UNIVERSITY
OF OSLO

**Master's thesis**

# Graph Theory and Decompositions

**Stine Myrvåg**

Mathematics, Lektorprogrammet
30 ECTS study points

Department of Mathematics
Faculty of Mathematics and Natural Sciences


Spring 2023

**Stine Myrvåg**

# Graph Theory and Decompositions

Supervisor:
Geir Dahl

## Abstract

This master's thesis primarily focuses on the decomposition of graphs into bipartite subgraphs. Prior to an exploratory section, the thesis begins with concepts and relevant graph theory, as well as network flow theory and some examples of the use of decomposition in flow algorithms. Furthermore, it presents applications for graphs and network flows. As an introduction to the exploratory part of the thesis, the Graham-Pollak Theorem is introduced. This section of the thesis is based on this theorem and examines what happens when changes are made to the conditions of the theorem. We investigate the number of bipartite subgraphs required to decompose a complete graph with $n$ nodes, and we examine the number of complete bipartite subgraphs necessary to decompose different types of graphs.

# Contents

# List of Tables



# List of Figures

# Acknowledgements

I would first and foremost like to thank my supervisor Geir Dahl. Thank you for giving me a problem that I have really enjoyed working with, and for the support and encouragement throughout the semester. I would also like to thank everyone in my study hall for the ups and downs, fun study breaks and unlimited support. Lastly, I would like to thank my family and friends for always believing in me, even though they most of the time understood nothing of the mathematics I have been studying throughout the last five years.

Chapter 1

# Introduction

## 1.1 Background

Graph theory and network flows have useful applications across a wide range of sciences. Decompositions are an important part in graph theory and network flows, and can be used to break down complex graphs into simpler components. This thesis mainly deals with bipartite decompositions of diffentert classes of graphs.

The exploratory research in this thesis is based on the Graham-Pollak Theorem,

**Theorem 1.1.1** (The Graham-Pollak Theorem, [AZ18, p. 79]). *If the complete graph $K_n$ is decomposed into complete bipartite subgraphs $H_1$, $H_2$, ... , $H_m$, then $m \geq n - 1$.*

We will explore several possible adjustments of the conditions in the theorem, and see if it is possible to find a minimum number of decompositions of a given class graph with the new conditions.

## 1.2 Outline

The first part is relevant and basic theory about graphs and network flows, as well as the Graham-Pollak Theorem which is the starting point for my independent research.

In Chapter 6, the $\gamma$-problem is introduced. We look at what would happen if the decomposition of the complete graph $K_n$ is decomposed into bipartite subgraphs instead of complete bipartite subgraphs. It is presented a pattern that gives a minimal decomposition, with an associated proposition. We start by proving the proposition for complete graphs on 4 and 5 vertices before we look at the general case for the complete graph on $n$ vertices.

In Chapter 7, the $\kappa$-problem is introduced. We look at what would happen if we do not decompose the complete graph $K_n$, but we instead look at complete bipartite decomposition of different classes of graphs. The classes taken into consideration are paths, cycles, wheels, generalised stars, and bipartite graphs. For each of the four first classes of graphs, it is presented a proposition with a following proof. For the bipartite graphs, we find an algorithm to find a complete bipartite decomposition.

Chapter 2

# Graph Theory

A graph $G$ consists of vertices and edges and can be expressed as the ordered pair $(V(G), E(G))$. $V(G)$ is the set of vertices, and $E(G)$ is the set of edges. This, together with an incidence function, $\psi_G$, which associates each edge in $E(G)$ to an unordered pair of vertices in $V(G)$, is an unambiguous graph. If both the set of edges and vertices are finite, we have a finite graph. In this master thesis we will only look at finite graphs [BM08].

The order of a graph $G$ is the defined by the number of edges of G and is denoted by $e(G)$. The size of a graph $G$ is denoted by $v(G)$ and is defined by the number of vertices. A graph can also be graphically represented. A graph may be drawn in many ways, as long as the relation between the edges and vertices is preserved. Each edge is represented by a line, and each vertex by a point [BM08]. Here follows an example with

$$G = (V(G), E(G))$$

where

$$\begin{aligned} V(G) &= \{v_1, v_2, v_3, v_4, v_5\} \\ E(G) &= \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8\} \end{aligned} \tag{2.1}$$

and the incidence function $\psi_G$ is defined by

$$\begin{aligned} &\psi_G(e_1) = v_1v_2, \quad \psi_G(e_2) = v_2v_3, \quad \psi_G(e_3) = v_3v_4, \\ &\psi_G(e_4) = v_1v_5, \quad \psi_G(e_5) = v_2v_5, \quad \psi_G(e_6) = v_4v_5. \end{aligned} \tag{2.2}$$

with the belonging drawing of the graph,

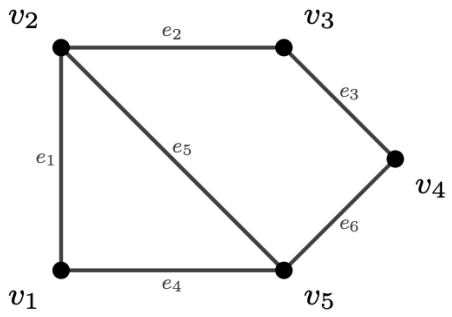



Figure 2.1: Drawing of a graph described with (2.1) and (2.2).

It is needed some more concepts to describe a graph. Two vertices are *adjacent* if there is an edge connecting them, and they are *incident* with any edge that are connecting them to another vertex. Two adjacent vertices are called *neighbours*. The edges of a graph can be either a *link* or a *loop*. If each of the ends of an edge is the same, we have a loop. If an edge have two distinct ends, we have a link. Two edges that are incident to the same pair of vertices are called *parallell edges* [BM08]. In Figure 2.2 it is demonstrated the different types of edges. The edge $e_4$ is a loop, $e_1$, $e_2$ are parallel edges and $e_1$, $e_2$, $e_3$, $e_5$ are links.

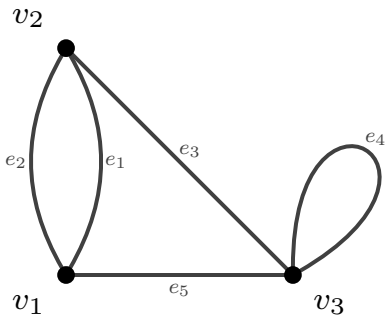



Figure 2.2: Example of parallel edges, links and loops.

### Different types of graphs

A *simple graph* consists of no loops or parallel edges, an example is shown in Figure 2.3. If in addition every vertex is adjacent we have a complete graph. The complete graph on $n$ vertices are denoted $K_n$. In Figure 2.4 there is an example of $K_5$. When working with simple graphs, there is no need for an incidence function $\psi_G$, we can instead denote each edge connecting the vertices $u$ and $v$ as $uv$ or $(u, v)$. [BM08]

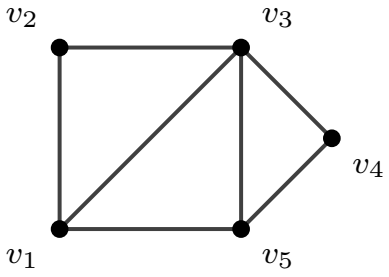



Figure 2.3: Example of a simple graph.

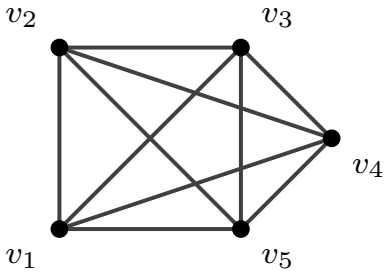



Figure 2.4: Example of a complete simple graph.

A *bipartite graph* $G$ is a graph where the set of vertices $V(G)$ can be divided into disjoint subsets $X$ and $Y$ such that every edge has one vertex in $X$ and one vertex in $Y$. If every vertex in $X$ has an edge adjacent to every vertex in $Y$, we have a *complete bipartite graph* [BM08]. An example of a complete bipartite graph and a bipartite grah is respectively shown in Figure 2.6 and Figure 2.5.

A *path* is a simple graph with a sequence of vertices and edges $(v_0, e_1, v_1, e_2, v_2, ...)$ such that you can move along each of the edges and not cross any vertex more than once. A

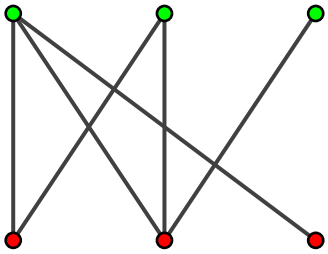


Figure 2.5: Example of a bipartite graph.

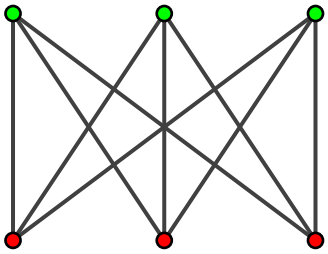


Figure 2.6: Example of a complete bipartite graph.

*cycle* is a simple graph such that if you arrange the vertices in a cyclic sequence, each consecutive pair of vertices are adjacent and if the pair of vertices are not consecutive, they are not adjacent. The length of a path or a cycle equals the number of edges. Hence, in Figure 2.7 we have a cycle of length 4 and in Figure 2.8 we have a path of length 3. A cycle is odd if the length is odd, and even if the length is even. A graph containing a cycle is called a cyclic graph, and acyclic if it contains no cycles [BM08].

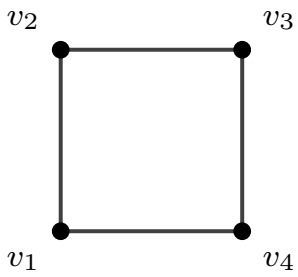



Figure 2.7: Example of a cycle.

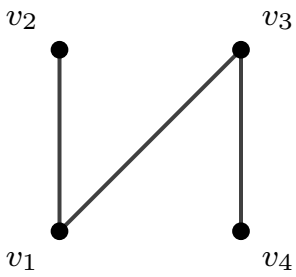



Figure 2.8: Example of a path.

A graph with a cycle on $n$ vertices and an extra vertex that are adjacent to each vertex in the cycle, is called a *wheel*. A wheel on $n+1$ vertices is denoted $W_n$ [Wei23]. The wheel $W_6$ is shown in Figure 2.9.

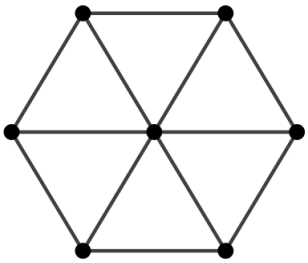


Figure 2.9: Example of a wheel on six vertices.

A graph is said to be *connected* if for each par of vertices we can find a path between them, see Figure 2.10. If you cannot find a path between any pair of vertices, the graph is *disconnected*, as in Figure 2.11 [BM08].

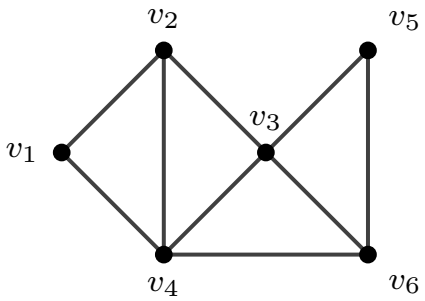



Figure 2.10: Example of a connected graph.

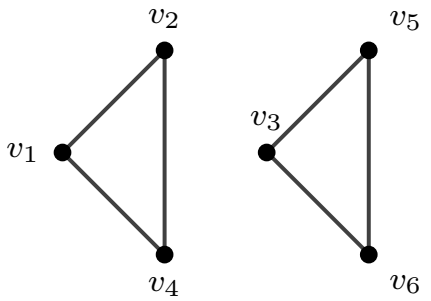
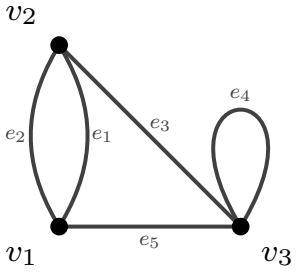



Figure 2.11: Example of a disconnected graph.

