Lecture Notes for MAT4004

Suggested Citation: (2018), "Lecture Notes for MAT4004", : Vol. xx, No. xx, pp 1–18.
DOI: 10.1561/XXXXXXXXX.

the essence of knowledge

Boston — Delft

Contents

1 Definition **2**

2 Connectivity **6**

2.1 Bipartite Graphs 6

2.2 Bounds on the number of edges 7

2.3 Edge & Vertex Connectivity 8

3 Hamiltonian Graphs **12**

3.1 Necessary Conditions 12

3.2 Sufficient Conditions 13

4 Trees **19**

4.1 Characterisation for Trees 19

4.2 Spanning Trees 20

4.3 Prufer Code 21

4.4 Minimum Spanning Tree 23

5 Shortest Path **25**

5.1 Dijkstra's Algorithm 25

5.2 All-Pairs Shortest Paths 26

6 Maximum Flow **29**
6.1 Problem Setting 29
6.2 Linear Programming Formulation 30
6.3 Flow Augmentation 31
6.4 Flow Decomposition 33
6.5 Dinic's Algorithm 34
6.6 Maximum Flow & Minimum Cut 34

7 Bipartite Matching **39**
7.1 Max Cardinality Bipartite Matching 39
7.2 Weighted Bipartite Matching 42

8 TSP **47**
8.1 Heuristics Method for Finding TSP Tour 47
8.2 TSP Formulation as an Integer Programming 50

9 Graph Coloring **52**
9.1 Facts about Colourability 53
9.2 Colouring Edges 54

10 Planar **56**
10.1 Euler's Polyhedral Formula 57
10.2 Non-planar Graphs 58
10.3 Geometric Dual of a Planar Graph 59
10.4 Crossing Number 60

Lecture Notes for MAT4004

ABSTRACT

This document is the typed lecture notes for MAT4004

(2018), “Lecture Notes for MAT4004”, : Vol. xx, No. xx, pp 1–18. DOI: 10.1561/XXXXXXXXXX.

1

Definition

Definition 1.1 (Graph). A graph $G = (V, E)$ consists of a non-empty finite set V of elements called *vertices*; and a finite family of *unordered* pairs of vertices called *edges*.

1. The edge $e = \{v, w\}$ is said to join the vertices v and w
2. The vertex v is *adjacent* to the vertex w if there exists an edge $\{v, w\}$
3. The edge e is also said to be *incident* to v.

Definition 1.2 (Simple). Multiple (parallel) edges are the edges joining the same pair of vertices; Loops are the edges of the form $\{v, v\}$; A *loopless* graph with *no* multiple edges is called a *simple* graph.

Definition 1.3 (Isomorphism). Two graphs G_1 and G_2 are *isomorphic* if there is a one-to-one correspondence between the vertices of G_1 and the vertices of G_2 such that the number of edges joining any two vertices of G_1 equals the number of edges joining the corresponding pair of vertices in G_2.

Definition 1.4 (Adjacent & Incident). Two vertices are *adjacent* if there is an edge joining them; the edges are *incident* to the edge; two edges are *adjacent* if they share a common vertex.

Definition 1.5. The *degree* of a vertex v, say $\deg(v)$, is the number of edges incident to v.

In particular, a loop contributes 2 to the $\deg(v)$; a vertex with degree 0 is an *isolated* vertex; a vertex of degree 1 is an *end-vertex.*

Definition 1.6 (Degree sequence). The *degree sequence* of a graph is the sequence of degrees of its vertices, written in *non-decreasing* order.

Theorem 1.1. In any graph, the sum of all the vertex-degrees is an *even* number.

Corollary 1.2. In any graph, the number of vertices with *odd* degree is *even.*

Definition 1.7. A graph H is a *subgraph* of a graph $G = (V, E)$, if each of its vertices belong to $V(G)$, and each of its edges belongs to $E(G)$.

Definition 1.8. The subgraphs can be obtained by the following operations:

1. $G - e$: removing edge e
2. $G - F$: removing the set of edges F
3. $G - v$: removing vertex v and all its incident edges
4. $G - S$: removing the vertices in the set S and all edges incident to any vertex in S
5. $G \setminus e$: *contracting* edge e. (question: identifying endpoints of e and delete the possible parallel edges)

Definition 1.9 (Complement). If G is a simple graph, its complement, denoted as $\bar{G}$, has the same vertex set, while two vertices are adjacent in $\bar{G}$ if and only if thery are not adjacent in G.

Definition 1.10 (Null & Complete). A *null* graph is the one where the edge set is empty; a simple graph where each distinct pair of vertices are adjacent is a *complete* graph; a complete graph on n vertices is denoted as K_n.

Definition 1.11 (Walk & Path & Cycle).

1. A walk consists of a sequence of edges, one following after another
2. A walk in which no vertex appears *more than once* is called a path
3. A walk in which no vertex appears *more than once*, except for begining and end vertices which coincide, is called a *cycle.*

Definition 1.12 (Bipartite). If the vertex set of a graph consists of the union of two *disjoint* sets A and B such that each edge of G joints a vertex in A and a vertex in B, then G is a *bipartite* graph. Moreover, if each vertex in A is joined to each vertex in B by an edge, it is a *complete bipartite graph*, denoted as $K_{|A|,|B|}$.

Definition 1.13 (Directed Graph). The graph $G = (N, A)$ is a *directed graph* (or *digraph*) if V is a finite set of nodes and A is a finite family of *directed edges* (*arcs*). Each arc in A, denoted as (v, w), is an ordered pair of nodes.

The diagraph D is simple if the arcs are all distinct, and there are no loops.

Definition 1.14 (Underlying Graph). The underlying graph $G = (N, E)$ of a directed graph $G = (N, A)$ has the same node set; every edge $e = \{v, w\}$ of E corresponds to an arc $(v, w) \in A$.

Question: does underlying graph mean the *undirected* graph?

Definition 1.15 (Out & In-degree). Recall that the edge (v, w) is *incident* from v and *incident* to w. The *out-degree* of vertex v is the number of edges *incident from* v; the *in-degree* of vertex v is the number of edges *incident to* v.

Theorem 1.3. *In any digraph, the sum of all the in-degrees is equal to the sum of all the out-degrees.*

Proof. Question Each edge (v, w) contribute 1 to the in-degree and 1 to the out-degree. $\square$

Definition 1.16 (Adjacency matrix)**.** The *adjacency matrix* for an *undirected graph* $G = (V, E)$ is a $|V| \times |V|$ square matrix where the element (i, j) is the number of edges joining the vertices i and j.

Definition 1.17 (Incidence matrix)**.** The *incidence matrix* for an *undirected graph* $G = (V, E)$ is a $|V| \times |E|$ matrix, where the element (i, j) is 1 if the vertex i is incident to edge j.

Definition 1.18 (Node-Arc Incidence matrix)**.** The *node-arc incidence matrix* for a directed graph $G = (V, E)$ is a $|V| \times |E|$ matrix where the element (i, j) is 1 if edge j is incident from vertex i, and is -1 if edge j is incident to i.

2

Connectivity

2.1 Bipartite Graphs

Definition 2.1 (Cycle). A *cycle* is a sequence of adjacent edges

$$(v_0, v_1), \quad (v_1, v_2), \quad \ldots, \quad (v_{m-2}, v_{m-1}), \quad (v_{m-1}, v_m = v_0)$$

where all the vertices $v_0, v_1, \ldots, v_{m-1}$ are *distinct*. The number of deges in the cycle is called its *length*.

Definition 2.2 (Bipartite). A graph $G = (V, E)$ is *bipartite* if its vertex-set is a union of two disjoint sets, say A and B, such that every edge in E joins a vertex in A to one in B.

Theorem 2.1. A graph G is *bipartite* if and only if every cycle of G has even length.

Proof. Necessity. Take any cycle and "trace" its edges, the cylce must have even length.

Sufficiency. w.l.o.g., G is connected. Construct A as the vertices s.t. the shortest path to v has even length. Then B is the set of vertices that are not in A.

We claim that (A, B) forms a bipartite graph, since otherwise we can construct a cycle with odd length. □

2.2 Bounds on the number of edges

Definition 2.3 (Connected). A graph is *connected* if there is a path from every vertex to any other vertex

Definition 2.4 (Simple). A graph is *simple* if there are no parallel edges nor loops.

The goal is to give a bound on a simple connected graph with n vertices.

Definition 2.5 (Connected).

1. If $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are graphs, and the vertex sets V_1 and V_2 are disjoint, then their *union* is the graph $G = (V_1 \cup V_2, E_1 \cup E_2)$
2. A graph is *connected* if it cannot be expressed as a union of graphs, and disconnected otherwise.

Definition 2.6 (Component). Any *disconnected* graph G can be expressed as a union of connected graphs, each of which is called a *component* of G.

Theorem 2.2. Let G be a simple graph on n vertices with k componets. The number m of edges of G satisfies

$$n - k \leq m \leq (n-k)(n-k+1)/2.$$

Lower Bound (By induction on the number of edges): The lower bound is true for a null graph G. We assume the lower bound holds true for number of edges less than m_0. Consider a graph G (suppose that it has n vertices and k components) with *minimal* edges, say m_0, i.e., removing one edge will increase the number of components by 1. Therefore, the resultant graph has n vertices, $k+1$ components and $m_0 - 1$ edges. By the induction hypothesis, we imply

$$n - (k+1) \leq m_0 - 1 \implies n - k \leq m_0.$$

Upper bound: It suffices to consider the case where each component of G is a complete graph (since we are considering the upper bound). Suppose that there are two components C_i and C_j with n_i and n_j vertices, respectively, and $n_i \geq n_j > 1$.

Now we argue that n_i should be larger than n_j as much as possible ($n_i + n_j$ keeps the same) to obtain more vertices: Consider the components B_i and B_j with $n_i + 1$ and $n_j - 1$ vertices, respectively. Then the toal number of vertices remains unchanged, but the total number of edges is more than the previous one:

$$\frac{n_i(n_i+1)}{2}+\frac{(n_j-2)(n_j-1)}{2}-\frac{(n_i-1)n_i}{2}-\frac{n_j(n_j-1)}{2}=n_i-n_j+1>0.$$

Therefore, in order to obtain the maximum number of edges, G must consists of a complete graph with $n-k+1$ vertices and $k-1$ isolated vertices.

Corollary 2.3. If a simple graph on n vertices has more than $(n-1)(n-2)/2$ edges, it must be connected.

Proof. The key insight is that the graph with more componets should have less vertices. The vertices for the graph with more than 1 components is upper bounded by $(n-1)(n-2)/2$, by applying Theorem (2.2). □

2.3 Edge & Vertex Connectivity

Definition 2.7 (Disconneting Set). A *disconnecting (edge) set* of a graph G is a set of edges whose removal increases the number of components of G

Definition 2.8 (Cutset). A *cutset* of a graph G is a *minimal disconnecting set*; a cut-set of size 1 is called a *bridge*

Definition 2.9 (Edge-Connectivity). For connected graph G, its *edge-connectivity* $\lambda(G)$ is the number of edges of the smallest cutset of G. In other words, $\lambda(G)$ is the fewest number of edges to be removed such that the resulting graph is dis-connected.

