Graph Partitioning Problems

We look at 3 problems.

Definition 12.96 *(ST-cut problem). Given $s, t \in V$, we want to find $F \subseteq E$ such that s and t lies in distinct components of $G \backslash F$.*

Definition 12.97 *(Multiway-cut). Suppose we are given k terminals $t_1, ..., t_k$. We want $t_1, ..., t_k$ to be in distinct components of $G \backslash F$.*

Remark 12.98 *The multiway-cut is a generalisation of the ST-cut where $k = 1$.*

Definition 12.99 *(Multiway). Suppose we are given terminal pairs $(s_1, t_1), ..., (s_k, t_k)$. We want a partition π such that (s_i, t_i) are in distinct components for each i.*

Remark 12.100 *The multiway problem is a generalisation of the multiway cut problem if we let the pairs be every possible pair of combinations*

12.13.2 LP relaxation for ST-cut

We now describe the setup for the LP relaxation of the ST-cut. We define the variables to be $x_e \forall e \in E$. We wish to minimise $\sum_e c_e x_e$ such that

$$\sum_{e \in P} x_e \geq 1 \quad \forall P \in P_{(s,t)}$$

$$x_e \in \{0, 1\}$$

where $P_{(s,t)}$ denotes the set of all ST-paths. This constraint says that for all ST paths, we need to remove at least one edge from the path in order to separate s and t.

Theorem 34: Separation oracle for ST-cut

The inequality

$$\sum_{e \in P} x_e \geq 1$$

holds if and only if the shortest ST-path satisfies this.

Remark 12.101 *As a result, we only need to look at the shortest path between ST to see whether if this constraint is satisfied.*

From this observation, we can recast the original problem.

Theorem 35: ST-cut metric formulation

We denote the variables as $d_{u,v}$ for all $u, v \in V$ where $d_{u,v}$ is the distance of the shortest path under x_e from u to v. Hence, our objective function is

$$\min \sum_e c_e d_e$$

$$s.t. d_{u,v} \in [0, 1]$$

$$d_{s,t} \geq 1$$

$$d_{u,v} \leq d_{u,w} + d_{w,v} \quad \forall u, v, w \in V.$$

This new linear program formulation embeds the vertices into a metric space.

Lemma 12.102 *The optimal value of the original linear program is the same as the optimal value of the modified metric linear program.*

Definition 12.103 *(Cut metric). The variable $d_{u,v}$ is known as the **cut metric**. This satisfies the properties of a metric.*

Lemma 12.104 *Vertices being in the same component is a equivalence relation.*

12.13.3 Randomised Rounding for Metric Linear Programming

We now specify our randomised algorithm.

Algorithm 7: Randomised Rounding for ST-cut LP

```
Input: Graph G=(V,E) and cut metric $d_{u,v}$
Output: A ST-cut $F \subseteq E$ of the graph
r ← radius from uniform (0,1) distribution
F ← edges cut by balls $B(s, r)$ where s is the source
return $F$
```

Lemma 12.105 *Let F be the cut from the randomised rounding algorithm. Then*

$$\mathbb{E}\Big\{c(F)\Big\} \leq \sum_e c_e d_e$$

where $\mathbb{E}$ is the expectation operator.

Corollary 12.106 *The ST-cut metric formulation is integral.*

12.13.4 Linear Programming for minimum multiway cut

We now define the linear program for the multiway cut.

Theorem 36: Multiway cut LP formulation

We define the LP formulation for the minimum multiway cut to be

$$\min \sum_e c_e d_e$$

$$s.t. d_{t_i,t_j} \geq 1 \quad \forall 1 \leq i < j \leq k$$

$$d_{u,v} \leq d_{u,w} + d_{w,v} \quad \forall u,v,w \in V$$

$$d_{u,v} \in [0,1].$$

We now define the randomised algorithm for the multiway cut.

Algorithm 8: RANDOMISED ROUNDING FOR MULTIWAY CUT LP

```
Input: Graph G=(V,E) and cut metric $d_{u,v}$
Output: A multiway-cut $F \subseteq E$ of the graph
r ← radius from uniform $[0, \frac{1}{2})$ distribution
$F_i$ ← edges cut by balls $B(t_i, r)$ where $t_i$ is the source
F ← $F_1 \cup ... \cup F_k$.
return $F$
```

Lemma 12.107 *Choosing $r \in [0, \frac{1}{2}]$ ensures that each ball is disjoint.*

Lemma 12.108 *Let F be the cut from the randomised rounding multiway cut algorithm. Then*

$$\mathbb{E}\Big\{c(F)\Big\} \leq 2\sum_e c_e d_e$$

where $\mathbb{E}$ is the expectation operator.

The randomisation algorithm is a 2-approximation algorithm.

12.13.5 Linear Programming for minimum multi-cut

We now define the linear program for the minimum multi-cut.

Theorem 37: Multi-cut LP formulation

We define the LP formulation for the minimum multi cut to be

$$\min \sum_e c_e d_e$$

$$s.t. d_{s_i,s_i} \geq 1 \quad \forall i$$

$$d_{u,v} \leq d_{u,w} + d_{w,v} \quad \forall u,v,w \in V$$

$$d_{u,v} \in [0,1].$$

One issue now is that we cannot guarantee the balls we pick our algorithm will be disjoint.

We now define the randomised algorithm for the multi cut.

Algorithm 9: RANDOMISED ROUNDING FOR MULTI CUT LP

```
Input: Graph G=(V,E) and cut metric $d_{u,v}$
Output: A multi-cut $F \subseteq E$ of the graph
r ← radius from uniform $[0, \frac{1}{2})$ distribution
[K] ← random permutation
$F_i$ ← edges cut by balls $B(s_i, r) \cap V$ where $s_i$ is the source
F ← $F_1 \cup ... \cup F_k$.
return $F$
```

Lemma 12.109 *Let F be the cut from the randomised rounding multi cut algorithm. Then*

$$\mathbb{E}\Big\{c(F)\Big\} \leq \mathcal{O}(logk) \sum_e c_e d_e$$

where $\mathbb{E}$ is the expectation operator.

The randomisation algorithm is a log k approximation algorithm.