### Matrix representation of graphs

Drawings of graphs is a good visual presentation, but it is not a great way to store information in a computer. Instead, it can be stored as either an adjacency or incidence matrix. An *incidence matrix* is an $m \times n$-matrix, $M := (m_{ve})$ where $m = |V(G)|$ and $n = |E(G)|$. Each element $m_{ve}$ tells us how many times a vertex $v$ and an edge $e$ are incident. 0 if they are not incident, 1 if they are only incident once, and 2 if $e$ is a loop. An *adjacency matrix* is an $m \times m$-matrix $M := (a_{uv})$ where $a_{uv}$ tells us how many times two vertices are adjacent. A loop counts as two edges. This matrix is naturally symmetric. In Figure 2.12 there is the graph $G$ and its belonging incidence and adjacency matrices. When working with simple graphs there exists an easier and more compact way to describe a graph. This is a list such that for each vertex $v$, the neighbours are listed. A list of all these lists are $(N(v) : v \in V)$ and are called the *adjacency list*. Since this is a more efficient way to store information in a computer, simple graphs are usually stored as adjacency lists in computers [BM08].




(a)

| | $e_1$ | $e_2$ | $e_3$ | $e_4$ | $e_5$ |
|---|---|---|---|---|---|
| $v_1$ | 1 | 1 | 0 | 0 | 1 |
| $v_2$ | 1 | 1 | 1 | 0 | 1 |
| $v_3$ | 0 | 0 | 1 | 2 | 1 |

(b)

| | $v_1$ | $v_2$ | $v_3$ |
|---|---|---|---|
| $v_1$ | 0 | 2 | 1 |
| $v_2$ | 2 | 0 | 1 |
| $v_3$ | 1 | 1 | 2 |

(c)

Figure 2.12: (a) A graph $G$, with its associated (b) incidence and (c) adjacency matrix.

The number of edges that are incident with a vertex $v$ is called the *degree* of the vertex $v$, denoted by $d_G(v)$ [BM08]. For instance, in Figure 2.1, $d_G(v_1) = 3$.

### Trees

If an acyclic graph is connected, it is called a *tree*.

Proposition 2.0.1 ([BM08, p. 99]). *In a tree, any two vertices are connected by exactly one path.*

Thus, there is no obvious way to draw a graph if only the number of vertices and edges

are known. Figure 2.13 shows three different ways to draw a tree with five vertices and four edges.

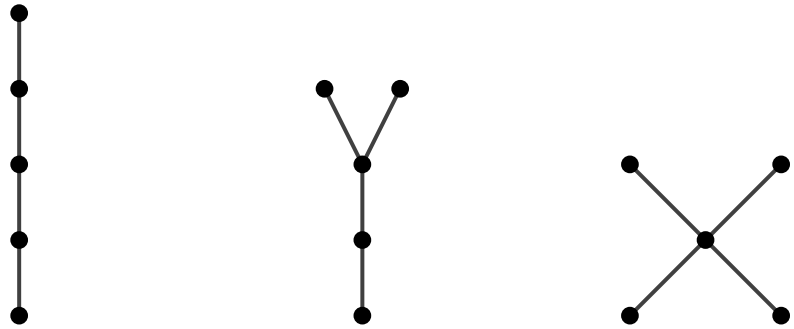



Figure 2.13: Three different types of trees on five vertices.

Theorem 2.0.2 ([BM08, p. 41]). *Let $G$ be a graph in which all vertices have degree at least two. Then $G$ contains a cycle.*

From Theorem 2.0.2 it is known that any graph where all vertices are of degree at least 2, we do have a cycle. It follows that a tree must have at least one vertex of degree at most 1. This type of vertex is called a leaf [BM08].

Proposition 2.0.3 ([BM08, p. 100]). *Every nontrivial tree has at least two leaves.*

Theorem 2.0.4 ([BM08, p. 100]). *If $T$ is a tree, then $e(T) = v(T) - 1$.*

A *star* is a tree $S$ where only one of the vertices has $d_S(v) > 2$, while every other vertex has $d_S(v) = 1$. In Figure 2.13c we a star on five vertices. A star on $n + 1$ vertices is denoted $S_n$. Hence, in Figure 2.13c we have $S_4$ [Wik23c].

### Subgraphs

From every graph there is possible to find a subgraph by either deleting one or more edges, deleting one or more vertices, or a combination of deleting edges and vertices. The subgraphs should indeed be graphs, that means if a vertex is deleted, then every edge incident to that vertex must be deleted as well [BM08].

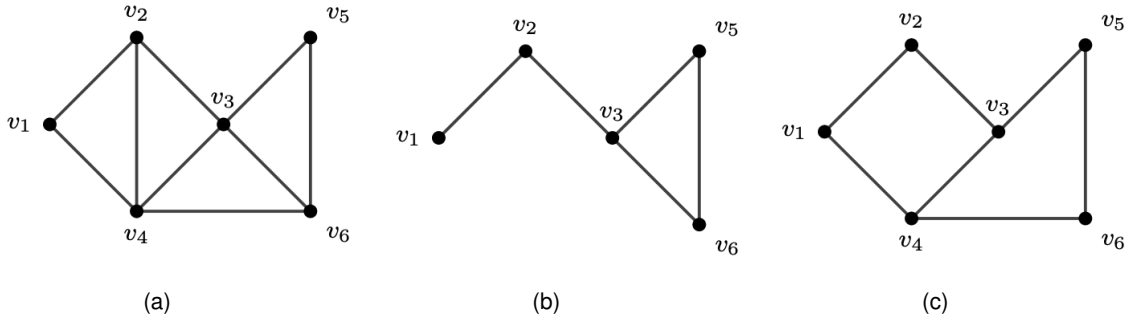



Figure 2.14: (a) A graph $G$ with examples of subgraphs made from (b) vertex- and (c) edge-deletion.

Examples of subgraphs by vertex-deletion and edge-deletion can be seen in Figure 2.14b and Figure 2.14c respectively. The different types of subgraphs can be denoted as $G \setminus e$ if we have an edge-deleting subgraph of G, and $G - v$ if we have a vertex-deleting graph. Every graph is a subgraph of itself. All other subgraphs is called *proper subgraphs* of G [BM08].

### Spanning Trees

If a subgraph of a graph is also a tree, we have a subtree. If the subgraph is in addition obtained by only deleting edges, it is a spanning subgraph [BM08].

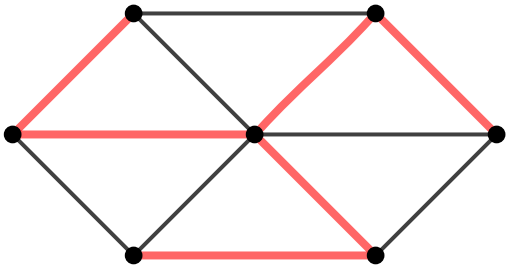


Figure 2.15: Example of spanning trees of a graph.

**Theorem 2.0.5** ([BM08], p. 106). *A graph is connected if and only if it has a spanning tree.*

**Theorem 2.0.6** ([BM08], p. 106). *A graph is bipartite if and only if it contains no odd cycle.*

A proof of this theorem follows from [BM08, p. 106].

*Proof.* Firstly, we need to show that if we have a bipartite graph, it contains no odd cycles. Start by letting $G = (I, J)$ be a connected bipartite graph. Because we have a bipartite graph we know that any path in $G$ has vertices alternately belonging to the set $I$ and the set $J$. This means that all paths connecting vertices in different sets are of odd length, and the paths connecting vertices in the same sets are of even length. Because we have defined G to be a bipartite graph, we know that each edge has one end in $I$ and one end in $J$. Hence, every cycle of $G$ is of even length.

Secondly, we need to show that if a graph contains no odd cycle, it is indeed bipartite. Now we start by letting $G = (I, J)$ be a connected graph with no odd cycles, and $V$ be the vertex set of $G$. From Theorem 2.0.5 we know that $G$ has a spanning tree $T$, and from Proposition 2.0.1 that any two vertices in $T$ are connected by a unique path in $T$. We then choose a vertex $x$ as a starting point, and every vertex $v$ that has en even path connecting $x$ and $v$ is in the set $I$, and $J := V \setminus I$. We then have that $(I, J)$ is a bipartition of $T$. We need to show that this is also a bipartition of $G$. We have an edge $e = uv$ of $E(G) \setminus E(T)$, and $P := uTv$ is the unique $uv$-path in $T$. Because $G$ only consists of even cycles, we know that $P + e$ is even, and $e$ is obviously odd, then $P$ is also odd. Hence, the edges of $P$ and $e$ belong to distinct sets. Therefore, $(I, J)$ is also a bipartition of $G$. ■

### Decomposition of graphs

A decomposition of a graph $G$ is a set $\mathcal{F}$ of subgraphs that all are pairwise edge-disjoint and fulfil the following equation,

$$\bigcup_{F \in \mathcal{F}} E(F) = E(G). \tag{2.3}$$

There are different ways to decompose a graph. In this thesis, we will focus on decompositions of different types of graphs into bipartite subgraphs. If a graph $G$ can be decomposed into only bipartite subgraphs or complete bipartite subgraphs, it is respectively a *bipartite-decomposition* or *complete bipartite decomposition* of $G$ [BM08].

Two ordinary decompositions to look at are *cycle decompositions* and *path decompositions*. In a cycle-decomposition every subgraph itself is a cycle. Similarly, a path-decompositon consists of subgraphs that are all paths. Every graph that contains no loops can be decomposed into a trivial path decomposition where each path is of length 1, but not every graph can be decomposed into a cycle decomposition. Because a decomposition should be edge-disjoint and $d_C(v) = 2$ for every vertex in a cycle, each vertex must be twice the number of cycles it is contained in. A graph is *even* if every vertex is of an even degree. Hence, if a graph have a cycle decomposition, if it is even [BM08].

**Theorem 2.0.7** (Veblen's theorem [BM08, p. 56]). *A graph admits a cycle decomposition if and only if it is even.*

A proof of the theorem follows from [BM08, p. 58].

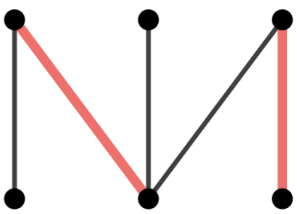
*Proof.* Assume we have a cycle that admits a cycle decomposition. In a cycle $C_n$ with vertices $v_1, ..., v_n$ we know that $deg_C(v_i) = 2$ for $i = 1, ..., n$. Thus, if we have a cycle decomposition of a graph $G$, we know that each vertex has twice the degree of the number of cycles it is included in. Hence, every vertex is of even degree and we have an even graph.

Suppose the graph $G$ is even. If $G$ is containing no edges then $E(G)$ is decomposed by the empty family of cycles. If not, consider the subgraph $F$ of $G$ that are induced by every vertex in $F$ that are of positive order. Hence, if $G$ is even, $F$ is also even, and we know that for every vertex $v \in V(F)$ $deg_F(v) \geq 2$. Using Theorem 2.0.2 we know that $F$ contains a cycle, $C$. This gives rise to a new subgraph $G' = G \setminus E(C)$ which also are even and consists of fewer edges than $G$. By the induction hypothesis, $G'$ has a cycle decomposition $C'$ and therefore $G$ has a cycle decomposition $C = C' \cup \{C\}$. ■

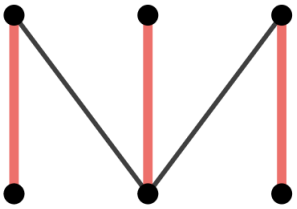
A related concept to decompositions is coverings. A *covering* is a set of subgraphs that fulfil (2.3), but the subgraphs does not need to be edge-disjoint. Hence, an edge can be covered multiple times. If each edge is covered $k$ times, it is called a *k-cover* and it is a $uniform$ covering. Similarly to decompositions of graphs, if a cover exclusively consists of paths of cycles, it is respectively a *path-covering* or a *cycle-covering* [BM08].

### Matching

A *matching* in a graph $G = (V, E)$ is a set of edges where each vertex is incident to a maximum of one edge. If two vertices are adjacent in a matching $M$, it is said that the vertices are *matched* under $M$. If a vertex is incident with an edge of $M$ is said to be *covered* by $M$. A $perfect$ matching covers every vertex of a graph, while a *maximum* matching covers as many vertices as possible, this is shown in Figure 2.16b. If a perfect matching exists, the graph is *matchable*. The number of edges in a maximum matching is called the *matching number*. A *maximal matching*, see Figure 2.16a, is a matching that can not be extended [BM08].



(a) A maximal matching



(b) A perfect and maximum matching

Figure 2.16: Example of (a) a maximal matching and (b) a perfect and maximum matching.