We say G is *k-edge connected* if $\lambda(G) \geq k$

Definition 2.10 (Separating Vertex Set). A *separating vertex set* of a connected graph G is a set of vertices whose deletion (together with their incident edges) disconnects G; we say v is a *cut-vertex* if a separating set has only one vertex v.

Definition 2.11 (Vertex-Connectivity)**.** For connected graph G, its *vertex-connectivity* $\kappa(G)$ is the minimum size S such that $G-S$ is disconnected or has only one vertex.

We say G is k *(-vertex) connected* if $\kappa(G) \geq k$.

For example, $\kappa(K_n) = n-1$, where K_n denotes a complete graph with n vertices.

The Theorem (2.4) shows that the vertex-connectivity is more intrinsical than the edge-connectivity.

Theorem 2.4. For a connected graph G, we have

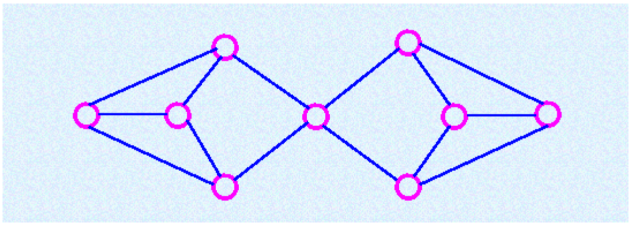
$$\kappa(G) \leq \lambda(G) \leq \delta(G),$$

where $\delta(G)$ denotes the smallest vertex-degree of G.

Proof. Upper Bound: The upper bound is trivial, since removing all the edges of the vertex with smallest degree will lead to a disconnected graph, i.e., $\lambda(G) \leq \delta(G)$.

Lower Bound: Skipped, since not understand. □

Example 2.1. It's possible for both inequality in Theorem (2.4) to be strict:



In such case, $\kappa(G) = 1, \lambda(G) = 2, \delta(G) = 3$.

Theorem 2.5 (Menger's Theorem)**.** A graph G is *k-edge-connected* if and only if any two distinct vertices of G are joined by at least k disjoint paths, i.e., no two of which have any edge in common.

Theorem 2.6 (Menger's Theorem)**.** A graph with at least $k+1$ vertices is k-connected if and only if any two distinct vertices of G are joined by at least k disjoint paths.

Definition 2.12 (Disconnecting Set)**.** A vw-disconnecting set of a graph G is a set of edges F of G such that every path from v to w includes an edge of F.

Theorem 2.7 (Menger's Theorem (edge form))**.** The *maximum number of edge-disjoint paths* connecting two distinct vertices v and w of a connected graph is equal to the *minimum number of edges in a vw-disconnecting set.*

Proof. It's trivial that the the maximum number of edge-disjoint paths cannot be more than the number of edges in a vw-disconnecting set.

The reverse direction relies on the induction on the number of edges. □

From Eulerian Problem into the Chinese Postman Problem Recall that a connected graph is *Eulerian* if there exists a closed trail that includes every edge of G. The necessary and sufficient condition is that every vertex has even degree.

When a graph is not Euliean, we want to find a walk that covers all the edges, but avoid repeating edges as much as possible. Or equivalently, how many and which edges need to be duplicated such that the resultant is Eulerian? This motivates the Chinese Postman Problem.

Definition 2.13 (The Chinese Postman Problem)**.** A postman has to deliver letters to a given neighbourhood. He needs to walk through all the streets in the neighbourhood and back to the post office. How can he design his route so that he walks the shortest distance?

Definition 2.14 (Connected)**.** A directed graph G is connected if the corresponding underlying graph is connected

Definition 2.15 (Strongly Connected)**.** A directed graph G is *strongly connected* if for any vertex v and w, there is a directed path from v to w

Definition 2.16 (Orientable)**.** We say an *undirected* graph is *orientable* if each edge can be *directed* such that the resulting graph is *strongly connected.*

Theorem 2.8. A connected graph is *orientable* if and only if every edge of G lies in at least one cycle.

Sufficiency. Choose any cycle C and direct its edges *cyclically.* If all edges of G have been oriented, the proof is complete; otherwise choose an edge, say e, that is not in C but adjacent to an edge of C. By hypothesis, the edge e is in some cycle C', and orient the edges in C' cyclically except for those already directed.

We proceed this operation, and thus directing at least one more edge for each time. Therefore, we are done until all edges have been oriented. The oriented edges form a strongly-connected graph. $\square$

3

Hamiltonian Graphs

Motivation We are interested in the Eulerian graph, i.e., whether there exists a closed trail (that travels each edge once and only once), for a connected graph. There is a simple necessary and sufficient condition for a graph to be Eulerian: *every vertex has even degree*. Similarly, is there a cycle that includes every vertex once and only once, for a connected graph?

Definition 3.1 (Hamiltionian).

1. A *Hamiltonian cycle* in a graph is a *cycle* that includes every vertex (once and only once).
2. A graph is *Hamiltonian* if it contains a Hamiltonian cycle.
3. A non-Hamiltonian graph is *semi-Hamiltonian* if there exist a *path* through every vertex.

However, not all graphs are *Hamitonian*. We are studying necessary or sufficient conditions for a graph to be Hamiltonian.

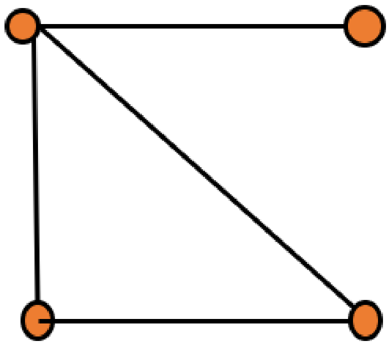
3.1 Necessary Conditions

Theorem 3.1. If G is Hamitonian, then for each non-empty set $S \subseteq V$, the graph $G - S$ has at most $|S|$ componenets.

As a result, *every Hamitonian graph must be* 2*-connected.*

Proof. Consider a Hamiltionian cycle in G, which must pass through all vertices including both $G - S$ and S. Each time when leaving a component of $G - S$, it must go next to a vertex in S, and must be a distinct vertex in S. Therefore, the number of components of $G - S$ cannot be more than the number of vertices in S. □

Remark 3.1. However, the converse may not be necessarily true. Consider the counter-example



3.2 Sufficient Conditions

Most sufficient conditions are of the form: *If the graph has enough edges, then it is Hamiltonian.*

Theorem 3.2 (Ore's theorem)**.** If G is a simple graph with $n \geq 3$ vertices, and if

$$\deg(v) + \deg(w) \geq n,$$

for each pair of non-adjacent vertices v and w, then G is Hamitonian.

Proof. Suppose on the contrary that G is not, and w.l.o.g., G is maximal. Thus there must be a path $v_1 \to v_2 \to \cdots \to v_n$ passing through each vertex but v_1 and v_n are non-adjacent. Therefore, $\deg(v_1)+\deg(v_n) \geq n$, i.e., there must be a vertex v_i adjacent to v_1 with the property that v_{i-1} is adjacent to v_n, which leads to a Hamitonian cycle:

$$v_1 \to \cdots \to v_{i-1} \to v_n \to v_{n-1} \to v_{n-2} \to \cdots \to v_i \to v_1$$

□

Corollary 3.3. If G is a simple graph with $n \geq 3$ vertices, and $\deg(v) \geq \frac{1}{2}n$ for each vertex v, then G is Hamiltionian.

Theorem 3.4. If G is a simple graph with $n \geq 3$ vertices, and if G has at least $(n-1)(n-2)/2+2$ edges, then G is Hamitionian.

Proof.

1. If every vertex is adjacent to every other vertex, then G is complete, and therefore Hamitonian
2. Otherwise, let v and w be two non-adjacent vertices, and $H = G - \{v, w\}$ contains $(n-2)$ vertices, with maximum number of edges $(n-2)(n-3)/2$. There exists at least $(n-1)(n-2)/2+2-(n-2)(n-3)/2 = n$ vertices incident to either v or w. Therefore, $\deg(v) + \deg(w) \geq n$. By applying Ore's theorem, G must be Hamitonian.

□

Theorem 3.5. Let $G = (V_1 \cup V_2)$ be a bipartite graph. If G is Hamitonian, then $|V_1| = |V_2|$.

If G is semi-Hamitonian, then $|V_1|$ and $|V_2|$ differ by at most 1.

Proof. Consider the Hamitonian cycle. □

Corollary 3.6. The converse of Theorem (3.5) also holds if G is a *complete* bipartite graph with at least 3 vertices.

Proof. Construct the Hamitonian cycle. □

Definition 3.2 (Hamiltonian Closure)**.** The *Hamiltonian closure* of a graph $G = (V, E)$ is the graph $C(G)$ obtained from G by iteratively adding edges joining pairs of *non-adjacent* vertices whose degree sum is at least $|V|$, until no such pair remains.

If $G = C(G)$, then we say G is closed.

The closure does not depend on the sequence by which the edges are added.

Lemma 3.7. The *Hamitonian closure* of a graph $G = (V, E)$ is well-defined.

Proof. Suppose we add sequence of edges $\{e_1, \ldots, e_f\}$ to form G_1 and $\{f_1, \ldots, f_s\}$ to form G_2.

Note that a pair of vertices (u, v) are added iff theya are non-adjacent and the sum of degree is at least $n = |V|$. Thus by definition, f_1 is addable to G, and must be added at some point in the first sequence, i.e., $f_1 \in G_1$. Inductively, $f_1, f_2, \ldots, f_s \in E(G_1)$, which implies $E(G_2) \subseteq E(G_1)$. Similarly, $E(G_1) \subseteq E(G_2)$. □

Theorem 3.8 (Bondy-Chvatal)**.** A simple graph G is Hamitionian if and only if its closure $C(G)$ is Hamitionian

Proof. Clearly, if G is Hamitonian, so is $C(G)$. Conversely, let $\{G_0 := G, G_1, \ldots, C(G)\}$ be a sequence of graphs where each G_i is formed by performing a single closure step to G_{i-1}. Let k be the minimal value such that G_k is Hamiltonian. Let (w, u) be the edge added to form G_k from G_{j-1} and let C be a Hamiltionian cycle in G_k. Therefore, C must contain the edge (w, u), and suppose it is of the form

$$u := v_1 \to \cdots \to v_n = w \to u.$$

Now we argue that G_{k-1} is Hamiltonian. Since there are n edges incident to either u or v, there exists a vertex v_i incident to u such that v_{i-1} is incident to w in G_{k-1}, which leads to a Hamitonian cycle in G_{k-1}:

$$u = v_1 \to v_i \to v_{i+1} \to \cdots \to v_n = w \to v_{i-1} \to v_{i-2} \to \cdots \to v_2 \to v_1 = u$$

□

Theorem 3.9 (Chvatal)**.** Let G be a simple graph on $n \geq 3$ vertices with degrees $d_1 \leq d_2 \leq \cdots \leq d_n$. If for all $i < n/2$, either $d_i > i$ or $d_{n-i} \geq n - i$, then G is Hamiltonian.