### Directed Graphs

A graph may not be sufficient to describe a problem. Every so often the orientation of an edge is useful to know. If the orientation of edges is known, it is a *directed graph*, also called a *digraph*. A digraph $D$ is an ordered pair $(V(D), A(D))$, where $V(D)$ is a set of vertices and $A(D)$ is a set of the directed edges, also called *arcs*. This, together with an *incidence function* $\psi_D$, describes a digraph. If $a$ is an arc and with the incidence function $\psi_D(a) = uv$, $u$ is the *tail* and $v$ is the *head* of the arc $a$ and that $u$ *dominates* $v$. For each vertex $v$, it can be distinguished between which vertices dominate $v$ and which vertices are dominated by $v$. The vertices that dominates $v$ is called its *in-neighbours* and can be denoted by $N_D^-(v)$, while the vertices that are dominated by $v$ are called its *out-neighbours* and can be denoted by $N_D^+(v)$ [BM08].

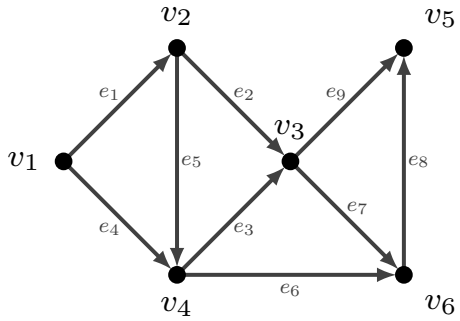



Figure 2.17: Example of a directed graph

Any digraph has an *underlying graph*, denoted $G(D)$, that is the same set of vertices, but with edges instead of arcs. Similarly, an *associated digraph* of $G$, denoted $D(G)$, has the same vertex set as in $D$, but arcs instead of edges. As with graphs, a digraph can be visually represented, where the arcs are edges that have arrows from the tail to the head of the edge [BM08]. An example is given in Figure 2.17.

The concept of degrees of a vertex in a digraph is very similar to the degrees in an undirected graphs, but now it is also distinguished between the *in-* and *outdegree*, which are respectively denoted by $d_D^-(v)$ and $d_D^+(v)$. It can also be separated between the maximum and minimum in- and outdegree of a digraph, $D$. The minimum in- and outdegree can be denoted by

$$\delta^+(v) = \min\left\{d_D^+(v), \quad v \in D\right\}$$
$$\delta^-(v) = \min\left\{d_D^-(v), \quad v \in D\right\}.$$

The maximum in- and outdegree can be denoted by,

$$\Delta^+(v) = \max\left\{d_D^+(v), \quad v \in D\right\}$$
$$\Delta^-(v) = \max\left\{d_D^-(v), \quad v \in D\right\}.$$

A *source* is a vertex $v$ with $d_D^-(v) = 0$ and a *sink* is a vertex $v$ with $d_D^+(v) = 0$. It is a *directed path* or *cycle* if every vertex dominates the succeeding vertex in the sequence [BM08].

Chapter 3

# Network Flow

In this chapter we will look at network flows, which is a continuation of the directed graphs that were presented in Chapter 2. A network flow is a concept used to describe the movement of a resource through a network of interconnected vertices and edges. In a network flow, each edge and vertex is assigned a value. This value can describe the requirement at each vertex and the possible flow on each edge [Dah13]. Figure 3.1 shows an example of a network flow.

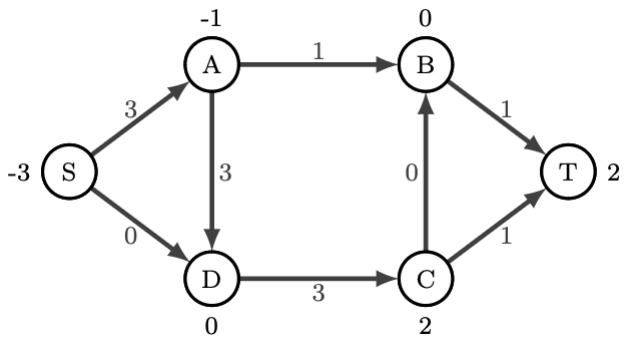



Figure 3.1: Example of a network.

## 3.1 Network flow theory

This section, Section 3.1, is based on theory from Geir Dahl's note "Network flows and combinatorial matrix theory" [Dah13].

In a network flow, some special sets are of extra interest.

$$\delta^+(v) = \{e \in E : e = (u, w) \text{ for some vertex } w \in V\} : \text{the set of edges leaving } v.$$

$$\delta^-(v) = \{e \in E : e = (u, w) \text{ for some vertex } w \in V\} : \text{the set of edges entering } v.$$

The flow between vertices can be described with the function

$$x : E \to \mathbb{R}.$$

Hence, $x$ assigns a value $x(e)$ to each edge $e$, which is the flow in the given edge. The divergence of a flow is the difference between the total inflow and outflow at each edge. This can be described with the following equation,

$$div_x(v) = \sum_{e \in \delta^+(v)} x(e) - \sum_{e \in \delta^-(v)} x(e).$$

In general it is

$$\sum_{v \in V} div_x(v) = 0.$$

The most interesting network flow is when the divergence is given

$$b : V \to \mathbb{R},$$

hence $b = div_x$ and $\sum_{v \in V} b(v) = 0$ and it satisfies

$$b(v) = \sum_{e \in \delta^+(v)} x(e) - \sum_{e \in \delta^-(v)} x(e).$$

This is called the *flow balance equation*.

If $div_x = O$, where $O$ is the zero vector, it is a *circulation*, and in every vertex, the total inflow equals the total outflow. This is called *flow conservation*.

In a network flow, there exists a *capacity function* along with the divergence function. The capacity function $c$, like the divergence function, assigns a value to each of the network's directed edges. As the name suggests, the capacity function describes the capacity at each edge,

$$c : E \to \mathbb{R}.$$

Thus, it can be looked upon as how many units of flow it is possible to get through each edge. This naturally gives rise to a capacity constraint. It is obviously not possible to send more units of flow through a given edge than there is capacity for at that edge. A flow is said to be feasible if the following inequality is satisfied

$$0 \leq x(e) \leq c(e) \qquad \text{for all} \qquad e \in E.$$

In a digraph, a *directed path* is a sequence alternating between distinct vertices and edges,

$$P : v_0, e_1, v_1, ..., e_t, v_t$$

where $e_i = (v_{i-1}, v_i)$ and $1 \leq i \leq t$. $P$ can also be called $v_0v_t$-*path* or a path from $v_0$ to $v_t$.

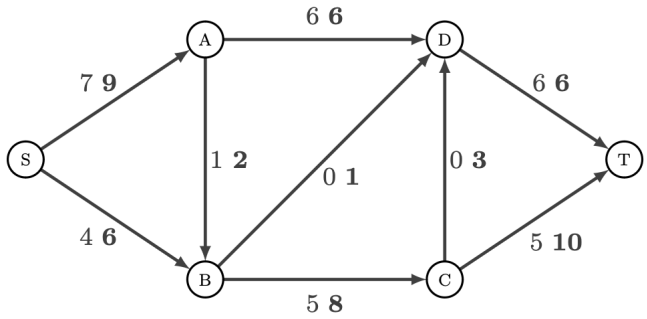



Figure 3.2: A network flow with capacity constraints assigned to each edge, shown with bold font.

### 3.1.1 Existence of circulations and flows

The boundary of a subset $S \subseteq V$ can be described with the following properties

$$\delta^+(S) = \{e \in E : e = (u, w), v \in S, w \notin S\} : \text{the set of edges leaving } S,$$

$$\delta^-(S) = \{e \in E : e = (u, w), v \notin S, w \in S\} : \text{the set of edges entering } S.$$

It is also needed an *auxiliary graph* $D_x = (V, E_x)$ associated with the flow $x$ where

$$E_x = \{e \in E : x(e) < u(e)\} \cup \{e^{-1} \in E : l(e) < x(e)\}$$

and $e^{-1} = (v, u)$, "the inverse edge" of $e = (u, v)$.

**Theorem 3.1.1** (Hoffman's circulation theorem, [Dah13, p. 4]). *Let $l, u : E \to \mathbb{R}$ be edge functions satisfying $l \leq u$. Then there exists a circulation $x$ in $D$ such that*

$$l \leq x \leq u$$

*if and only if*

$$\sum_{e \in \delta^-(S)} l(e) = \sum_{e \in \delta^+(S)} u(e) \qquad (S \subseteq V).$$

*Moreover, if $l$ and $u$ are integral (the function values are integral), then $x$ can be taken to be integral.*

The proof can be found in [Dah13, p. 4].

### 3.1.2 Maximum flow and minimum cut

In this subsection, the maximum flow problem and the minimum cut problem will be presented.

Let $D = (V, E)$ be a directed graph with nonnegative edge capacity function $c : E \to \mathbb{R}_+$. There is a source $s$ and a sink $t$. An *st-flow* is a flow $x$, satisfying

$$\sum_{e \in \delta^+(v)} x(e) = \sum_{e \in \delta^-(v)} x(e) \qquad (v \in V \setminus \{s, t\})$$

$$O \leq x \leq c.$$

The value of an *st*-flow is the total outflow from the source,

$$\mathrm{val}(x) = \sum_{e \in \delta^+(v)} x(e).$$

In an *st*-flow there will be no edge entering the vertex $s$ and therefore $\mathrm{val}(x) = \mathrm{div}_x(s)$. This gives rise to one of the two problems presented in this chapter, the *maximum flow problem*. The problem is to find an *st*-flow $x$ that maximises the value of the flow, hence, a *maximum flow*.

The other optimisation problem is to find an *st-cut*. This is a subset of edges $K = \delta^+(S)$ for a vertex set $S \subseteq V$ with $s \in S$ and $t \notin S$. The capacity of each edge can be described with $c : E \to \mathbb{R}_+$, and the *capacity* of the *st*-cut $K$ is

$$\mathrm{cap}_c(K) = \sum_{e \in K} c(e).$$

This gives rise to the other optimisation problem presented in this chapter, the *minimum cut problem*. The problem is to find an *st*-cut $K$ with as small capacity as possible. This cut $K$ is called a *minimum cut*.

Lemma 3.1.2 ([Dah13, p. 7]). *The following inequality holds*

$$max\{val(x) : x \text{ is st-flow}\} \leq min\{cap_c(x) : K \text{ is st-cut}\}.$$

The inequality in Lemma 3.1.2 is actually an equality and was proven by Dantzing and Fulkerson. This gives the *max-flow min-cut theorem*, an important result in combinatorics and combinatorial optimisation.

Theorem 3.1.3 (Max-flow min-cut theorem, [Dah13, p. 8]). *For any directed graph D, edge capacity function c, and distinct vertices s, t, the value of a maximum st-flow equals the minimum st-cut capacity, i.e.,*

$$max\{val(x) : x \text{ is st-flow}\} = min\{cap_c(x) : K \text{ is st-cut}\}. \tag{3.1}$$

The following proof is based on Dahl's proof from [Dah13].

*Proof.* It is already known from Lemma 3.1.2 that there is a maximum *st*-flow less than or equal to the minimum *st*-cut. Hence, it is only needed to show that there in fact exists an *st*-flow that is equal to the minimum cut capacity $M$. The graph $D$ does not contain the edge $(t, s)$. The graph $D' = D \cup (t, s)$. Define $l(t, s) = u(t, s) = M$ and $l(e) = 0$, $u(e) = c(e)$ for each $e \in E$. In order to continue the proof, Hoffman's circulation theorem Theorem 3.1.1, is used. Look at the case where $s \in S$, $t \notin S$. Hoffman's theorem then gives

$$\sum_{e \in \delta^-(S)} l(e) = M + 0 = M$$

and

$$\sum_{e \in \delta^+(S)} u(e) = \sum_{e \in \delta^+(S)} c(e) = \mathrm{cap}_c(\delta^+(S)).$$

This gives

$$\text{cap}_c(\delta^+(S)) \geq M \qquad (S \subseteq V, s \in S, v \notin S).$$

Because $M$ is the minimum cut capacity this condition is satisfied. From Hoffman's circulation theorem Theorem 3.1.1 it is clear that there is a circulation $x$ in $D'$ with $l \leq x \leq u$. Hence, $x(t,s) = l(t,s) = u(t,s) = M$, and therefore the flow $x$ is an $st$-flow with val($x$) = M. ■

### 3.1.3 The Ford-Fulkerson Algorithm

The *Ford-Fulkerson algorithm* is a common algorithm for finding a maximum $st$-flow in a network. There exists other algorithms that compute this and even though they might be more efficient, the Ford-Fulkerson is common because it is neat and simple.

The conditions are the same as earlier. There is a digraph $D$, vertices $s$ and $t$, a nonnegative capacity function $c$ and assume that $D$ contains an $st$-path. If $x$ is an $st$-flow it can be constructed an *auxiliary graph* $D_x = (V, E_x)$ where

$$E_x = \{e \in E : x(e) < c(e)\} \cup \{e^{-1} : e \in E, \;\; x(e) > 0\}. \tag{3.2}$$

In the auxiliary graph $D_x$ every $st$-path $P$ is called an *x-augmenting path.* The $P$-edges that also lie in $E$ are called *forward edges* and are in the set $P^+$, the rest of the edges from $E$ corresponds to *backward edges* in D and are in the set $P^-$. The auxiliary graph to the graph in Figure 3.2 is presented in Figure 3.2.

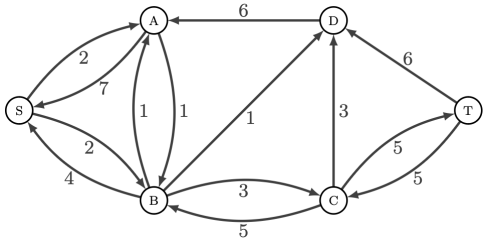



Figure 3.3: The associated auxiliary graph to Figure 3.2.

**Theorem 3.1.4** ([Dah13, p. 9]). *Let $x$ be an $st$-flow. Then $x$ is a maximum flow if and only if $D_x$ contains no $x$-augmenting path.*

The following proof is based on Dahl's proof in [Dah13].

*Proof.* Start by assuming there is an augmenting path $P$ in $D_x$. Let $\epsilon$ be the minimum of the following numbers

(*i*) $c(e)$ - $x(e)$ for each $e \in P^+$

(*ii*) $x(e)$ for each $e \in P^-$.

Hence, $\epsilon > 0$. Then it can be made a greater $st$-flow by adding $\epsilon$ to the flow in each edge $e \in P^+$ and subtracting $\epsilon$ for each edge in $P^-$. This gives a new flow called $x'$, which is also an $st$-flow because the difference between the total outflow and total inflow in every vertex $v \neq s, t$ is zero. In addition $\text{val}(x') = \text{val}(x) + \epsilon$. Since $\epsilon > 0$, $\text{val}(x') > \text{val}(x)$. Hence, if $D_x$ contains an augmenting path, there will always be possible to find a greater value for the $st$-flow.

Assume now that $D_x$ contains no $x$-augmenting path. $S(x)$ is the set of vertices to which we can find an augmenting $sv$-path in $D_x$, and define the cut $K = \delta^+(S(x))$. Then $x(u, v) = c(u, v)$ for each edge $e = (u, v) \in K$, otherwise $v \in S(x)$. Also note that $x(u, v) = 0$ for each edge $e = (u, v)$, $u \notin S(x)$, $v \in S(x)$. The flow in each edge in the cut $K$ is as great as possible while the flow in the reverse cut $\delta^-(S(x))$ is zero. Hence, $\text{val}(x) = \text{cap}_c(K)$. From Lemma 3.1.2 we then get that $x$ is a maximum $st$-flow and $K = \delta^+(S(X))$ is a minimum $st$-cut. ■

### Ford-Fulkerson max-flow algorithm

The Ford-Fulkerson algorithm to find a maximum flow is described as follows in [Dah13, p. 10].

1. Start with the zero flow $x = O$.
2. Look for an $x$-augmenting path $P$ in $D_x$.
   (a) If such path $P$ exists, then find the maximum possible increase $\epsilon$ of flow in $D$ along the path corresponding to $P$. Augment the flow $x$ accordingly.
   (b) If no such $P$ exists, then the present $x$ is a maximum flow. Moreover, a minimum $st$-cut if $\delta^+(S(x))$ where $S(x)$ denote the set of vertices to which we can find an augmenting $sv$-path in $D_x$.

In order to find an $x$-augmenting path in $D_x$ let $V_0 = \{s\}$. Then, iteratively, let $V_{i+1}$ be the vertices $V \setminus (V_0, ..., V_i)$ that can be reached by only one edge from a vertex in $V_i$.

The value of the flow is increased by at least one unit in each of the flow augmentations. This means that the Ford-Fulkerson algorithm requires at most the same iterations as the value of the maximum flow $M$.

Example 3.1.5. Find the maximum flow through the network shown in Figure 3.4. The bold typing along the arcs are the capacities, while the not bold typing is the flow, which initially is 0 at every arc.

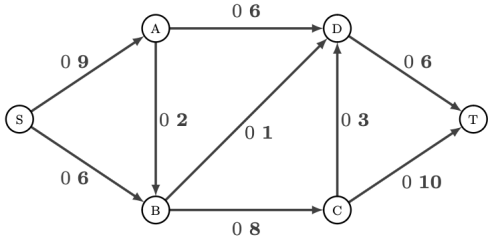



Figure 3.4: A network to find a maximum flow through.

1. The first $x$-augmenting path $P_1$ can be $P_1 : \ S - A - B - C - T$. The maximum possible increase is $\epsilon = 2$ along this path. This results with the auxiliary graph in Figure 3.5a.
2. The next $x$-augmenting path $P_2$ can be $P_2 : \ S - A - D - T$. The maximum possible increase is $\epsilon = 6$ along this path. This results with the auxiliary graph in Figure 3.5b.
3. The next $x$-augmenting path $P_3$ can be $P_3 : \ S - B - C - T$. The maximum possible increase is $\epsilon = 6$ along this path. This results with the auxiliary graph in Figure 3.5c.
4. It is not possible to find any $x$-augmenting path Figure 3.5c, and we have a maximum flow, as shown in Figure 3.5d.

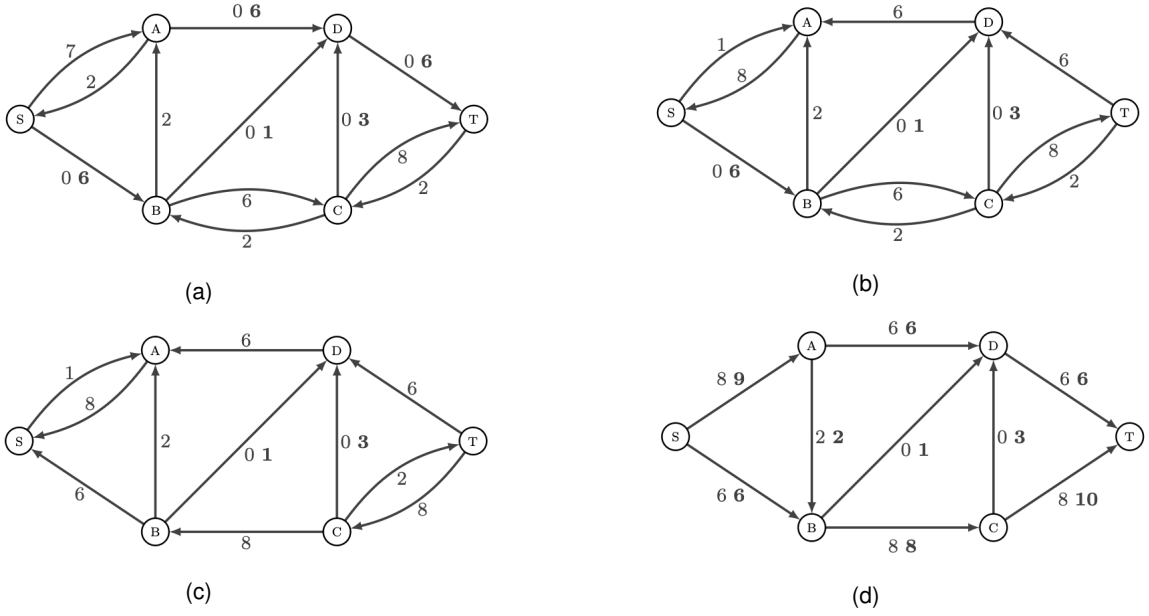



Figure 3.5: (a) The auxiliary graph after increasing the flow with $\epsilon = 2$ on the path $S-A-B-C-T$. (b) The auxiliary graph after increasing the flow with $\epsilon = 6$ on the path $S - A - D - T$. (c) The auxiliary graph after increasing the flow with $\epsilon = 6$ on the path $S - B - C - T$. (d) A maximum flow through the original graph.

## 3.2 The Flow Decomposition Theorem

This section, Section 3.2, is based on theory from "Network flows. Theory, algorithms, and application" [AMO93].

In this section the Flow Decomposition Theorem is presented. This theorem is important because it breaks down a complicated flow problem into smaller problems that can be solved more easily. The theorem identifies which elements of the network are contributing to the flow by decomposing the flow into pathways and cycles, and we can focus on optimising the flow along each path and cycle independently. Before presenting the theorem and the algorithm obtained from the proof, there is a need for some more definitions than what is already presented in Section 3.1.

The flow of each arc can be described with the linear system

$$\begin{aligned} &\sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(i,j)\in A} x_{ij} = -e(i) \qquad \forall i \in N \\ &0 \le x_{ij} \le u_{ij} \qquad \forall i \in N \\ &\text{where} \qquad \sum_{i=1}^{n} e(i) = 0 \end{aligned} \tag{3.3}$$

The integer $e(i)$ tells the difference in inflow and outflow of each vertex and $e(i)$ is the imbalance in the vertex $i$ [AMO93, p. 80].

- $e(i) > 0$: excess vertex
- $e(i) < 0$: deficit vertex
- $e(i) = 0$: balanced vertex

It is needed notation to distinguish between flows on a path $P$ and a cycle $W$.

$$\begin{aligned} \mathcal{P} &: \quad \text{collection of all directed paths} \\ \mathcal{W} &: \quad \text{collection of all directed cycles} \end{aligned}$$

For every directed path $P \in \mathcal{P}$ and every directed cycle $W \in \mathcal{W}$ there is the following variables

$$\begin{aligned} f(P) &: \quad \text{the flow on path P} \\ f(W) &: \quad \text{the flow on path W} \end{aligned}$$

A flow $x_{ij}$ can be described by summing the flows $f(P)$ and $f(W)$ for the paths and cycles that are contained within the given arc $(i, j)$. To describe the flow $x_{ij}$ it can be introduced some new variables as in [AMO93, p. 80],

$$x_{ij} = \sum_{P\in\mathcal{P}} \delta_{ij}(P) f(P) + \sum_{W\in\mathcal{W}} \delta_{ij}(W) f(W),$$

where

$$\begin{aligned} \delta_{ij}(P) = 1; &\quad \text{if arc } (i,j) \text{ is contained in } P \\ \delta_{ij}(P) = 0; &\quad \text{if arc } (i,j) \text{ is not contained in } P \\ \delta_{ij}(W) = 1; &\quad \text{if arc } (i,j) \text{ is contained in } W \\ \delta_{ij}(W) = 0; &\quad \text{if arc } (i,j) \text{ is not contained in } W \end{aligned}$$

This shows that each path and cycle flow determines arc flows uniquely.

Theorem 3.2.1 (Flow Decomposition Theorem [AMO93, p. 80]). *Every path and cycle flow has a unique representation as nonnegative arc flows. Conversely, every nonnegative arc flow $x$ can be represented as a path and cycle flow (though not necessarily uniquely) with the following properties:*

- *Every directed path with positive flow connects a deficit vertex to an excess vertex.*

- *At most $n + m$ paths and cycles have nonzero flow; out of these, at most $m$ cycles have nonzero flow.*

The following proof is obtained from [AMO93].

*Proof.* We have already observed that each path and cycle flow determines arc flows uniquely, but we need to show the contrary. The proof of this is an algorithm that shows that any arc flow, $x$, can be decomposed into a path and cycle flow.