Proof. It suffices to consider the case where G is closed. Suppose on the contrary that $G = C(G)$ is non-Hamitonian, which is not complete.

Among all pairs of non-adjacent vertices, let (u, v) be the pair with maximum degree sum, w.l.o.g., $\deg(u) \le \deg(v)$. Since G is closed, we have $\deg(u) + \deg(v) < n$, i.e., $\deg(u) < n/2$. Suppose that $\deg(u) = k$, then every other vertex in V not adjacent to v must have degree no larger than $\deg(u) = k$; there are $n - 1 - \deg(v)$ such vertices. Note that $\deg(u) + \deg(v) \le n - 1$ implies that $n - 1 - \deg(v) \ge \deg(u) = k$. Therefore, we have found k vertices of degree at most k.

Similarly, every vertex that is not adjacent to u has degree at most $\deg(v)$. By assumption, $\deg(v) < n - k$. There are $(n - 1) - \deg(u)$ such vertices. Also, u it self has the degree no larger than $\deg(v)$, thus there are $n - k$ vertices with degree smaller than $n - k$. □

Consider for some sequence $(d_1, \dots, d_n)$ we have $d_h \le h$ and $d_{n-h} \le n - h - 1$ for some $0 < h < n/2$. Now we want to construct a graph with this degree sequence that is non-Hamitonian.

Consider one possible example:

$$(\underbrace{h, \dots, h,}_{h \text{ times}} \underbrace{n-h-1, \dots, n-h-1,}_{n-2h \text{ times}} \underbrace{n-1, \dots, n-1,}_{h \text{ times}})$$

The graph has two components: a complete bipartite graph K_{hh} with $V_1 = \{v_1, \dots, v_h\}$ and $V_2 = \{v_{n-h+1}, \dots, v_n\}$ and a complete graph K_{n-h} with $V = \{v_{h+1}, \dots, v_n\}$.

Definition 3.3 (Hamiltonian for Directed Graph)**.** Consider a *directed* graph D. We say D is *Hamiltonian* if there is a directed cycle that includes every vertex of D.

A non-Hamiltonian digraph that contains a directed path through every vertex is semi-Hamiltonian.

We are considering the necessary and sufficient conditions for a digraph to be Hamiltonian; and are there some special types of graphs more likely to be Hamiltonian?

Definition 3.4 (Tournament)**.** A directed graph D is a *tournament* if there is exactly one arc between every pair of vertices. In other words, a tournament is an orientation of a complete simple graph.

Theorem 3.10 (Redei)**.** Every non-Hamiltonian tournament is semi-Hamiltonian, i.e., every tournament contains a Hamiltonian path

Proof. We show this result by induction on $|V| = n$. Suppose the tournament $T - v$ contains a Hamitonian path, say $v_1 \to \cdots \to v_{n-1}$, and show that T has a Hamitonian path. Consider three cases:

1. There is an arc $(v, v_1) \in T$
2. There is an arc $(v_{n-1}, v) \in T$
3. There is no arc for (v, v_1) and (v_{n-1}, v)

In previous two cases, the result is trivial. In the third case, define i to be the smallest index such that (v, v_i) is an arc in T. Therefore, we construct a Hamitonian path:

$$v_1 \to v_2 \to \cdots \to v_{i-1} \to v \to v_i \to v_{i+1} \to \cdots \to v_{n-1}$$

□

Theorem 3.11 (Camion)**.** Every strongly connected tournament is Hamitonian.

Proof. For a strongly connected tournament T, it must have at least 3 vertices. We will show that a tournament with $|V| = n$ contains cycles of length $3, 4, \ldots, n$ by induction.

1. Base Case: Consider any vertex v in T. Since T is strongly connected, there must be at least one vertex incident to v and at least one vertex incident from v. Let V_{in} be the set of vertices incident to v, V_{out} the set of vertices incident from v. Since T is strongly connected, there must be an arc $(v_o, v_i) \in V_{\text{in}} \times V_{\text{out}}$, and therefore a 3-cycle exists.

2. It remains to show that if there is a cycle C of length k, then there is also a cycle of length $k + 1$. Suppose there is a vertex c not in the cycle such that there exists arcs incident to c from some vertices in C, and arcs incident from c to some vertices in C. Thus there must be an arc (u, v) in C such that T contains arcs (u, c) and (c, v). Thus we have a $(k + 1)$-cycle.

3. If no such vertex exists, then for each vertex w not in the cycle, its arcs that are joined to vertices in the cycle are either all incident from w, or all incident to w. Let A and B be the two (disjoint) sets of vertices with such properties. Since T is strongly connected, neither A nor B can be empty. There must exist an arc from some vertex $b \in B$ to some vertex $a \in A$. Therefore, we can replace two consecutive arcs in the k-cycle, say (u, c) and (c, v) by the arcs $(u, b), (b, a), (a, v)$ to get a $(k + 1)$-cycle.

□

Theorem 3.12. Let D be a strongly connected digraph with n vertices. If $\text{outdeg}(v) \geq n/2$ and $\text{indeg}(v) \geq n/2$ for every vertex v, then D is Hamitonian.

The Travelling Salesman Problem Given the distance between each pair of cities (on a given list), we want to find a Hamiltonian cycle (that visits all the cities on the list and return to his starting point), with mimimum distance. How many Hamiltonian cycles in a complete graph? $\frac{(n-1)!}{2}$!

4

Trees

4.1 Characterisation for Trees

Definition 4.1 (Tree)**.** An undirected graph is *acyclic* if it has no cycles. A connected acyclic undirected graph is a *tree*.

- A vertex of degree 1 in a tree is called a *leaf*
- An acyclic (undirected) graph but *not connected* is called a *forest*

Remark 4.1. Trees are simple graphs. They are the simplest non-trivial graphs, with some nice properties.

Theorem 4.1. Let T be an undirected graph with n vertices. TFAE:

1. T is a tree
2. T contains no cycles and has $n-1$ edges
3. T is connected and has $n-1$ edges
4. T is connected and each edge of T is a bridge (i.e., a cutset of a *single* edge)
5. Any two vertices of T is connected by a unique path

6. T is acylic, but the addition of any new edge creates *exactly* one cycle

Proof. (1) implies (2): The proof is by induction on n. The removal of any edge of T disconnects the graph into two components, each of which is a tree. Therefore, applying the induction hyphothesis on each components gives the desired result. (2) implies (3): Suppose T is disconnected. Since T is acyclic, then each component is a tree. Therefore, the number of edges in each component tree is one fewer than the number of vertices, which implies the graph T has fewer than $n-1$ edges, which is a contradiction.

(3) implies (4): Suppose the removal of edge e of T does not disconnect the graph. Since $T-e$ is a tree, and the number of edges is $n-1$, we derive a contradiction.

(4) implies (5): Since T is connected, there is at least one pathbetween any pair of vertices. If a pair of vertices is connected by two distinct paths, then they must contain a cycle. The removal of an edge on this cycle does not disconnect the graph. Thus the edge is not a bridge.

(5) implies (6): T does not contain a cycle. Consider an edge e that is not in $V(T)$, and there is a path in T connecting the two end-vertices of e. Therefore, the addition of e creates a cycle. If there are two cycles in $T+e$, then both cycles must contain e, which implies there exists a cycle in $T+e$ not containing e, which is a contradiction.

(6) implies (1): Suppose that T is disconnected, then it's possible to add an edge to T that joins two different components but not create a cycle. □

4.2 Spanning Trees

Consider a simple connected graph G on n vertices. We are interested in the subgraphs of G that are trees.

Definition 4.2 (Spanning Tree). A subgraph T of $G=(V,E)$ that is a tree and contains all the vertices V of G is a *spanning tree* of G.

Theorem 4.2. If $T=(V,F)$ is a spanning tree of a connected graph $G=(V,E)$, then

1. Each cutset of G contains an edge in T
2. Each cycle of G contains an edge in $\bar{T} := (V, E - F)$.

Proof. 1. Suppose K is a cutset of G that disconnects G into two components G_1 and G_2. Since T is a spanning tree, it must contain an edge joining one vertex in G_1 and one vertex in G_2

2. If C is a cycle in G with no edge in common with $E - F$, then all the edges in C are in Fm which contradicts that T is acyclic.

□

Theorem 4.3. If T and T' are spanning trees of a connected graph G and edge $e \in E(T) - E(T')$, then there is an edge $e' \in E(T') - E(T)$ such that $T - e + E'$ is a spanning tree of G.

Proof. By part (4) in Theorem (4.1), each edge of T is a cutset of T. Let k and k' be two components of $T - e$. Since T is a spanning tree of G, it is connected and there exists an edge $e' \in T'$ that joins a vertex in K to a vertex in K'. Therefore, the constructed $T - e + e'$ is connected and has $n - 1$ edges, therefore a spanning tree of G. □

4.3 Prufer Code

Motivation How many different (i.e. non-isomorphic) graphs are there with a given property? In particular, how many regular graph (i.e., each vertex has the same degree) of degree 2? How many regular graph of degree n? Unfortunately, it is almost impossible to express these results by simple closed-form formulas.

One application is in the enumeration of chemical molecules, i.e., how to count the saturated hydrocarbons with the formula C_nH_{2n+2}? Note that it is a tree since it is connected with $n + (2n + 2)$ vertices and $(4n + (2n + 2))/2 = 3n + 1$ edges.

Therefore, the problem turns into the enumeration of labelled trees with a given number of vertices.

In particular, we assume the graphs with label $1 - 2 - 3 - 4$ and $4 - 3 - 2 - 1$ are the same since they have the same adjacent matrix, but they are both different to $1 - 2 - 4 - 3$.

Theorem 4.4. There are $2^{n(n-1)/2}$ distinct labelled simple graphs with n vertices; and n^{n-2} distinct labelled trees with n vertices.

4.3.1 Prufer's Code

Now we aim to represent each tree with n vertices as a sequence of $n-2$ numbers selected from $\{1, 2, 3, \ldots, n\}$

Algorithm 1 Prufer Coding for a tree T

```
Input:
    A tree $T$ with $n$ labelled vertices
Output:
    The sequence $(p_1, \ldots, p_{n-2}) \subseteq \{1, 2, 3, \ldots, n\}$
1: Initialize $T_1 = T$ and $i = 1$;
2: Stop if $T_i$ has only one edge;
3: Otherwise, let $v_i$ be the lowest labelled leaf of $T_i$, and $p_i$ be its
   neighbour. Let $T_{i+1} = T_i - v_i$;
4: Repeat previous step;
```

Given a sequence of $n-2$ numbers selected from $\{1, \ldots, n\}$, we can decode it into the corresponding labelled tree. Consider the nontrivial case $n \geq 3$:

Algorithm 2 Constructing a Tree from a (Prufer) Sequence

```
Input:
    The sequence $(p_1, \ldots, p_{n-2}) \subseteq \{1, 2, 3, \ldots, n\}$ A tree $T$ with $n$
    labelled vertices
Output:
    A tree $T$ with $n$ labelled vertices
1: Initialize $V_0 = \{1, \ldots, n\}$ and $i = 1$;
2: Let $P_i = (p_i, p_{i+1}, \ldots, p_{n-2})$, and $v_i$ be the lowest labelled vertex
   that do not appear in $P_i$ Join it into vertex $p_i$, and let $V_i = V_{i-1} \backslash \{v_i\}$;
3: If $i = n-2$, then join the two remaining vertices in $V_i$. STOP;
4: Otherwise, increment $i$ by 1 and repeat from step 2;
```

4.4 Minimum Spanning Tree

Given a network with costs labelled for each edges, we are interested in finding a spanning tree with minimum cost. The computation cost is high for finding each of the n^{n-2} trees.