Suppose $i_0$ is a deficit vertex, then we have an arc $(i_0, i_1)$ with positive flow. If $i_1$ is an excess vertex, we stop. If this is not the case, (3.3) implies that we have some other arc $(i_1, i_2)$ that carries a positive flow. Repeat this until we either encounter an excess vertex or we revisit a vertex. If we encounter an excess vertex we obtain a directed path from the deficit vertex $i_0$ to an excess vertex $i_k$. If we revisit a vertex we obtain a directed cycle. If we obtain a directed path we let

$$f(P) = \min\{-e(i_0), e(i_k), \min\{x_{ij} : (i, j) \in P\}\}$$

and redefine

$$e(i_0) = e(i_0) + f(P)$$
$$e(i_k) = e(i_k) - f(P)$$

and for every arc $(i, j) \in P$

$$x_{ij} = x_{ij} - f(P).$$

If we obtain a directed cycle we let

$$f(W) = \min\{x_{ij} : (i, j) \in W\}$$

and for every arc $(i, j) \in W$

$$x_{ij} = x_{ij} - f(W).$$

Repeat this until all vertex imbalances are zero. Then choose any vertex with a least one outgoing arc with a positive flow as the new $i_0$ and repeat the procedure, which in this case must find a directed cycle. Terminate when the arc flow, $x = 0$ for the redefined problem. The original flow is now the sum of flows on the paths and cycles identified by this method.

Each time we identify a directed path we reduce that excess/deficit of some vertex to zero or the flow on some arc to zero, and each time we identify a directed cycle we reduce the flow on some arc to zero. This means that the path and cycle representation of the given flow $x$ contains at most $n + m$ directed paths and cycles, and at most $m$ of these are directed cycles.

■

Example 3.2.2. The proof of Theorem 3.2.1 gives a flow decomposition algorithm. The algorithm can be demonstrated on the network flow shown in Figure 3.1.

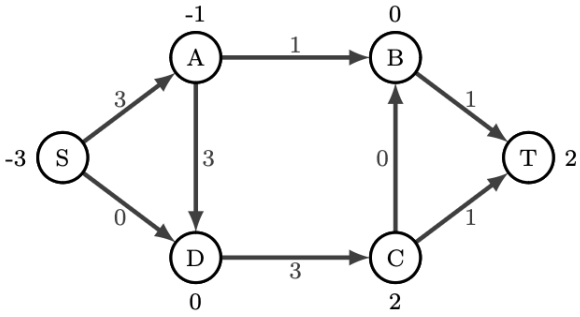



Figure 3.1: Example of a network.

1. We start with the deficit vertex $A$. We then obtain the directed path $P_1 : A - D - C - T$ with a flow of 1 unit. See Figure 3.6a.
2. We then choose $S$ as the next deficit vertex. We then obtain the directed path $P_2 : S - A - D - C$ with a flow of 2 units. See Figure 3.6b.
3. $S$ is still a deficit vertex and starting with $S$ we obtain the directed path $P_3 : S - A - B - T$ with a flow of 1 unit. See Figure 3.6c.
4. Now there are no more deficit vertices, and we have a complete decomposition of the network in Figure 3.1, made up by $P_1$, $P_2$ and $P_3$.

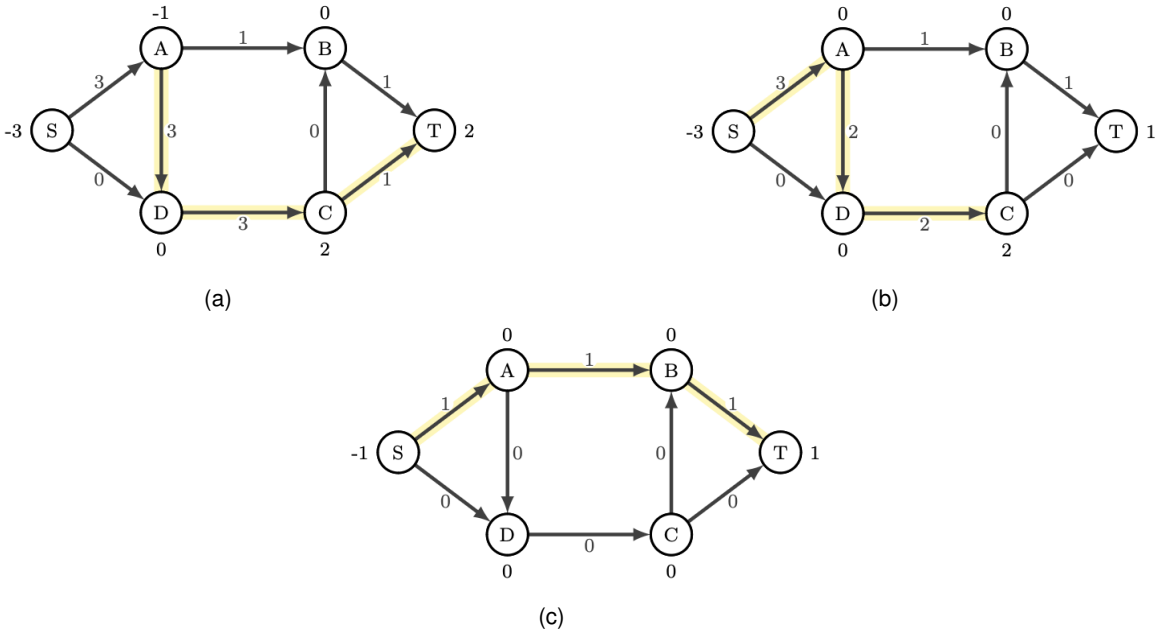



Figure 3.6: A decomposition of Figure 3.1 using the flow decomposition algorithm. (a) In yellow, the path $P_1 : A - D - C - T$ (b) The flow without $P_1$ and, in yellow, the path $P_2 : S - A - D - C$ (c) The flow without $P_1, P_2$ and, in yellow, the path $P_3 : S - A - B - T$.

Chapter 4

# Practical use of graphs and network flows

Graphs have a great number of real-life applications. The different types of graphs can be used to describe different types of problems. Graphs may not only be relevant in the natural sciences, but for instance also in linguistics and social sciences [Fou12]. The following sections describe some examples of the use of graphs.

## 4.1 Graphs in chemistry

Every molecule can be described with a structural formula, which again can be represented by a graph. Each atom is a vertex and the bonds are the edges. If there are some atoms with more than one binding connecting them, this is shown with parallel edges between the given atoms [Fou12]. An example of this is the formula of acetic acid, $CH_3COOH$, as shown in Figure 4.1.

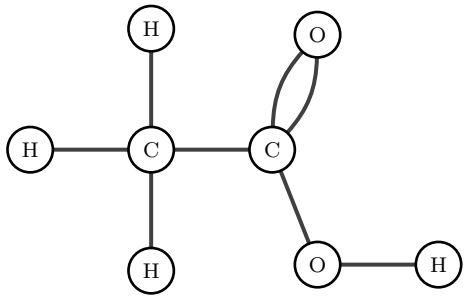



Figure 4.1: Acetic acid $CH_3COOH$ represented as a graph.

## 4.2 Graphs in civil engineering

Graphs are commonly used to model problems that civil engineers encounter. Problems such as traffic flow through a city or problems where mass needs to be transported from a construction site where it has been excavated to another area that might be in need of the mass [Fou12].

### 4.2.1 Earthwork problems

Earthwork problems are a common example of something that can be shown using a graph. For instance, in smaller cities around Oslo, there is a fast development of the areas including building of a great amount of apartment buildings. Such work may require

earth fillings. This earth fillings need to be dug from suitable places, *borrow pits*, and transported to the construction site. Each borrow pit has a known amount of fillings, the construction site require a known amount of filling at several spots, and it is a known cost to transport the fillings from the borrow pit to the different spots with requirements. This gives rise to a problem of finding the minimum cost of this transportation [Fou12].

This is a problem that can be modelled with a complete bipartite graph, $G$, with the partitions $I$ and $J$ such that $I \cup J = G$. $I$ is the borrow pits, with belonging values of how much filling is available. $J$ is the spots where the filling is required, with belonging values of the requirements at each spot. The value along an edge $v_i v_j$ if the cost of transporting from $v_i$ to $v_j$. An example is shown in Figure 4.2, with $I = \{A, B, C\}$ as the borrow pits and $J = \{D, E, F; G\}$ as the areas needing filling [Fou12].

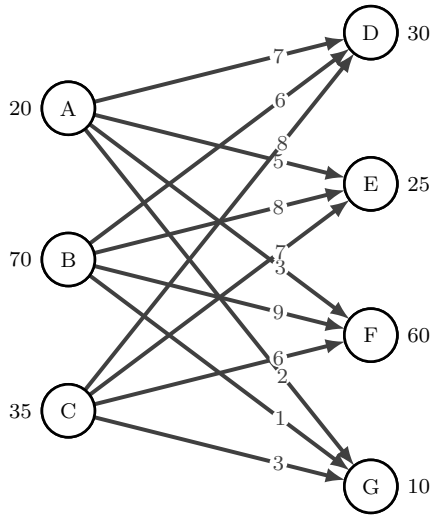



Figure 4.2: A model of a earthwork project.

## 4.3 Social science

Graphs are a very aesthetically pleasing and easily understandable way to present connections between people. Whether it is used to describe a person's connections in social media or who has collaborated with whom in a group of mathematicians.

### 4.3.1 Connections in social media

A trivial example of a possible use of graphs, that most are familiar with, is friends/connections on social media. Everyone can, in theory, connect with anyone, no one can connect with oneself (a loop), and you can only connect with someone once (no parallel edges). In Figure 4.3 you can see an example of a social network. Taking Anne as a starting point, the graph can describe the connection between her friends on a social platform, and again show which of her connections that are connected.

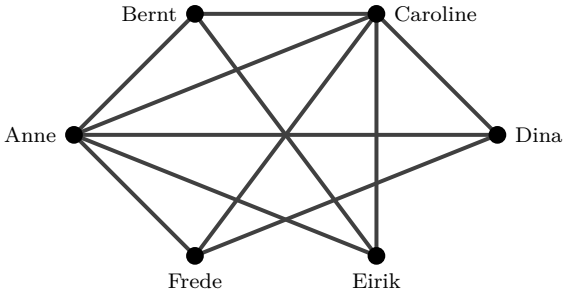



Figure 4.3: Example of a graph used to describe social connections.

### 4.3.2 Co-authorship graps and Erdős number

In science research collaboration between scientists can be an important part of the work. An example from physics is the detection of elementary particles, where it is needed a lot of people to run accelerators. This means that many physicists have collaborated and are in some way connected to each other. Hence, it can be presented as a graph. Collaboration in mathematics may not be as common as in other fields of science, but it may nevertheless be interesting to look at the connection between mathematicians [Fou12].

When using the words scientists/mathematicians in this section, it is important to note that they are indeed published scientists/mathematicians.

#### Co-authorship graphs

Graphs can be used to describe the relation between scientists who have collaborated and not, with a *co-authorship graph*. A co-authorship graph is a graph where the set of vertices is a given set of scientists, and if two scientists have collaborated at any given time, there should be an edge connecting their vertices. This type of graphs can be made from scientists in the same field of study, or across fields of studies [BK02].

#### Erdős number

The Hungarian mathematician Paul Erdős was well known for being very productive with his publishing and he did collaborate with a lot of scientists. This gave rise to the *Erdős number* (EN). A person who has collaborated with Erdős would have EN = 1, while a person who has collaborated with someone who has collaborated with Erdős would have EN = 2, and so on. Paul Erdős himself has EN = 0. This phenomenon can be described with a co-authorship graph centering around Paul Erdős. Every scientist who had collaborated with him would be given the Erdős number 1 and would be one edge away from Erdős in the graph [Wik23a].

There exist calculators that compute the collaboration distance between any two published scientists, and hence also the Erdős number. My supervisor, Geir Dahl, has collaborated with Richard A. Buraldi, who has collaborated with Noga Alon, who has collaborated with Paul Erdős. This means that Geir Dahl has the Erdős number EN = 3 [Bar23].

Chapter 5

# The Graham-Pollak Theorem

From Chapter 2 it is known that there exists several ways to decompose a graph. In this section, we want to look at a special type of decomposition, namely complete bipartite decomposition. What is the smallest possible number of subgraphs needed to decompose the complete graph $K_n$ into complete bipartite subgraphs? The Graham-Pollak theorem states that the complete graph $K_n$ needs at least $n-1$ subgraphs for the decomposition to be complete bipartite. The Graham-Pollak theorem was first published by Ronald Graham and Henry O. Pollak in 1971 and 1972 and gives a beautiful result in graph theory [Wik23b].

We start by restating the Graham-Pollak theorem.