Kruskal's Algorithm Let $G = (V, E)$ be a connected graph with n vertices with weights on the edges. Then the following algorithm gives a spanning tree $H = (V, F)$ of G with a minimum edge weight:

Algorithm 3 Kruskal's Algorithm

```
1: Initialize $E(H) = \emptyset$ and let the edges in $E$ be sorted in non-decreasing order;
2: Let $e$ be an edge in $E$ of smallest weight. Add this edge to $H$ as long as it does not create a cycle in $H$;
3: Repeat above step until obtaining $n - 1$ edges in $H$.
```

Theorem 4.5 (Optimality for finding MST). Let T be a spanning tree of G. Let $e = \{v_1, v_2\}$ be an edge of T. The removal of e disconnects T into two components, say T_1 and T_2 with vertex sets V_1 and V_2 respectively. Let $K(G)$ be an edge cutset of G that disconnects G into two components with vertex sets V_1 and V_2.

Then T is a MST of G if and only if

$$\forall e \in \{v_1, v_2\} \in E(T), w(e) \leq w(f), \quad \forall f \in K(G)$$

Jarnik-Prim's Algorithm We denote $[W, V \setminus W]$ as the edge cutset that disconnects G into two components with vertex-sets W and $V \setminus W$. In other words, $[W, V \setminus W]$ is the set of edges in G that joins a vertex in $W \subseteq V$ to a vertex not in W.

Algorithm 4 Jarnik-Prim's Algorithm

```
1: Initialize $V_1 = \{v_1\}$ (any vertex), $T_1 = (V_1, E_1 = \emptyset)$, $k = 1$, $d(v_1) = 0, d(v) = \infty, \forall v \neq v_1$, $n(v) = v_1, \forall v \neq v_1$;
2: **Stop** if $V_k = V$;
3: Otherwise for $v \neq v_k$, let $d(v) = \min\{d(v), w(\{v, v_k\})\}$, and $n(v) = v_k$ if $d(v) = w(\{v, v_k\})$
4: Let $v_{k+1} = \arg\min_{v \neq V_k} d(v)$ (essentially looking for the edge of least weight in cutset $[V_k, V \setminus V_k]$), $V_{k+1} = V_k \cup \{v_{k+1}\}, T_{k+1} = (V_{k+1}, E_k \cup \{(v_{k+1}, n(v_{k+1}))\})$.
   Increment $k$ by 1 and go to step 2
```

5

Shortest Path

Problem setting Now we turn the attention to directed graphsm where there is a cost associated with each arc:

- How to find a shortest path from a given (origin) node to a given (destination) node?
- How to find a shortest path from a given (origin) node to each other node?

5.1 Dijkstra's Algorithm

Remark 5.1. The Dijkstra's algorithm does not work for negative arc lengths.

Efficiency of Dijkstra's Algorithm

- Initialization: $O(n)$
- Iteration: repeat for n times
  - Termination: $O(1)$

Algorithm 5 Dijkstra's Algorithm

```
Input:
    The graph $G = (N, A)$ with cost function $c$ on arcs
Output:
    Shortest path from origin $s$ to destination $t$
1: Initialize OPEN list $S := \emptyset$, $T = N$; $d(i) = M$ for each $i \in N$;
   $d(s) = 0$, pred$(s) = 0$;
2: While $|S| < n$, do
       • Let $i = \arg\min\{d(k) \mid k \in T\}$; $S := S \cup \{i\}$; $T := T \setminus \{i\}$;
           – for each $(i, j) \in A$,
             if $d(j) > d(i) + c(i, j)$, then $d(j) = d(i) + c(i, j)$;
             pred$(j) = i$;
   End While
```

  - Finding minimum: $O(n)$ comparisons. Indeed, the data structure technique can help to reduce into $O(m \log n)$ operations using binary heaps, and $O(m + n \log n)$ operations using Fiboniacci heaps.

- Updating: $O(m)$ in total

Thus Dijkstra's algorithm takes $O(n) + n(O(1) + O(n)) + O(m) = O(n^2)$ steps.

5.2 All-Pairs Shortest Paths

Given a directed graph $G = (N, A)$ with arc lengths c. Here we allow negative arc length but assume G has no cycles of negative length.

Theorem 5.1 (Optimality Condition). For every pair of nodes (i, j), let $d[i, j]$ denote the length of a directed path from i to j. Then $d[i, j]$ is the shortest path from i to j for all $i, j \in N$ if and only if

1. $d[i, j] \leq d[i, k] + d[k, j]$ for $\forall i, j, k \in N$

2. $d[i,j] \leq c(i,j)$ for all $(i,j) \in A$

An obvious label correcting type algorithm is developed based on the optimality condition. We start with some labels $d[i,j]$, and update the labels until the optimality conditions are satisfied.

Algorithm 6 Dijkstra's Algorithm

```
1: Initialize
       $d[i,j] = \infty, \forall [i,j] \in N \times N$
       $d[i,i] = 0, \forall i \in N$
       $d[i,j] = c(i,j),$ for each $(i,j) \in A$
2: While there exists 3 nodes $i, j, k$ such that $d[i,j] > d[i,k] + d[k,j]$,
       do $d[i,j] := d[i,k] + d[k,j]$
   End While
```

Algorithm 7 Floyd-Warshall Algorithm

```
1: Initialize
       $d[i,j] = \infty, \forall [i,j] \in N \times N$
       $pred[i,j] = 0, \forall [i,j] \in N \times N$
       $d[i,i] = 0, \ \forall i \in N; d[i,j] = c(i,j), \ \forall (i,j) \in A$
2: For each $k = 1 : n$, for each $[i,j] \in N \times N$, if $d[i,j] > d[i,k] + d[k,j]$,
   then
       $d[i,j] = d[i,k] + d[k,j]$ and $pred[i,j] = pred[k,j]$
```

Remark 5.2. By adding one more test when updating the labels in the Floyd-Warshall algorithm, we can detect negative cycles if they exist in the graph:

- If $i = j$, check if $d[i, i] < 0$
- If $i \neq j$, check if $d[i, j] < -nC$, where C is the largest arc length.

If either two cases are true, then the graph contains a negative cycle.

Faster Implementation of Dijkstra's Algorithm Consider large and sparse graphs, i.e., the number of edges m is much smaller than n^2. We can apply the modified Dijkstra's algorithm n times, which is faster than Floyd-Warshall algorithm. With a d-heap implementation, which is a technique in Data structure, the Dijkstra's algorithm takes

$$O(m \log_d n + nd \log_d n) \text{ operations}$$

Taking $d = \max\{2, \lceil m/n \rceil\}$, we conclude that Dijkstra's algorithm take $O(m \log_2 n)$ steps in this case.

6

Maximum Flow

6.1 Problem Setting

In this chapter we consider directed graphs and we assign a flow to each arc. Specifically, given

1. A directed graph $G = (N, A)$
2. A *source node* $s \in N$ and a *sink node* $t \in N$
3. A function $u : A \to \mathbb{R}^+$, which denotes the *arc capacity*

Definition 6.1 (Feasible Flow)**.** We consider assigning a flow x_{ij} on each arc $(i, j) \in A$ such that

1. $0 \leq x_{ij} \leq u_{ij}$
2. The flow in equals the flow out at each (non-source and non-sink) node.

We aim to find the maximum amount that can flow from s to t.

6.2 Linear Programming Formulation

The maximum flow problem can be formulated using linear programming:

$$\begin{array}{ll} \max & v \\ \text{such that} & 0 \le x_{ij} \le u_{ij}, \quad \forall (i,j) \in A \\ & \sum_{(i,j)\in A} x_{ij} - \sum_{(j,k)\in A} x_{jk} = \begin{cases} -v, & \text{if } j = s \\ 0, & \text{if } j \in N \setminus \{s,t\} \\ v, & \text{if } j = t \end{cases} \end{array} \tag{6.1}$$

Here v is called the *value* of a given flow x from s to t.

6.2.1 Upper bound on the flow value

Definition 6.2. $[S,T]$ is called an $s-t$ cut of G if $s \in S, t \in T, S \sqcup T = N$.

Theorem 6.1. The flow value is upper bounded by the capacity of any cutset $[S,T]$.

Proof. Summing up the second constraint of (6.1), we imply

$$\begin{aligned} &\sum_{j\in S}\left[\sum_{(i,j)\in A} x_{ij} - \sum_{(j,k)\in A} x_{jk}\right] = -v \\ \Longleftrightarrow &\sum_{j\in S}\sum_{(i,j)\in A, i\in T} x_{ij} + \sum_{j\in S}\sum_{(i,j)\in A, i\in S} x_{ij} \\ &\quad - \sum_{j\in S}\sum_{(j,k)\in A, k\in S} x_{jk} - \sum_{j\in S}\sum_{(j,k)\in A, k\in T} x_{jk} = -v \\ \Longleftrightarrow &\sum_{j\in S}\sum_{(i,j)\in A, i\in T} x_{ij} - \sum_{j\in S}\sum_{(j,k)\in A, k\in T} x_{jk} = -v \end{aligned}$$

Therefore,

$$v \le \sum_{j\in S}\sum_{(j,k)\in A, k\in T} x_{jk} \le \sum_{j\in S}\sum_{(j,k)\in A, k\in T} u_{jk},$$

which is the capacity of the cutset $[S,T]$. □

Corollary 6.2. If for the given feasible flow x, the flow value v_x equals $U[S',T']$ for some cutset $[S',T']$, then v_x must be the maximum flow value.

6.3 Flow Augmentation

Definition 6.3 (Residual Graph)**.** Given a directed graph G and a feasible $s-t$ flow x, we define the corresponding *residual graph* $G_x = (N, A_x)$ with *residual capacities* $r : A \to \mathbb{R}^+$ on the arcs as follows:

1. The arc $(i, j) \in A_x$ if
   - either $(i, j) \in A$ with $x_{ij} < u_{ij}$
   - or $(j, i) \in A$ and $x_{ji} > 0$
2. The residual capacity of $(i, j) \in A_x$ is $r_{ij} = (u_{ij} - x_{ij}) + x_{ji}$

In other words, the residual graph is such that the residual capacities assigned into each arc is positive, i.e., $G_x = G - x$.

Definition 6.4 (residual capacity of the path)**.** Let P_x be a directed path from s to t in augmenting network G_x. Define the *residual capacity of the path* P_x as

$$r(P_x) := \min_{(i,j)\in P_x} r_{ij}$$

If $r(P_x) > 0$, we say that there is an augmenting path from s to t.

Clearly, if $r(P_x) > 0$, we can increase the value of the (network) flow by augmenting the flow x with some path in P_x.