Theorem 1.1.1 (The Graham-Pollak Theorem, [AZ18, p. 79]). *If the complete graph $K_n$ is decomposed into complete bipartite subgraphs $H_1$, $H_2$, ... , $H_m$, then $m \geq n-1$.*

The easiest way to obtain the optimal decomposition is by ordering the vertices, and for every vertex make a star such that the vertex is adjacent to every latter vertex in the ordering. If this is done with the $n-1$ first vertices, it is obtained as few complete bipartite subgraphs as possible. There are other ways to also make $n-1$ complete bipartite subgraphs, but this way will always make sure the fewest subgraphs is obtained[Wik23b].

Example 5.0.1. An example of a decomposition of the complete graph $K_4$ into complete bipartite subgraphs, using the approach described in the paragraph above.

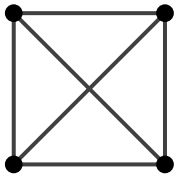


Figure 5.1: The complete graph $K_4$.

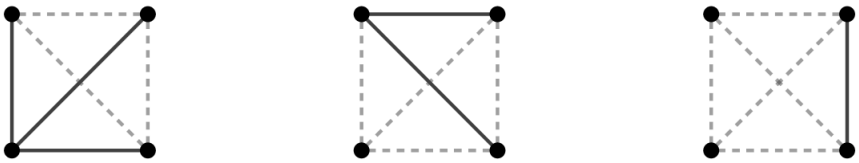


Figure 5.2: A complete bipartite decomposition of $K_4$.

## 5.1 The Proof by Tverberg of the Graham-Pollak Theorem

The Graham-Pollak theorem, Theorem 1.1.1, has an elegant proof made by the Norwegian mathematician Helge Tverberg, using linear algebra.

*Tverberg's proof, [AZ18, p. 80].* Let the vertex set of $K_n$ be $\{1, ..., n\}$, and let $L_j$, $R_j$ be the defining vertex sets of the complete bipartite graph $H_j$, $j = 1, ..., m$. To every vertex $i$ we associate a variable $x_i$. Since $H_1, ..., H_m$ decompose $K_n$, we find

$$\sum_{i<j} x_i x_j = \sum_{k=1}^{m} \Big( \sum_{a \in L_k} x_a \cdot \sum_{b \in R_k} x_b \Big) \tag{5.1}$$

Now suppose the theorem is false, $m < n - 1$. Then the system of linear equations

$$\begin{aligned} x_1 + ... + x_n &= 0, \\ \sum_{a \in L_k} x_a &= 0 \qquad (k = 1, ..., m) \end{aligned}$$

has fewer equations than variables, hence there exists a nontrivial solution $c_1, ..., c_n$. From (5.1) we infer

$$\sum_{i<j} c_i c_j = 0.$$

But this implies

$$0 \;=\; (c_1 + ... + c_n)^2 \;=\; \sum_{i=1}^{n} c_i^2 \;+\; 2 \sum_{i<j} c_i c_j \;=\; \sum_{i=1}^{n} c_i^2 \;>\; 0,$$

we have a contradiction, and the proof is complete. ■

Chapter 6

# The $\gamma$-problem

The Graham-Pollak theorem states that the least possible number of complete bipartite graphs that the complete graph $K_n$ can be decomposed into is $n-1$. But what happens if the subgraphs do not need to be complete bipartite, but we are satisfied with bipartite subgraphs? We want to explore this change in the conditions of the theorem. What is the smallest possible number of bipartite subgraphs needed to decompose the complete graph $K_n$?

## 6.1 What is the $\gamma$-problem?

To approach this problem, we need to introduce some new definitions.

Definition 6.1.1. The size of a decomposition is the number of subgraphs in the decomposition.

Definition 6.1.2. $\gamma_n$ is the smallest possible number of bipartite subgraphs needed to decompose a complete graph $K_n$ for $n \in \mathbb{N}$.

The problem is to find the smallest possible number of bipartite subgraphs needed to decompose the complete graph $K_n$. In comparison to the problem in the Graham-Pollak theorem, it is now open for less strict decompositions. Hence, there will be more possible ways to decompose $K_n$. Because we now have more freedom when making the decomposition it is conceivable that the number of decomposition needed when the subgraphs are complete bipartite is greater than if the subgraphs need only be bipartite. Hence,

$$\gamma_n \leq n-1. \tag{6.1}$$

When trying to solve this problem it is natural to start by checking what will happen to the decomposition of $K_n$ when $n$ is small. We have tried out what will happen for $n = 3, ..., 10$ and possibly found a pattern that gives us $\gamma_n$. The complete graph is not of great interest if $n < 3$. $K_2$ is by definition a complete bipartite graph, so it can clearly be decomposed into only one subgraph, while $K_1$ only consists of one vertex.

| $n$ | $n-1$ | $\gamma_n$ |
|---|---|---|
| 3 | 2 | 2 |
| 4 | 3 | 2 |
| 5 | 4 | 3 |
| 6 | 5 | 3 |
| 7 | 6 | 4 |
| 8 | 7 | 4 |
| 9 | 8 | 5 |
| 10 | 9 | 5 |
| $\vdots$ | $\vdots$ | $\vdots$ |
| $n$ | $n-1$ | $\lceil \frac{n}{2} \rceil$ ? |

Table 6.1: A comparison of the number of bipartite subgraphs need to decompose a complete graph, depending on whether or not the subgraphs are complete bipartite.

Table 6.1 presents a comparison of the number of subgraphs necessary to decompose the complete graph $K_n$, when the subgraphs need to be complete bipartite, $n-1$, and only bipartite, $\gamma_n$. To find these numbers we have made an algorithm where as many vertices as possible are split in each subgraph. If $n$ is an even number, I find as many unique partitions of the edges such that $|I| = |J| = n/2$. If $n$ is an odd number, it is obviously not possible to make partitions of equal size if all vertices should be included. Instead one can make as many unique partitions with $|I| = \lceil n/2 \rceil$ and $|J| = \lfloor n/2 \rfloor$ as possible. For each subgraph, every edge not covered in any former subgraphs is included. This makes sure that each vertex is in different sets at least once and that every edge is included.

Conjecture 6.1.3. *The smallest size of a complete bipartite decomposition of the complete graph $K_n$ is n divided by 2 and rounded up to the nearest whole number, hence*

$$\gamma_n = \left\lceil \frac{n}{2} \right\rceil . \tag{6.2}$$

In a decomposition, it is required that each edge only appear once. Each subgraph can be described with

$$S_i = (I_i, J_i, F_i) \qquad i = 1, ..., s, \tag{6.3}$$

where $I$, $J$ is the disjoint sets of vertices, $s$ is the number of subgraphs and $F$ tells which edges is contained in the subgraph. Each subgraph contains only edges between vertices in $I$ and $J$.

In the following sections, Conjecture 6.1.3 is proven for $n = 4$ and $n = 5$ and it is made an outline of a proof for the general complete graph $K_n$.

## 6.2 The $\gamma$-problem on $K_4$

We start by proving that Conjecture 6.1.3 holds for $n = 4$. In Figure 6.2 the complete graph $K_4$ is decomposed into as few bipartite subgraphs that we can manage. But is this indeed the smallest number of subgraphs in this type of decomposition? We want to show that this is true and hence $\gamma_4 = 2$.

We start with an example showing a possible complete bipartite decomposition of $K_4$.

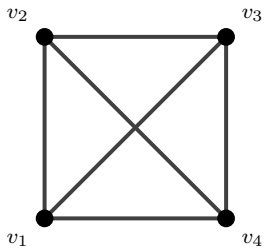



Figure 6.1: The complete graph $K_4$.

A bipartite decomposition of $K_4$ from Figure 6.1 is shown in Figure 6.2.

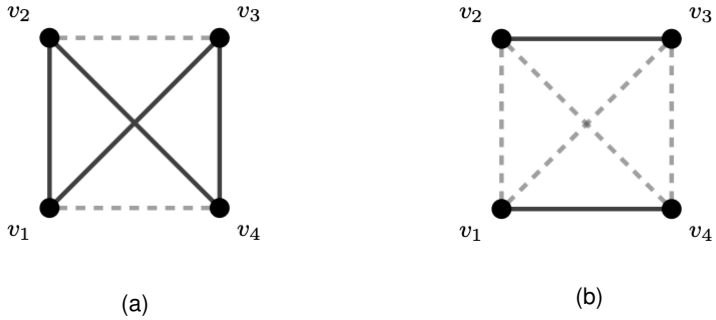



Figure 6.2: Decomposition of $K_4$ into bipartite subgraphs, (a) $S_1$ and (b) $S_2$.

Using the notation from (6.3) we can describe the two subgraphs of $K_4$ seen in Figure 6.2 with

$$\begin{aligned} S_1: &\quad I_1 = \{v_1, v_4\}, \quad J_1 = \{v_2, v_3\}, \quad F_1 = \{v_1v_2, v_1v_3, db, dc\} \\ S_2: &\quad I_2 = \{v_1, v_2\}, \quad J_2 = \{v_4, v_3\}, \quad F_2 = \{v_1v_4, v_2v_3\}. \end{aligned}$$

Because Figure 6.2 shows that there is possible to decompose $K_4$ into as little as two bipartite subgraphs, we know that $\gamma_4 \leq 2$. We already know from The Graham-Pollak theorem that $n - 1 = 4 - 1 = 3$. Hence, $\gamma_4 \leq 3$. This is consistent with (6.1).

### 6.2.1 Proof that $\gamma_4 = 2$

In Figure 6.2 $K_4$ is successfully decomposed into two bipartite subgraphs. I need to show that is not possible to decompose $K_4$ into only one bipartite subgraph.

Proposition 6.2.1. *The smallest size of a complete bipartite decomposition of $K_4$ is 2, hence*

$$\gamma_4 = \left\lceil \frac{4}{2} \right\rceil = 2.$$

**Observation 6.2.2.** *When trying to make only one bipartite subgraph of $K_4$, we can disregard bipartitions where $I \cup J \neq V(K_4)$. This is because we want to decompose $K_4$ into only one subgraph and for that one subgraph to be $K_4$ we need to include all vertices in $V(K_4)$.*

*Proof.* If we want to decompose a graph on four vertices into only one subgraph, we can make bipartitions such that we have either $|I|= 2$ and $|J|= 2$, or $|I|= 1$ and $|J|= 3$.

With these sets of vertices, we can make the following complete bipartite graphs.

$$S_1: \quad I_1 = \{v_1, v_2\}, \quad J_1 = \{v_3, v_4\}, \quad F_1 = \{v_1v_3, v_1v_4, v_2v_3, v_2v_4\} \tag{6.4}$$

Here the edges $\{v_1v_2, v_3v_4\}$ are not included, and therefore it is not a decomposition of $K_4$. The other possible distribution is as follows,

$$S_1: \quad I_1 = \{v_1\}, \quad J_1 = \{v_2, v_3, v_4\}, \quad F_1 = \{v_1v_2, v_1v_3, v_1v_4\} \tag{6.5}$$

Here the edges $\{v_2v_3, v_2v_4, v_3v_4\}$ are not included, and therefore it is not a decomposition of $K_4$.

Hence, it will not be possible to decompose $K_4$ into less than two bipartite subgraphs and we have $\gamma_4 = 2$.

■

## 6.3 The $\gamma$-problem on $K_5$

We continue to show that Conjecture 6.1.3 holds for $K_5$ also. In Figure 6.4 the complete graph $K_5$ is decomposed into as few bipartite subgraphs that we can manage. But is this indeed the smallest number of subgraphs in this type of decomposition? We want to show that this is true, hence $\gamma_5 = 3$.

We start by showing an example of a complete bipartite decomposition of $K_5$.

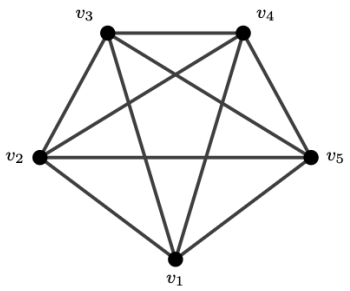



Figure 6.3: The complete graph $K_5$.

A complete bipartite decomposition of $K_5$, in Figure 6.3 is shown in Figure 6.4.

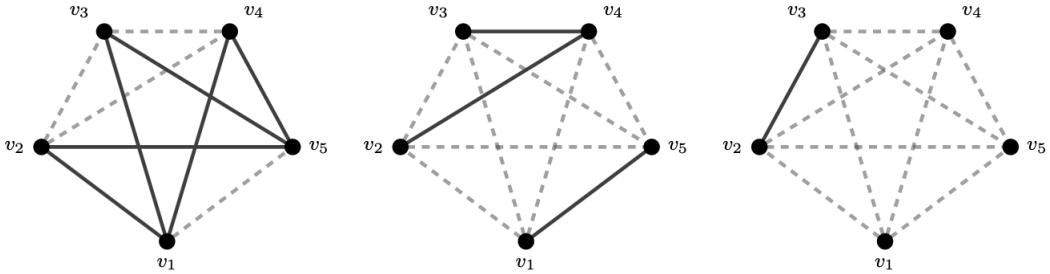



Figure 6.4: A bipartite decomposition of $K_5$.

Because Figure 6.4 shows that there is possible to decompose $K_4$ into as little as two bipartite subgraphs, we know that $\gamma_4 \leq 2$. We already know from The Graham-Pollak theorem that $n - 1 = 5 - 1 = 4$. Hence, $\gamma_5 \leq 4$. This is still consistent with (6.1).

There are many ways to split the vertices in $K_5$ into distinct sets. We choose to always have $I \cup J = V(K_5)$. This is because the important part is which edges, not vertices, that are included in the bipartite subgraphs. Hence, a subgraph with $|I| = 1$ and $|J| = 1, 2, 3$ can give rise to the same subgraphs as if $|I| = 1$ and $|J| = 4$. Similarly, we get the same possible subgraphs if $|I| = 2$ and $|J| = 3$, as if $|I| = 2$ and $|J| = 1, 2$. It only depends on which edges we include in the subgraph.

Hence, there are two different ways to split the set of vertices of $K_5$, if we want $I \cup J = V(K_5)$,

- $|I| = 1$, $|J| = 4$
- $|I| = 2$, $|J| = 3$.

Thus, we can decompose $K_5$ into two subgraphs with the following size of partitions,

- $S_1$: $|I_1| = 1$, $|J_1| = 4$ and $S_2$: $|I_2| = 1$, $|J_2| = 4$
- $S_1$: $|I_1| = 1$, $|J_1| = 4$ and $S_2$: $|I_2| = 2$, $|J_2| = 3$
- $S_1$: $|I_1| = 2$, $|J_1| = 3$ and $S_2$: $|I_2| = 2$, $|J_2| = 3$.

### 6.3.1 Proof that $\gamma_5 = 3$

In Figure 6.4 we have successfully decomposed $K_5$ into three bipartite subgraphs. We need to show that it is not possible to decompose $K_5$ into any less than three bipartite subgraph.

Proposition 6.3.1. *The smallest size of a complete bipartite decomposition of $K_5$ is 3, hence*

$$\gamma_5 = \left\lceil \frac{5}{2} \right\rceil = 3.$$

*Proof.* To prove this we start by parting the first subgraph, $S_1$ into the sets

$$I_1 = \{v_1\}, \quad J_1 = \{v_2, v_3, v_4, v_5\}, \quad F_1 = \{v_1v_2, v_1v_3, v_1v_4, v_1v_5\} \tag{6.6}$$

We then have two options when making the bipartite subgraph $S_2$.

$$I_2 = \{v_2\}, \quad J_2 = \{v_1, v_3, v_4, v_5\}, \quad F_2 = \{v_2v_3, v_2v_4, v_2v_5\} \tag{6.7}$$

By using this distribution of vertices it will never be possible to separate the vertices $\{v_3, v_4, v_5\}$ and there will not be possible to include the edges $\{v_3v_4, v_3v_5, v_4v_5\}$. Hence, we do not get a decomposition of $K_5$.

But what happens if $S_2$ has the distribution $|I_2| = 2$ and $|J_2| = 3$ instead? We have to look at what happens if $v_n \in I_1$ is in either $v_n \in I_2$ or $v_n \in J_2$.

First, $v_n \in I_2$, in this case, $v_1 \in I_2$.

$$I_2 = \{v_1, v_2\}, \quad J_2 = \{v_3, v_4, v_5\}, \quad F_2 = \{v_2v_3, v_2v_4, v_2v_5\}. \tag{6.8}$$

As in (6.7) we will never get the edges $\{v_2v_3, v_2v_4, v_2v_5\}$ and we will not get a decomposition of $K_5$.

Then, $v_n \in J_2$, in this case, $v_1 \in J_2$,

$$I_2 = \{v_4, v_5\}, \quad J_2 = \{v_1, v_2, v_3\}, \quad F_2 = \{v_2v_4, v_2v_5, v_3v_5, v_3v_5\}. \tag{6.9}$$

In (6.9) the edges $\{v_2v_3, v_4v_5\}$ will never be included and it will not be a decomposition of $K_5$.

Hence, we will not get a decomposition of $K_5$ with only two subgraphs that are bipartite if one of the subgraphs has the distribution $|I_1| = 1$ and $|J_1| = 4$.

The last possible option is for both of the subgraphs to have the distribution $|I| = 2$ and $|J| = 3$.

$$I_1 = \{v_1, v_2\}, \quad J_1 = \{v_3, v_4, v_5\}, \quad F_1 = \{v_1v_3, v_1v_4, v_1v_5, v_2v_3, v_2v_2, v_2v_5\}. \tag{6.10}$$

for it to be a decomposition of $K_5$ into bipartite subgraphs we need to separate the elements in $I_1$ and $J_2$ respectively, as shown in 6.11,

$$v_1/v_2, \quad v_3/v_4, \quad v_3/v_5, \quad v_4/v_5 \tag{6.11}$$

If $v_3$ needs to be separated from both $v_4$ and $v_5$, we must, for instance, have $v_3 \in I_2$ and $v_4, v_5 \in J_2$. But we need also separate $v_4$ and $v_5$, which means that we must have $v_4 \in I_2$ and $v_5 \in J_2$. Hence, it is impossible to fulfil all the requirements from 6.11.

Hence, it is not possible to decompose $K_5$ into less than three bipartite subgraphs.

■

## 6.4 The $\gamma$-problem on $K_n$

In the previous chapter Conjecture 6.1.3 is proven for $n = 4$ and $n = 5$. The natural next step is to try and prove it for every $n \in \mathbb{N}$. This is a rather difficult problem to solve. It will therefore in this section be presented some explanations of why the algorithm used to find the values in Table 6.1 is an efficient way to find the smallest possible bipartite decomposition of the complete graph $K_n$.

### 6.4.1 Ideas for a proof of Conjecture 6.1.3

To make a decomposition of $K_n$ where every subgraph is a bipartite graph, we need to make sure that each pair of vertices $(v_i, v_j)$ is in separate sets at least once. If this is not the case, we will never get to include the edge $v_iv_j$, and if one edge is missing we do not have a decomposition. For each subgraph, we can make sure that at least two distinct pairs of vertices are in each set of vertices. I want also to always include every edge, not included in any former subgraphs, in each subgraph.

#### The optimal size of sets of each subgraph

If we want to have as few decompositions as possible, we want to include as many edges as possible in each subgraph. We can start by making a subgraph where each subset has as similar size as possible, hence $|I| = |J|$, or $|I| = |J| - 1$. This gives the most possible edges. We can prove this with some short calculations. The number of possible edges in the cut is the size of the sets multiplied. We need to look at separate cases, one where $n$ is even, and one where $n$ is odd.

If $n$ is even, we can split the graphs into equal sized sets, $|I| = |J|$. Then we get

$$|I| \cdot |J| > (|I| + 1) \cdot (|J| - 1) > (|I| + 2) \cdot (|J| - 2)...$$

Hence, the sets of equal size give rise to more possible edges than sets of not equal size.

If $n$ instead is odd, we can split the sets such that $|I| = |J| - 1$. Then we get

$$(|J|) \cdot (|J| - 1) > (|J| + 1) \cdot (|J| - 2) > (|J| + 2) \cdot (|J| - 3)...$$

Hence, the smaller the difference in the size of the sets, the more edges can be included in the subgraph. This holds when $n$ is both even and odd.

As mentioned earlier, we need every pair of vertices to be in different sets at least once to get every edge included somehow in the decomposition. If every subgraph has a distribution where $|I|$ and $|J|$ are as similar as possible, we get the greatest possible number of edges to choose from in each subgraph, and also the greatest possible number of edges to choose from in total. Though it is important that two subgraphs do not have the same distribution, then we will only achieve the same subgraph. To split every pair of vertices at least once, we need to find how many unique ways we can split the vertices in the best possible way. Hence, how many times can we divide $n$ by 2?

### Visual explanation of Conjecture 6.1.3

My initial idea is to draw the graph $K_n$ and then use the visual representation of complete graphs to explain why the number of subgraphs is $\lceil n/2 \rceil$. Any graph can be drawn in a way such that the vertices are arranged in a circle, as in Figure 6.5. Using this way to draw a graph can be used to show how we should make subgraphs. Each straight red line splits the vertices in two sets. If this is done such that we get as many uniquq equal (or as equal as possible) subgraphs. It is a very intuitive way to explain that we have to have at least $\lceil n/2 \rceil$ subgraphs to decompose $K_n$ into bipartite subgraphs. This way makes sure that every pair of vertices is in different sets at least once, and hence every edge can be included and we have a decomposition.

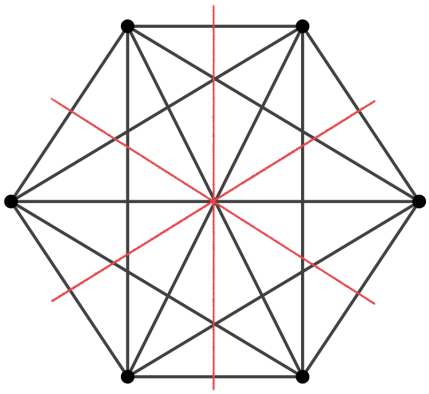


Figure 6.5: An example of how to make subgraph distributions to have each pair of vertices in different sets at least once.

Chapter 7

# The $\kappa$-problem

There are many decomposistions of graphs to explore. In the previous chapter we introduced the $\gamma$-problem, where we adjusted the problem from the Graham-Pollak theorem by looking at what would happen if the subgraphs no longer were complete bipartite, but the only condition was that the subgraphs were bipartite. What would happen if we wanted a complete bipartite decomposition of a graph, but the graph does not need to be the complete graph $K_n$?

## 7.1 What is the $\kappa$-problem?

It is difficult to find a general pattern for the number of subgraphs in a decomposition of an arbitrary graph. We can instead explore what would happen if we look at different classes of graphs. To do this we need to introduce a new definition.

Definition 7.1.1. $\kappa(G)$ is the smallest possible number of subgraphs needed to make a complete bipartite decomposition of the graph $G$.

We also repeat the definiton of the size of decomposition from Chapter 6.

Definition 6.1.1. The size of a decomposition is the number of subgraphs in the decomposition.

### Greedy Algorithms

A rather intuitive way to solve an optimisation problem is by using what we call a *greedy algorithm*. A greedy algorithm will try to find the optimal solution at each step in the algorithm. This will not necessarily result in the overall optimal solution for the problem. When trying to decompose a graph into as few subgraphs as possible, a greedy algorithm will always start by finding the largest possible subgraph [MKR16].

Example 7.1.2. *Find the maximum sum in Figure 7.1 by choosing a path on two edges staring with the vertex* 5. The greedy algorithm will try to find the best solution in each step, this will give us the red path and the sum $5 + 9 + 11 = 25$. That is clearly not the best possible solution, the blue path will give the sum $5 + 7 + 17 = 29$.

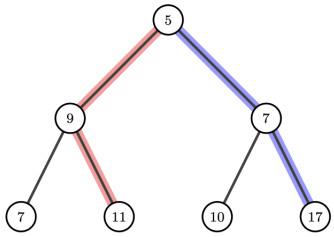



Figure 7.1: An example where a greedy algorithm does not find the optimal solution.

## 7.2 The $\kappa$-problem on paths

The first class of graphs to prove the $\kappa$-problem for is paths. We start by looking at a path and a possible complete bipartite decomposition.

Example 7.2.1. A complete bipartite decomposition of a path on four vertices.


Figure 7.2: A path on four vertices, $P_4$.

The path on four vertices shown in Figure 7.2 can be decomposed into the following complete bipartite subgraphs.


(a) A complete bipartite subgraph.

(b) A complete bipartite subgraph.

Figure 7.3: A complete bipartite decomposition of the path in Figure 7.2.