Theorem 6.3 (Optimality condition on maximum flow)**.** An $s-t$ flow x is of maximum value iff there is no $s-t$ augmenting path w.r.t. x.

Proof. For the forward direction, if there is an augmenting path, then clearly flow can be increased.

For the reverse direction, suppose there is no augmenting path w.r.t. x. Let S be the set of nodes to which there is an augmenting path from s, and therefore $t \notin S$. Let $T = N \setminus S$, which implies

$$\sum_{i\in S}\sum_{j\in T,(i,j)\in A} = \sum_{i\in S}\sum_{j\in T,(i,j)\in A} u_{ij}, \quad \sum_{i\in T}\sum_{j\in S,(i,j)\in A} x_{ij} = 0$$

Therefore, $v_x = U[S, T]$, and v_x must be the maximum flow value. □

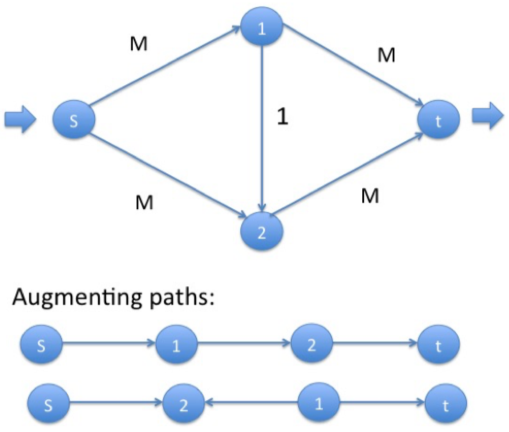
Algorithm 8 Ford-Fulkerson Max-Flow Algorithm

```
1: If no $s$-$t$ augmenting path exists, then current flow is maximum
2: Otherwise, augment flow (up to residual capacity of augmenting
   path); repeat from step (1)
```

Corollary 6.4. If all arc capacities are integer-valued, then the maximum flow value is also integer-valued, so are the arc flows.

Proof. Start with flow value $v_x = 0$. Every flow augmentation increases the flow values by an integer value. □

Efficiency of the Max-Flow Algorithm Consider the following max-flow problem:



Each time, we augment by one unit of flow. So it take $2M$ augmentations, which is not efficient.

Therefore, we need the flow decomposition technique.

6.4 Flow Decomposition

Given an assignment of flows x_{ij} to each $(i,j) \in A$, the excess flow at any $j \in A$ is the quantity

$$e(j) = \sum_{(i,j)\in A} x_{ij} - \sum_{(j,k)\in A} x_{jk}$$

which denotes the difference between the flow getting into j and the flow getting out of j. If x is a feasible flow, then the excess flow $e(j)$ is always zero, except $j = s, t$.

If $e(j) > 0$, then we say j is an *excess* node; if $e(j) < 0$, then j is a *deficit* node.

Algorithm 9 Flow Decomposition Algorithm

```
1: Start with a node $i_0$ with $e(i_0) < 0$ or where there is an arc $(i_0, i_1)$ with positive flow. Let $k = 1$
2: If $i_k$ has $e(i_k) > 0$, then we identified a path $P = \{i_0, \ldots, i_k\}$. Set $f(P) = \min\{-e(i_0), e(i_k), \min\{x_{ij} \mid (i,j) \in P\}\}$. Repeat from step 1.
3: If $i_k = i_h$ for some $0 \leq h < k$, then we identified a cycle $W = \{(i_h, i_{h+1}), \ldots, (i_{k-1}, i_k)\}$. Set the cycle flow $g(W) = \min\{x_{ij} \mid (i,j) \in W\}$. Repeat from step 1.
3: Otherwise there is an arc $(i_k, i_{k+1})$ with positive arc flow. Increase $k$ by 1. Repeat from Step 2.
```

Therefore, we decomposed arc-flows into path-flows and cycle-flows.

Theorem 6.5. If flow augmentation is done along an augmenting path of maximum residual capacity of maximum residual capacity, then no more than $P(m \log U)$ augmentations are required to obtain the maximum flow.

Theorem 6.6. If flow augmentation is done along an augmenting path with a minimum number of arcs, then no more than $O(mn/2)$ augmentations are required to obtain the maximum flow

6.5 Dinic's Algorithm

Definition 6.5 (Level Graph). Given an augmenting network G_x, define the *level* of a node i, $\ell(i)$, as the number of arcs in a *shortest* path from s to i.

We define the *level graph* $\mathcal{L}_x$ as the subgraph of G_x such that

- it contains nodes reachable from s
- it contains arcs (i, j) such taht $\ell(j) = \ell(i) + 1$

Algorithm 10 Dinic's Algorithm

```
1: Initially $x = 0$.
2: Construct level graph $\mathcal{L}_x$.
3: If $\mathcal{L}_x$ does not contain an $s$-$t$ augment path, stop.
3: Otherwise, repeat:
       • find an $s$-$t$ augmenting path $P_x$ in $\mathcal{L}_x$ by depth-first search
       • augment flow by $r(P_x)$
   Unitl all $s$-$t$ paths in $\mathcal{L}_x$ have zero residual capacity
4: Go to step 2.
```

Using Dinic's algorithm, the max-flow can be found in $O(n^2m)$ time

6.6 Maximum Flow & Minimum Cut

Theorem 6.7. For any network, the maximum flow value from source s to sink t is equal to the minimum cut capacity of an s-t cut.

Definition 6.6 (Disconnecting Set). A st-disconnecting set of a graph G is the set of edges F of G such that every path from s to t includes an edge of F

Theorem 6.8 (Menger's Theorem (edge form)). The maximum number of edge-disjoint paths connecting two distinct vertices s and t of a connected graph is equal to the minimum number of edges in an st-disconnecting set.

Theorem 6.9 (Menger's Theorem (vertex form))**.** The maximum number of vertex-disjoint paths connecting two distinct non-adjacent vertices s and t of a connected graph is equal to the minimum number of vertices in an st-separating set.

Theorem 6.10. A graph G is k-edge-connected if and only if any two distinct vertices of G are joined by at least k edge-disjoint paths.

Theorem 6.11. A graph G with at least $k+1$ vertices is k-connected if and only if any two distinct vertices of G are joined by at least k vertex-disjoint paths.

6.6.1 System of distict representatives

Consider a scheduling problem:

$$S_1 = \{3, 5, 6\}, \quad S_2 = \{3, 4, 6\}, \quad S_3 = \{5, 8\}, \ldots$$

where S_i is the possible time slots for exam i. We want to find a schedule where all the exams are scheduled and no two exams are scheduled at the same time slot.

Definition 6.7 (SDR)**.** In general, we consider

- a set of elements $E = \{e_1, \ldots, e_m\}$
- a collection $\mathcal{S}$ of subsets of E: $\mathcal{S} = \{S_1, \ldots, S_n\}$

We say there is a *system of distinct representatives (SDR)* of $\mathcal{S}$ if there are distinct elements $\{e_{r(1)}, \ldots, e_{r(n)}\}$ such that $e_{r(i)} \in S_i$ for $i = 1, \ldots, n$.

Actually, we can find a SDR by solving a max-flow problem: We model a graph with vertex s, a sink vertex t, vertices E, and vertices $\mathcal{S}$, and arcs:

- (s, e_i) with capacity 1 for $j = 1, \ldots, m$
- (S_j, t) with capacity 1 for $j = 1, \ldots, n$
- (e_i, S_j) with infinite capacity if $e_i \in S_j$.

If the max s-t flow of this network has value n, then a SDR exists.

Theorem 6.12 (Hall, 1935)**.** The colletion $\mathcal{S}$ has a SDR if and only if for all $k = 1, \dots, n$, the union of any k sets of $\mathcal{S}$ contain at least k distinct elements.

Applications of SDR

Theorem 6.13. For a zero-one matrix, the minimum number of *lines* that cover all the 1's in the matrix is equal to the maximum number of 1's that can be selected such that no two are on the same line.

A line can be either a column or a row.

Proof. Let ℓ be the minimum of lines needed, and L be the maximum number of 1's selected. It's clear that $\ell \geq L$. It suffices to show the equality.

Suppose that $\ell = r + c$, i.e., all the 1's of the matrix are covered by r rows and c columns (why?). w.l.o.g.,the rows and columns needed are the top r rows and left-most c columns.

Consider a SDR problem with $\mathcal{S} = \{S_1, \dots, S_r\}$, where $S_i = \{j \mid j > c, a_{ij} = 1\}$.

- Every union of k sets of $\mathcal{S}$ contain at least k distinct elements: Suppose on the contrary that $|S_1 \cup S_2 \cup \dots \cup S_k| \leq k - 1$, then all the 1's in the first k rows are in $k - 1$ columns, then we can cover all the 1's in the matrix using $r - k$ rows and $c + (k - 1)$ columns, which contradicts to the minimality of ℓ.

Therefore, $\mathcal{S}$ has a SDR. There exists r 1's in the first r rows and to the right of the first c columns, and no two of which are in the same column.

Similarly, there are c 1's in the first c columns and below the top r rows, and no two of which are in the same row.

Therefore, $L \geq r + c = \ell$. □

Definition 6.8 (Latin Square)**.** A n by n matrix is a *Latin square* if the entries are drawn from a set of n elements such that each element appears exactly once in each row and each column.

Definition 6.9 (Latin Rectangle). An r by n matrix is an r by n Latin Rectangle if each row contain the numbers $\{1, \ldots, n\}$ (in some permutation) and each column does not contain any repeated number.

Theorem 6.14. Given a Latin rectangle with $r < n$, we can add a row to form an $(r+1)$ by n rectangle.

Proof. Let $S_j, j = 1, \ldots, n$ be the set of numbers that don't appear in the column j. The SDR for this collection of n sets will be the elements in the row $(r+1)$.

We claim that every k sets in $\mathcal{S} = \{S_1, \ldots, S_n\}$ must contain at least k distinct numbers. Note that the k sets contain $k(n-r)$ numbers counting repeats.

Note that each number can appear in the k sets at most $(n-r)$ times. Therefore, there must be k distinct numbers in the k sets.

Therefore, by Hall's theorem, there is a SDR and row $(r+1)$ can be added to the Latin Rectangle. □

6.6.2 Max-flow Min-cut and Linear Programming Duality

Consider the circulation model of the max-flow problem, i.e., we introduce the arc (t, s) with infinite capacity. In this case, we have

$$\begin{array}{ll} \max & x_{ts} \\ \text{such that} & 0 \le x_{ij} \le u_{ij},\ \forall (i,j) \in A \\ & \sum_{(i,j)\in A} x_{ij} - \sum_{(j,k)\in A} x_{jk} = 0,\ \forall j \in \end{array}$$

The dual problem is given by:

$$\begin{array}{ll} \min & \sum_{(i,j)\in A} u_{ij} z_{ij} \\ \text{such that} & y_j - y_i + z_{ij} \ge 0,\ \forall (i,j) \in A \\ & y_s - y_t \ge 1,\ \text{for } (t,s) \\ & z_{ij} \ge 0,\ \forall (i,j) \in A \\ & y_j \text{unrestricted in sign}, \forall j \in N \end{array}$$

Consider a feasible solution defined by an s-t cut $[S, T]$:

$$y_j = \begin{cases} 1, & j \in S \\ 0, & j \in T \end{cases} \qquad z_{ij} = \begin{cases} 1, & \text{if } (i,j) \in A, i \in S, j \in T \\ 0, & \text{otherwise} \end{cases}$$

The objective value is

$$\sum_{(i,j)\in A, i\in S, j\in T} u_{ij} = U[S,T],$$

which is the capacity of the cut.

We can also show the max-flow min-cut theorem using linear programming duality.

7

Bipartite Matching

Problem Setting Consider an undirected loopless graph $G = (V, E)$

Definition 7.1 (Matching)**.** A subset $M \subseteq E$ is a matching in G, if no two edges in M are adjacent (i.e., incident to the same vertex)

Definition 7.2 (Saturation)**.** A matching M saturates a vertex v if there is some edge in M incident to v

Otherwise, v is called unsaturated, exposed, or unmatched

A matching M in G is *perfect* if it saturates every vertex of G.

7.1 Max Cardinality Bipartite Matching

Given a *bipartite* graph, we want to find the matching with the most number of edges.

Definition 7.3 (M-alternating)**.** An M-alternating path in G for the matching M is a path whose edges are alternately in M and $E \setminus M$.

An M-augmenting path in G for the matching M is an M-alternating path whose first and last vertices are *unsaturated.*

Theorem 7.1 (Berge)**.** A matching M in G is a maximum cardinality matching iff G contains no M-augmenting path.

Proof. Necessity: If G contains the M-augmenting path $\{e_1, \dots, e_k\}$, then M is not of maximum cardinality, i.e., $M' = (M \setminus \{e_2, e_4, \dots\}) \cup \{e_1, e_3, \dots\}$ is a matching with one more edge.

Sufficiency: Suppose M is not a max cardinality matching. Then we will show there exists an M-augmenting path. Let M^* be a maximum cardinality matching. Construct $E' = M \Delta M^*$, i.e., the symmetric difference of M and M^*, i.e., the edges are either in M or M^*, but not both.. Let $H = (V', E')$ be the subgraph of G spanned by E'

Each vertex of H is of degree either 1 or 2. Therefore, each component of H is either a cycle with edges alternately in M and M^*, or a path with edges alternately in M and M^*. Sicne M^* contains more edges than M, there must be an alternating path that starts and ends with edges in M^*, i.e., an M-augmenting path.

□

1. Let M be a matching for the bi-partite graph $G = (X \cup Y, E)$. Set all vertices as unlabelled. (M can be the empty matching)

2. (a) For all unsaturated vertices, label it as $\emptyset$
   (b) If there is no-unscanned vertex, *STOP*.
   (c) Scan labelled vertex i as follows:
      i. If $i \in X$: For all unlabelled vertex j with $(i, j) \in E$, label j with label i. Go to step (2b)
      ii. If $i \in Y$, if i is unmatched, go to step 3. Otherwise, label the j with label i such that $(i, j) \in M$. Go to step (2b)

3. Find an augmenting path by back-tracking from i. For all edges in the M-augmenting path that are in M, remove from M, for all edges in the M-augmenting path that are not in M, add to M. Go to step 1.

Example 7.1. To be added

Computational Complexity Find an augmenting path: $\mathcal{O}(m)$. Number of possible augmentations: $\mathcal{O}(\min(|X|, |Y|))$

Characertizaiton of maximum cardinality bipartite matching

Definition 7.4. For $S \subseteq V$, the neighbour set of S in G is the set of all vertices adjacent to vertices in S, denoted as $\mathcal{N}(S)$.

Theorem 7.2. Let $G = (X \cup Y, E)$ be a bipartite graph. Then G contains a matching that saturates every vertex in X iff

$$|\mathcal{N}(S)| \geq |S|, \quad \forall S \subseteq X$$

Note: to be added.

Corollary 7.3. Let $G = (X \cup Y, E)$ be a regular bipartite graph with $|X| \leq |Y|$. Then G contains a complete matching, i.e., one that saturates every vertex in X.

Proof. Consider any $S \subseteq X$. Since G is a regular graph, suppose that every vertex has degree k, then there are $k|S|$ edges incident to S. Let $T = \mathcal{N}(S)$, then $S \subseteq \mathcal{N}(T)$. The number od edges connecting T to $\mathcal{N}(T)$ is $kT = k\mathcal{N}(S)$. Only some subsets of there edges are conneted to S, i.e.,

$$k|S| \leq k\mathcal{N}(S)$$

□

Definition 7.5 (vertex cover)**.** A *vertex cover* of $G = (V, E)$ is a subset of vertices $K \subseteq V$ such that every edge in E is incident to some vertex in K

We say that a vertex cover K is minimum if G has no vertex cover of smaller size.

Theorem 7.4. Let $G = (X \cup Y, E)$ be a bi-partite graph. Then the size of a maximum cardinality matching of G equals to the size of a minimum vertex cover of G

Proof. Let M^* be the maximum cardinality matching and K^* be a minimum vertex cover. It's clear that we need $|M^*|$ vertices to cover the edges of the matching M^*, i.e., $M^* \leq |K^*|$.

Let U be the set of vertices in X that are un-saturated w.r.t M^*; Z be the set of vertices v in Y, such that there is an M^*-alternating path to v from some vertex in U; W be the set of vertices in X such that there is an M^*-alternating path from some vertex in U.

Since M^* is of maximum cardinality matching, vertices in U are the only vertices in W that remains unsaturated. Also, every vertex in Z is saturated, since otherwise M^*-augmenting path exists. ALso, every neighbour of a vertex in W must be in Z, since otherwise M^*-augmenting path exists. Therefore, $\mathcal{N}(W) = Z$.

Constrcut $\tilde{K} = (X \setminus W) \cup Z$. Every edge of G must be incident to a vertex in $\tilde{K}$, since otherwise there is an edge incident to a vertex in W and a vertex in $Y \setminus Z$, contradicting to $\mathcal{N}(W) = Z$.

Therefore, $\tilde{K}$ is a vertex cover, such that $|\tilde{K}| = |M^*|$. □

7.2 Weighted Bipartite Matching

Problem: given a bipartite graph $G = (X \cup Y, E)$ and edge weights $w : E \to \mathbb{R}$, find a matching of maximum total edge weight.

LP Review Consider the primal LP:

$$\begin{array}{ll} \text{max} & cx \\ \text{such that} & Ax \leq b \\ & x \geq 0 \end{array}$$

The dual LP is

$$\begin{array}{ll} \text{min} & yb \\ \text{with} & yA \geq c \\ & y \geq 0 \end{array}$$

Since both x and y are non-negative, it is easy to see that the weak duality result holds:

$$cx \leq yAx \leq yb$$

If we find (x, y) that is primal and dual feasible, and the complementarity condition holds:

$$(yA - c)x = y(Ax - b) = 0$$

then x is optimal for (P) and y is optimal for (D) and the optimal values of (P) and (D) are the same.

The complementary slackness conditions can be stated as:

- $x_j > 0$ implies $\sum_i y_i A_{ij} = c_j$ for $j = 1 : n$
- $y_i > 0$ implies $\sum_j A_{ij} x_j = b_j$ for $i = 1 : n$

The linear programming for max weight bipartite matching is given by:

$$(M) \qquad \begin{array}{ll} \min & \sum_{(i,j)\in E} w_{ij} x_{ij} \\ \text{with} & \sum_{j\in Y} x_{ij} \le 1,\ \forall i \in X \\ & \sum_{i\in X} x_{ij} \le 1,\ \forall j \in Y \\ & x_{ij} \ge 0,\ \forall (i,j) \in E \end{array}$$

The dual problem is

$$(DM) \qquad \begin{array}{ll} \min & \sum_{i\in X} u_i + \sum_{j\in Y} v_j \\ \text{with} & u_i + v_j \ge w_{ij},\ \forall (i,j) \in E \\ & u_i \ge 0,\ \forall i \in X \\ & v_j \ge 0,\ \forall j \in Y \end{array}$$

The complementarity conditions are:

- $x_{ij} > 0$ implies $u_i + v_j = w_{ij}$ for all $(i,j) \in E$
- $u_i > 0$ implies $\sum_j x_{ij} = 1$ for $i \in X$
- $v_j > 0$ implies $\sum_i x_{ij} = 1$ for $j \in Y$

The idea of the Hungarian algorithm is to construct a pair of primal and dual solutions to (M) and (DM) that are feasible and satisfy conditions (1) and (3) but not necessarily (2). Then iteratively improve the solutions until optimality attained.

Start with a null matching ($x_{ij} = 0$) and set dual variables $v_j = 0, u_i = W = \max\{w_{ij} \mid (i,j) \in E\}$.

We find an augmenting path to maintain feasibility and complementary slackness conditions (1) and (3). So, we use only edges where $w_{ij} = u_i + v_j$. If augmentation is possible, one more of condition (2) will be satisfied. Otherwise, change the value of the dual variables.

Consider an augmenting forest from all unmatched (exposed) vertices in X. Let F be the vertices in this augmenting forest. Suppose we increase v_j for $j \in F \cap Y$ and decrease u_i for $i \in F \cap X$. Consider the following cases:

- $i \in F, j \in F$: $(u_i - \delta) + (v_j + \delta) = w_{ij}$, no change
- $i \in F, j \notin F$: $(u_i - \delta) + (v_j) \geq w_{ij}$, LHS decrease
- $i \notin F, j \in F$: $u_i + (v_j + \delta) \geq w_{ij}$: LHS increase
- $i \notin F, j \notin F$: $u_i + v_j \geq w_{ij}$: no change.

For feasibility,we need $\delta = \min\{u_i + v_j - w_{ij} \mid i \in X, j \in Y\}$. How big can δ be? We want it to be big enough so that

- we can add one more edge to augmenting forest
- some u_i are driven to zero; one or more complementarity condition (2) is satisfied

Note that at every stage, the current matching is of maximum weight among all matchings of the same cardinality.

Hungarian Algorithm for Max Weight Bipartite Matching

1. Step 1: initialization: Let $M = \emptyset, u_i = W \equiv \max\{w_{ij} \mid (i,j) \in E\}, \forall i \in X$. For all $j \in Y$, set $v_j = 0$ and $\pi_j = \infty$
2. Step 2: scan:
   (a) Label $L(i) =$ Start for all unmatched vertex $i \in X$, and
   $$\text{LIST} = \{i \in X \mid i \text{ is unmatched}\}$$
   (b) If LIST is empty, go to step 4.
   (c) Remove a vertex k from LIST:
   - if $k \in X$, then for all $(k,j) \notin M$, set $\pi_j = \min\{\pi_j, u_k + v_j - w_{kj}\}$. If π_j is reduced, then set $L(j) = k$l if π_j is reduced to zero, add j into LIST
   - if $k \in Y$, if k is unmatched, go to step 3. Otherwise, find the unique edge $(k,i) \in M$, set $L(i) = k$, add i into LIST. Go to step2 (b)

3. Step 3: Augment: Update M, i.e., Trace augmenting path using $L(k), \dots$, and switch edges along augmenting path to be in and out of the matching. Reset all labels, i.e., set all $L(i)$ to null for all $i \in X \cup Y, \pi_j = \infty, \forall j \in Y$, Go to step2 (a)

4. Step 4: Update Dual: let $\delta = \min\{\delta_1, \delta_2\}$, where

$$\delta_1 = \min\{u_i : i \in X\}$$
$$\delta_2 = \min\{\pi_j : \pi > 0, j \in Y\}$$

   Set $u_i \leftarrow u_i - \delta$ for all $i \in X \cap F$, i.e. ,$L(i) \neq$ Null. Set $v_j \leftarrow v_j + \delta$ for all $j \in Y \cap F$, i.e., $\pi_j = 0$. Set $\pi_j \leftarrow \pi_j - \delta$ for all $j \in Y, j \notin F$, i.e., $\pi_j > 0$

   If any π_j is reduced to zero, add j into LIST. If $\delta = \delta_1$, stop. Otherwise, go to step (1b)

Hungarian Algorithm

0. Step 0 (Initialize): Let $M = \emptyset, u_i = W := \max\{w_{ij} \mid (i,j) \in E\}, \forall i \in X$, and $v_j = 0, \forall j \in Y$, and $\pi_j = \infty$.

1. Step 1 (Scan):

   (a) Denote Label $L(i) = \emptyset$ for all unmatched vertex $i \in X$, and LIST $= \{i \in X \mid i$ is unmatched$\}$.

   (b) If LIST is empty, go to step 3.

   (c) Remove a vertex k from LIST:

      i. For $k \in X$, for all $(k,j) \notin M$, set $\pi_j = \min\{\pi_j, u_k + v_k - w_{kj}\}$. If π_j is reduced, set $L(j) = k$; If π_j is reduced to zero, add j into LIST

      ii. For $k \in Y$, if k is unmatched, go to step 2. Otherwise, find the unique edge $(k,i) \in M$, set $L(i) = k$, add i to LIST. Go to step (1b)

2. Step 2 (Augment): Update M by trace augmenting path using $L(k), \dots$, and switch edges along augmenting path to be in and out of the matching. Reset all labels: set all $L(i)$ to Null for all $i \in X \cup Y$, $\pi_j = \infty, \forall j \in Y$. Go to step (1a)

3. Step 3 (Update Dual): Let $\delta = \min\{\delta_1, \delta_2\}$, where $\delta_1 = \min\{u_i \mid i \in X\}, \delta_2 = \min\{\pi_j \mid \pi > 0, j \in Y\}$. Then

$$\begin{aligned} u_i &\leftarrow u_i - \delta, \forall i \in X \cap F, \\ v_j &\leftarrow v_j + \delta, \forall j \in Y \cap F, \\ \pi_j &\leftarrow \pi_j - \delta, \forall j \in Y, j \notin F \end{aligned}$$

   If any π_j reduced to zero, add j to LIST. If $\delta = \delta_1$, stop, otherwise go to step (1b)

8

TSP

Problem setting Given a list of n cities and the distance c_{ij} from i to j, find a *tour* for the salesman such that each city is visited exactly once, before the salesman returns to the originating city, with total distance minimized. The variants can be added the assumption:

- symmetric: $c_{ij} = c_{ji}$
- Euclidean: c_{ij} is the Euclidean distance between i and j in 2-dimension plane
- Triangle Inequality: $c_{ij} \leq c_{ik} + c_{kj}, \forall i, j, k$

Suppose the actual travel distance/time between the cities is given by d_{ij}, Let c_{ij}be the shortest path from i to j (wrt d_{ij}), then c_{ij} satisfy the triangle inequality. The actual travel may pass through some cities more than once, but every city is visited.

8.1 Heuristics Method for Finding TSP Tour

Nearest Neighbour

1. Choose a city as originating city i_1 and $S_1 = \{i_1\}, k = 1$

2. For $k = 1 : n$:

   - Find a city not in S_k but closest to i_k, i.e., $i_{k+1} = \arg\min_{j \notin S_k} c_{i_k,j}$
   - Let $S_{k+1} = (i_1, \ldots, i_k, i_{k+1})$

However, the solution obtained can be arbitrarily bad, i.e., for any $r \geq 1$, we can construct an example where the ratio of the NN solution to the optimal TSP exceeds r. The idea: the tour successive driven towards a city where it is then forced to travel along an edge of large distance.

Nearest Addition

1. Start with a tour S of a single city.

2. Find a city not in S closest to the tour S, i.e., $\min_{i \in S, j \notin S} c_{ij}$, say $c_{k\ell} = \arg\min_{i \in S, j \notin S} c_{ij}$.

3. Let (k, p) be an edge incident to k on the tour S, replace it by (k, ℓ) and (ℓ, p), i.e., ℓ is inserted into the tour S next to k.

4. Repeat from step 2 unitl the tour is complete.

Nearest Insertion

1. Find a city ℓ closest to the tour S as above

2. Insert ℓ between u and v on the tour S, which minimizes the increase in tour length, i.e., replace (u, v) of S by (u, ℓ) and (ℓ, v) that minimizes
$$\min_{(i,j) \in S} c_{i\ell} + c_{\ell j} - c_{ij}$$

3. Repeat until the tour is complete

Now we consider the TSP with *non-negative symmetric* distances satisfying the *trianglular inequality.*

Minimum Spanning Tree Heuristic

1. Find a minimum weight spanning tree
2. Duplicate each edge in the spanning tree – to get an Eulerian graph.
3. Follow the Eulerian circuit and “skip ahead” if about to re-visit the city.

Theorem 8.1. Let MST be the value of the tour obtained by the Minimum Spanning Tree heuristic; Let SP^* be the value of the Minimum Spanning Tree; Let TSP^* be the value of the optimal Travelling Salesman Tour. For any instance of the problem, $MST \leq 2TSP^*$.

Proof. $MST \leq 2SP^*$. For the optimal tour, if we remove one edge of the tour, we get a spanning tree. Therefore $SP^* \leq TSP^*$, which follows

$$MST \leq 2SP^* \leq 2TSP^*$$

□

Christofides’ Heuristic A graph is Eulerian iff every vertex has even degree. We aim to get an Eulerian graph without doubling a spanning tree. Recall that in a spanning tree, there are an even number of vertices of *odd degree.*

1. Find a minimum weight spanning tree.
2. Find a minimum weight perfect matching of value M^* for the vertices of odd degree in the spanning tree.
3. The edges of the spanning tree, together with the edges of the matching yields an Eulerian graph. Find an Eulerian circuit.
4. To construct the TSP, follow the Eulerian circuit and “skip ahead” if next city has already been visited.

Theorem 8.2. Let CH be the value of the tour obtained by Christofides’ heuristic, then

$$CH \leq \frac{3}{2}TSP^*$$

Proof. By the triangle inequality, $CH \le SP^* + M^*$. Consider an optimal TSP tour, follow it and *skip* all the vertices of even degree in the optimal spanning tree SP^*, and we get a sub-tour S. By triangle inequality, the total distance $S \le TSP^*$. The edges of S can be partitioned into two matchings, M_1 and M_2, on the odd vertices of SP^*. Since the total length of the matchings is bounded by TSP^*, one of the two matchings must have length no greater than $TSP^*/2$. Therefore $M^* \le TSP^*/2$.

Also note that $SP^* \le TSP^*$. □

Clarke-Wright Savings Start with a set of sub-tours each originating at the depot city. Consider the savings if two cities y and z are put on the same sub-tour. Then save the distances.

The Clarke-Wright Savings algorithm begins with each sub-tour consisting of a trip from the depot to a single city. The pair of cities yielding the most savings when linked is first linked. Linking continues until one tour remains.

8.2 TSP Formulation as an Integer Programming

$$\begin{aligned}
\min \sum_{e \in E} c_e x_e & \\
\text{such that} \sum_{e \in \delta(i)} x_e = 2,\ & \forall i \in V \\
\sum_{\{e=(i,j) \mid i \in Q, j \notin Q\}} x_e \ge 2,\ & \forall \emptyset \subseteq Q \subseteq V \\
0 \le x_e \le 1,\ \text{integer}, & \forall e \in E
\end{aligned}$$

We can add additional valid inequalities so we can solve TSP as a linear program:

Consider subsets of vertices $H, T_1, \ldots, T_k \subseteq V$, with

$$|H \cap T_i| \geq 1$$
$$|T_i \setminus H| \geq 1$$
$$T_i \cap T_j = \emptyset$$
$$\sum_{e \in E(H)} x_e + \sum_{i=1}^{k} \sum_{e \in E(T_i)} x_e \leq |H| + \sum_{i=1}^{k} (|T_i| - 1) - \lceil \frac{k}{2} \rceil$$

This is the subtour elimination constraints

9

Graph Coloring

Problem Setting

Definition 9.1 (Colourable). Consider an undirected graph G without loops. G is c-colourable if each vertex can be assigned one of c colours such that the adjacent vertices have different colours.

Definition 9.2 (Chromatic). If G is k-colourable but not $(k-1)$-colourable, then G is called k-chromatic, and $\mathcal{X}(G) = k$.

w.l.o.g., assume that G is simple, loopless, and connected.
What is the chromatic number of:

- Complete graph K_n?
- The null graph (has no edges)?
- A bipartite graph?
- A path? A tree?
- A cycle? even or odd cycle?

Applications of Graph colouring

1. Exam period arranging: The vertices are classes. The class v and w are adjacent if v, w belong to the same student.

2. What is the minimum phases needed for the traffic light so that all cars in all lanes can go through the intersection? The vertices are traffice lanes. The lanes i and j are adjacent if there is a collision between them.

9.1 Facts about Colourability

Theorem 9.1. Let G be a simple graph with maximum vertex degree Δ. Then G is $(\Delta + 1)$-colourable.

Proof. By induction on the number of vertices. For the null graph with one vertex, it is 1-colourable. Consider simple graph G with n vertices. Delete any vertex v of G and its incident edges, the resultant graph G' has $(n - 1)$ vertices and degree at most Δ, which is $(\Delta + 1)$ colourable. Therefore, G is $(\Delta + 1)$-colourable by assigning a colour to v that is different to all of its adjacent vertices. $\square$

Theorem 9.2 (Stronger Result for Colourability)**.** Let G be a simple connected graph, and $\Delta(G)$ denote the maximum vertex degree of G. If G is neither a complete graph nor an odd cycle, then G is Δ-colourable, i.e., $\mathcal{X}(G) \leq \Delta(G)$.

Remark 9.1. Brooks' theorem give a tight bound for the chromatic number if a graph is regular, or "close" to regular, but it is not very helpful for graphs where a few vertices have very large degree

Theorem 9.3 (Colourability for Planar Graphs)**.** Every simple planar graph is 6-colourable.

Proof. Clearly, the result holds for all graphs with 6 or fewer vertices. Now we consider a planar graph G with n vertices, and assume that all simple planar graphs with $n - 1$ vertices are 6-colourable.

Recall that a simple planar graph must have a vertex v with degree at most 5. Removing this vertex and its incident edges, we obtain a graph with $n-1$ vertices that is 6-colourable. Assign a colour to v that is different to its 5 neighbours. $\square$

Theorem 9.4 (Colourability for Planar Graphs). Every simple planar graph is 5-colourable.

Proof. Clearly, the result holds for all graphs with 5 or fewer vertices. Now we consider a planar graph G with n vertices, and assume that all simple planar graphs with $n-1$ vertices are 5-colourable.

Recall that a simple planar graph must have a vertex v with degree at most 5. Removing this vertex and its incident edges, we obtain a graph with $n-1$ vertices that is 5-colourable.

- If this v has degree no more than 4, we are done.
- Otherwise v has degree 5 with neighbours $v_1, \dot{;} v_5$. If all v_i's are pairwise adjacent, $K_5 \subseteq G$, i.e., G cannot be planar. Suppose v_1 and v_2 are not adjacent. The graph contracting (v, v_1) and (v, v_2) are 5-colurable. We colour v_1 and v_2 with the colour assigned to the combined vertex in G', and colour v with a colour different from v_1, v_2, and the colours of v_3, v_4, v_5.

$\square$

Theorem 9.5. Every simple planar graph is 4-colourable.

Remark 9.2. a graph is obtained if a point is assigned in each county, and a line segment joins the points of two counties that share a boundary. Therefore, the 4-colour problem for maps is equivalent to the 4-colour vertex colouring problem for planar graphs.

9.2 Colouring Edges

Definition 9.3. Consider an undirected graph G without loops. G is h-edge-colourable if each edge can be assigned one of h colours such that the adjacent edges have different colours.

If G is k-edge-colourable but not $(k-1)$-colourable, then k is called the *chromatic index* of G, denoted as $\mathcal{X}'(G)$.

We assume that G is loopless and connected, but multiple edges are allowed.

Theorem 9.6 (Edge Colouring for Complete Graphs). For $n \geq 2$, $\mathcal{X}'(K_n) = n$ for odd n, and $\mathcal{X}'(n) = n - 1$ for even n.

Proof. Assume $n \geq 3$.

- For odd n, consider an n-cycle in K_n. Colour the edges of the cycle with a different colour for each edge. Colour each remaining edge automatically. We have a n-colurable scheme. The maximum number of edges with the same colour (non-adjacent) is $(n-1)/2$, so the maximum number of coloured edges is $\mathcal{X}'(K_n) \cdot (n-1)/2$, i.e., $\mathcal{X}'(K_n) = n$
- For even n, consider K_{n-1} obtained by removing one vertex v and its incident edges, which is $n-1$ coloruable. Colour that edge that joints each vertex to v with the missing colour.

□

Theorem 9.7. For a bipartite graph G with maximum vertex degree Δ, $\mathcal{X}'(G) = \Delta$.

Proof. The proof is by induction on the number of edges; we show that if all but one of the edges has been coloured using Δ colours, then the remaining edge can also be coloured to obtain a Δ-colouring of G. We call this an augmentation. □

Theorem 9.8. If G is a simple grpah, then $\Delta(G) \leq \mathcal{X}'(G) \leq \Delta(G) + 1$

However, determining whether a graph belongs to Class I or Class II is not easy.

10

Planar

Definition 10.1 (Planar). An undirected graph is *planar* if it can be drawn in the plane without *edges crossing*. Such a way of drawing is a *plane drawing* of the graph, or an *embedding* of the graph in the plane.

For example, K_4 is a planar graph, and only two way of drawing are plane drawings.

Characterisation of Planar Graphs

- Relationships among vertices, edges, and faces?
- Smallest non-planar graph?
- Necessary condition and sufficient conditions for a graph to be planar?
- Dual of a planar graph?
- Measure of non-planarity?

Definition 10.2 (Regular Polyhedron). A *regular polygon* is a convex figure (in 2-dimensions) where all edges are of same length and all (internal) angles at the vertices are the same.

A *regular polyhedron* is a solid (convex) figure (in 3 dimensions) with *all faces* being congruent regular polygons, with the same number arranged around each vertex.

Remark 10.1. For any plane drawing (embedding) of a planar graph, the plane is divided into faces. The face unbounded by the edges of the embedding is called the *infinite* face. A planar graph can have different embeddings, but the number of faces in any embedding is the same.

10.1 Euler's Polyhedral Formula

Theorem 10.1 (Euler)**.** Let G be a plane drawing of connected planar graph with n vertices and m edges. Let f be the number of faces of G. Then

$$n - m + f = 2$$

Proof. The proof is by induction on the number of edges m. For $m = 0$, we imply $n = 1, f = 1$, i.e., $n - m + f = 2$. Assume that the formula holds for all planar connected graphs with at most $m - 1$ edges, and let G be a plane drawing of a graph with m edges.

- If G is a tree, then $n = m + 1$ and $f = 1$, i.e., $n - m + f = 2$
- If G is not a tree, let e be an edge in some cycle of G, and $G - e$ is a connected planar graph with n vertices, $m - 1$ edges, and $f - 1$ faces. By induction hypothesis, $n - (m - 1) + (f - 1) = 2$, which follows that $n - m + f = 2$.

□

Corollary 10.2. Let G be a plane graph with n vertices, m edges and f faces, k components, then

$$n - m + f = k + 1$$

Corollary 10.3.
1. Let G be a simple connected planar graph with $n \geq 3$ vertices and m edges, then $m \leq 3n - 6$.
2. If in addition, G has no 3-cycles, then $m \leq 2n - 4$.

Proof. Bound the number of faces, i.e., each face is bounded by at least 3 edges, and each edge is on the boundary of at most two faces. Therefore, we have $3f \leq 2m$, i.e., $3(m - n + 2) \leq 2m$.

If there is no 3-cycles, then each face is bounded by at least 4 edges, and therefore $4f \leq 2m$, i.e., $4(m - n + 2) \leq 2m$ □

Corollary 10.4. Each simple planar graph contaisn a vertex of degree at most 5.

Proof. w.l.o.g., consider the connected graph with at least 3 vertices. If each vertex has degree at least 6, then $6n \leq 2m$, i.e., $3n \leq m \leq 3n - 6$, which is a contradiction. □

10.2 Non-planar Graphs

Theorem 10.5. The graph $K_{3,3}$ is non-planar.

Theorem 10.6. The graph K_5 is non-planar.

Proof. We can show the non-planarity of $K_5, K_{3,3}$ by Euler's Polyhedral Formula:

- For K_5, $n = 5, m = 10$, i.e., $3n - 6 = 9 < m$, contradiction
- For $K_{3,3}$, $n = 6, m = 9$, without 3 cycles, i.e., $2n - 4 = 8 < m$, contradiction.

□

Remark 10.2. Every subgraph of a planar graph is planar; Every graph with a non-planar subgraph must be non-planar; Every graph that 'contains' either K_5 or $K_{3,3}$ must be non-planar.

Definition 10.3 (homeomorphism)**.** When we replace an edge with a 2-edge path with a new vertex, we say that the edge has been sub-divided. We say two graphs are *homeomorphic* if they can be expanded from the same graph by edge sub-division.

Theorem 10.7. A graph is planar if and only if it contains no sub-graph homeomorphic to K_5 or $K_{3,3}$

Consider a graph H obtained from a graph G by *contracting* an edge.

Lemma 10.8. If the graph G is planar, then so is any graph H obtained from G by *contracting* an edge.

Theorem 10.9. If a graph G is contractible to either K_5 or $K_{3,3}$, then G cannot be planar.

10.3 Geometric Dual of a Planar Graph

Definition 10.4 (Geometric Dual)**.** Consider a plane drawing of a planar graph G, its geometric dual G^* is contructed as follows:

1. There is a vertex v^* of G^* for each face of G

2. If edge e in G separates two faces of G, then there is an edge e^* in G^* incident to the two vertices in G^* corresponding to two faces

3. If edge e is incident to only one face, then there is a corresponding self-loop in G^*.

Lemma 10.10. Let G be a connected planar graph, then G^* is also a connected planar graph. Let G be a planar graph with n vertices, m edges, f faces, and G^* be the geometric dual with n^* vertices, m^* edges, and f^* faces, then

$$n^* = f, \quad m^* = m, \quad f^* = n$$

Theorem 10.11. If G is a planar connected graph, then G^{**} is isomorphic to G.

Definition 10.5. A planar graph may also have self-dual, i.e., isomorphic to its geometric dual.

Theorem 10.12. Let G be a planar graph and G^* be a geometric dual of G. Then a set of edges of G form a cycle in G if and only if the corresponding set of edges of G^* form an *edge cutset* of G^*;

Proof. Assume G is connected. If C is a cycle of G, then C encloses one or more finite faces of G, corresponding to a non-empty set of vertices V^* in G^*. The edges in G^* corresponding to the edges of C form an edge cutset separating V^* from the rest of the graph G^*.

If K^* is a cutset of G^* separting the vertex sets V_1^*, V_2^* of G^*. w.l.o.g., V_1^* does not contain the vertex corresponding to the infinite face. For any two adjacent faces in G corresponding to $v_1^* \in V_1^*, v_2^* \in V_2^*$, the edge separating the two faces must be in the cutset K^*. Therefore, the edges in G corresponding to the edges in the cutset K^* forms a cycle. □

Corollary 10.13. A set of edges in G forms a cutset of G iff the corresponding edges in G^* form a cycle in G^*.

Definition 10.6. A graph G^* is an abstract dual of a graph G if there is a one-to-one correspondence between the edges of G and G^* such that: a set of edges form a cycle in G if and only if the corresponding edges form an edge cutset of G^*

The concept of the abstract dual generalises that of a geometric dual. If G is a plane graph, then its geometric dual is an abstract dual.

Proposition 10.1. $K_5, K_{3,3}$ has no abstract dual.

Theorem 10.14. A graph is planar if and only if it has an abstract dual

10.4 Crossing Number

Definition 10.7 (Crossing Number)**.** The crossing number $\mathrm{cr}(G)$ of a graph G is the minimum number of edge crossing that can occur when the graph is drawn in the plane.

Lemma 10.15. For a graph G with $n \geq 3$ vertices and m edges, $\mathrm{cr}(G) \geq m - 3n + 6$

Proof. For planar graph $m \leq 3n - 6$. Suppose G is non-planar with $\mathrm{cr}(G) = c$. For a plane drawing of G, "replace" each crossing with a new vertex (and 2 new edges). The resulting graph is planar and $(m + 2c) \leq 3(n + c) - 6$, i.e., $c \geq m - 3n + 6$. □

Lemma 10.16. For a complete bipartite graph $K_{p,q}$,

$$\mathrm{cr}(K_{p,q}) \leq \lceil p/2 \rceil \lceil (p-1)/2 \rceil \lceil q/2 \rceil \lceil (q-1)/2 \rceil$$

Remark 10.3. This bound is shown to be tight when $p \leq q$ and

- either $p \leq 6$
- or $p = 7, q \leq 10$

Definition 10.8 (Thickness). The thickness of a graph G is the fewest number of planar subgraphs whose union yields G.