Hence, we have the subgraphs

$$\begin{aligned} S_1 : \quad & I_1 = \{v_1, v_3\}, \quad & J_1 = \{v_2\} \\ S_2 : \quad & I_2 = \{v_3\}, \quad & J_2 = \{v_4\}. \end{aligned}$$

Observation 7.2.2. *We need only take into account bipartite subgraphs that are connected. If the bipartite subgraph is not connected, there will always be missing an edge between a vertex in the partition sets, and the subgraph will clearly not be complete bipartite.*

Proposition 7.2.3. *A path on $n$ vertices $P_n$ can always be decomposed into $\lceil (n-1)/2 \rceil = \lceil e/2 \rceil$ complete bipartite subgraphs. Hence,*

$$\kappa(P_n) = \left\lceil \frac{n-1}{2} \right\rceil.$$

*Proof.* A connected subgraph of a path will always be a path, so to prove Equation (7.7) we need only show that it holds for any path.

A path on $n$ vertices $(v_1, v_2, ...., v_n)$ is such that any two vertices in a path are adjacent if and only if they are consecutive. Any path is bipartite, every other vertex is not adjacent and can be put in different bipartition sets. Hence, we have a bipartite graph.

A path on four vertices $(v_1, v_2, v_3, v_4)$ is not complete bipartite, because $v_1$ and $v_4$ are in opposite sets, but are not adjacent. For any path of length $> 4$, there will be $\geq 3$ vertices in either $I$ or $J$, or both. In a path the degree of any vertex is $\leq 2$, and can not possibly be adjacent to more than two vertices in its opposite set.

Hence, for a path to be complete bipartite we need to have no more than three vertices and two edges. In Figure 7.3a and Figure 7.11b we see the two different types of complete bipartite graphs possible to obtain in a complete bipartite decomposition of a path. We can only have subgraphs such that $|I| + |J| \leq 3$.

We know from Theorem 2.0.4 that any path on $v$ vertices has $e = v - 1$ edges. In addition, we know that a path can only be complete bipartite if it has no more than three vertices and two edges. Hence, the number of complete bipartite subgraphs is dependent on the number of edges and vertices.

To get the smallest possible number of subgraphs, we need as many graphs as possible with three vertices and two edges, as in Figure 7.3a. Therefore, we can check how many times two add up in the number of edges. If the number of edges is odd, we will get a remainder of one half, and therefore the decomposition will include one subgraph with only one edge.

Hence, the smallest possible number of subgraphs needed to decompose a path into complete bipartite subgraphs can be expressed as $\lceil \frac{e}{2} \rceil = \left\lceil \frac{n-1}{2} \right\rceil$,

$$\kappa(P_n) = \left\lceil \frac{n-1}{2} \right\rceil \tag{7.1}$$

■

Observation 7.2.4. *It follows that a path with an odd number of edges will give the same minimal number of subgraphs as the path on the following even number of vertices.*

## 7.3 The $\kappa$-problem on cycles

When trying to find a complete bipartite decomposition of a cycle $C_n$, it is soon discovered that this is quite similar to what happens when the graph we want to decompose is a path. We start with an example of a complete bipartite decomposition of $C_5$ and the special case $C_4$ before we proceed with a proposition and a proof.

Example 7.3.1. A cycle $C_5$, can have the following complete bipartite decomposition

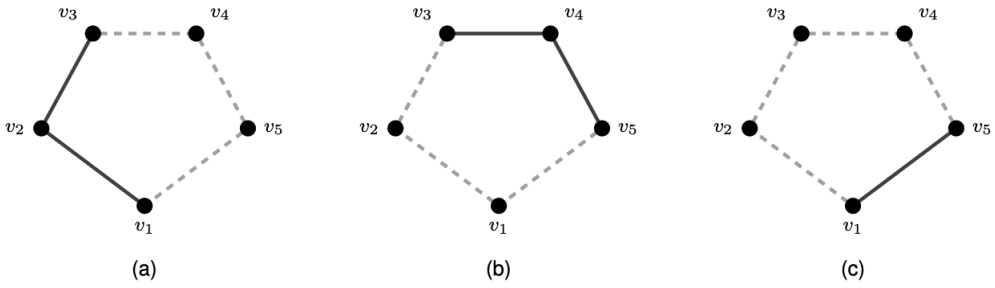



Figure 7.4: A complete bipartite decomposition of $C_5$.

This gives the following subgraphs, using the same notation as before

$$\begin{array}{lll} \text{(a)} \quad S_1: & I_1 = \{v_2\}, & J_1 = \{v_1, v_3\}, \\ \text{(b)} \quad S_2: & I_2 = \{v_4\}, & J_2 = \{v_3, v_5\} \\ \text{(c)} \quad S_3: & I_3 = \{v_1\}, & J_3 = \{v_5\}. \end{array}$$

Example 7.3.2. A cycle $C_4$ is indeed complete bipartite, as shown Figure 7.5.

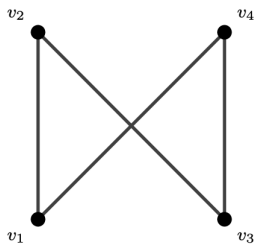



Figure 7.5: $C_4$ as an bipartite graph.

This is clearly complete bipartite with the partitions $I = \{v_2, v_4\}$ and $J = \{v_1, v_3\}$.

Proposition 7.3.3. *A cycle with $n > 4$ vertices $C_n$ has a complete bipartite decomposition of size $\lceil n/2 \rceil = \lceil e/2 \rceil$. Hence,*

$$\kappa(C_n) = \left\lceil \frac{n}{2} \right\rceil.$$

A cycle on $n$ vertices $(v_1, v_2, ..., v_n)$ is such that any two vertices are adjacent if and only if they are consecutive. The vertices are arranged in a cyclic sequence, hence, $v_n$ and $v_1$ are also consecutive and adjacent.

Observation 7.2.2 holds for this case also. This means that to decompose we need only take into account connected subgraphs. A connected subgraph of a cycle is either the whole cycle itself, or it is a path. Hence, we need only take into consideration subgraphs that are either paths or cycles.

*Proof.* From Theorem 2.0.6 we know that no cycle of odd length is bipartite. Hence, a cycle of length 3 is not complete bipartite. A cycle of length 4 is indeed complete

bipartite, as shown in Figure 7.5 and Proposition 7.3.3 does clearly not hold for the cycle in **??**.

For a cycle of length $> 4$, there will be $\geq 3$ vertices in either $I$ or $J$, or both. In a cycle, the degree of any vertex is 2, and can not possibly be adjacent to more than two vertices in its opposite set. Hence, no cycle of length $\geq 5$ is complete bipartite.

To find the smallest complete bipartite decomposition, we can proceed similarly as with paths. We can always decompose a cycle into paths of length $\leq 2$, and this is guaranteed complete bipartite. Thus, we can once more check how many times two adds up in the number of edges. If the number of edges is odd, we will get a remainder of one half, and therefore the decomposition will consist of one subgraph with only one edge.

Hence, the smallest possible number of subgraphs needed to decompose a cycle on $n > 4$ vertices into complete bipartite subgraphs is indeed $\lceil \frac{n}{2} \rceil = \lceil \frac{e}{2} \rceil$ and hence,

$$\kappa(C_n) = \left\lceil \frac{n}{2} \right\rceil .$$

■

## 7.4 The $\kappa$-problem on wheels

It is possible to expand the $\kappa$-problem even further and look at what happens if the graph we want to decompose is a wheel. This is a natural extension of the $\kappa$-problem on cycles.

In a wheel each edge $v_i v_{n+1}$ is called a spoke and the cycle made up by the $n$ first vertices is called the outer cycle of the wheel.

To decompose the wheel $W_n$ it is tempting to use greedy a star-decomposition and start by choosing the vertex with $d_W(v) \geq 3$. This gives a star with $I = \{v_{n+1}\}$ and $I = \{v_1, v_2, ..., v_n\}$. We are then left with a cycle that we know from Proposition 7.3.3 can be decomposed into $\lceil n/2 \rceil$ complete bipartite subgraphs. Hence, using this star-decomposition we will get $\lceil n/2 \rceil + 1$ complete bipartite subgraphs.

Proposition 7.4.1. *A wheel on $n+1$ vertices $W_n$ can be decomposed into $\lceil n/2 \rceil + 1$ complete bipartite subgraphs. Hence,*

$$\kappa(W_n) = \left\lceil \frac{n}{2} \right\rceil + 1. \tag{7.2}$$

But is this indeed the least possible subgraphs needed to decompose a wheel $W_n$ into complete bipartite subgraphs? We begin by looking at an example of a complete bipartite decomposition of $W_5$ and $W_6$.

Example 7.4.2. If we use a decomposition where we start by making a subgraph using the star in the wheel, and then decompose the outer cycle as in Proposition 7.3.3 to find a complete bipartite decomposition of the wheel $W_5$, we get

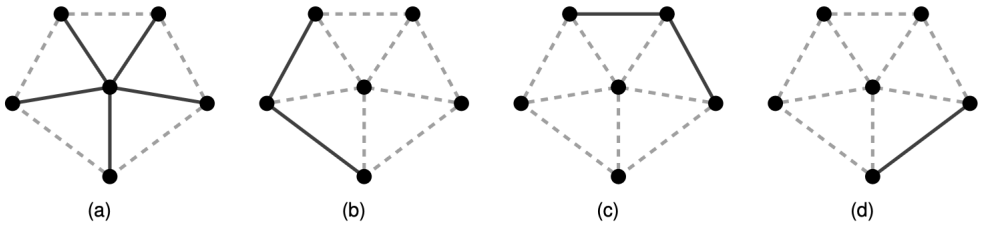



Figure 7.6: A complete bipartite decomposition of the wheel $W_5$.

A complete bipartite decomposition of the wheel $W_6$, using the same type of decomposition, gives us

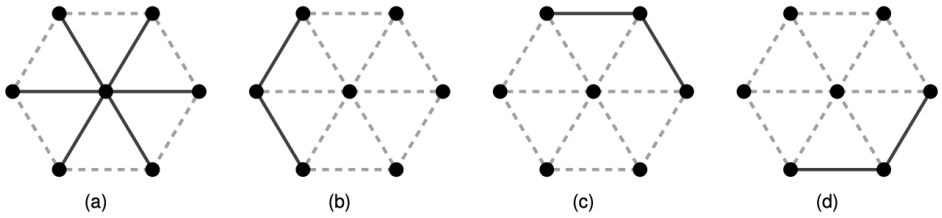



Figure 7.7: A complete bipartite decomposition of the wheel $W_6$.

Hence, both $W_5$ and $W_6$ can at least be decomposed into four complete bipartite subgraphs. This results in $\kappa(W_5) \leq 4$ and $\kappa(W_6) \leq 4$, which is consistent with Proposition 7.4.1.

### 7.4.1 Background for proof

Observation 7.2.2 is still valid for this case. Thus, to decompose the wheel into complete bipartite subgraphs, we need only consider connected subgraphs.

#### Possible complete bipartite subgraphs of a wheel

In a wheel $W_n$, $n > 2$, we know that every vertex $v_i$, $i = 1, ..., n$, has the property $d_W(v_i) = 3$, and they are adjacent to $\{v_{i-1}, v_{i+1}, v_{n+1}\}$, while the vertex $v_{n+1}$ has the property $d_W(v_{n+1}) \geq 3$ and it is adjacent to every vertex $v_i$. This means that no two vertices $v_i$ are adjacent to the exact same set of vertices. Hence, to get a complete bipartite subgraph we can never have two vertices $v_i$ in the same partition $J$, unless $I = \{v_{n+1}\}$. If we want the decomposition to consist of only complete bipartite subgraph, the following subgraphs are the possible options

- Figure 7.8a: A path on two vertices is complete bipartite.
- Figure 7.8b, Figure 7.8c: A path on three vertices is complete bipartite.
- Figure 7.8d: A cycle $C_4$ is complete bipartite with the sets $I = \{v_i, v_{n+1}\}$ and $J = \{v_{i-1}, v_{i+1}\}$.

- Figure 7.8e: The graph $T_3$ is complete bipartite with $I = \{v_i\}$ and $J = \{v_{i-1}, v_{i+1}, v_{n+1}\}$.
- Figure 7.8f: A star with $I = \{v_{n+1}\}$ is complete bipartite with either $J = \{v_1, ..., v_n\}$ or if $J$ consists only of the vertices not already adjacent to $v_{n+1}$ in another subgraph.

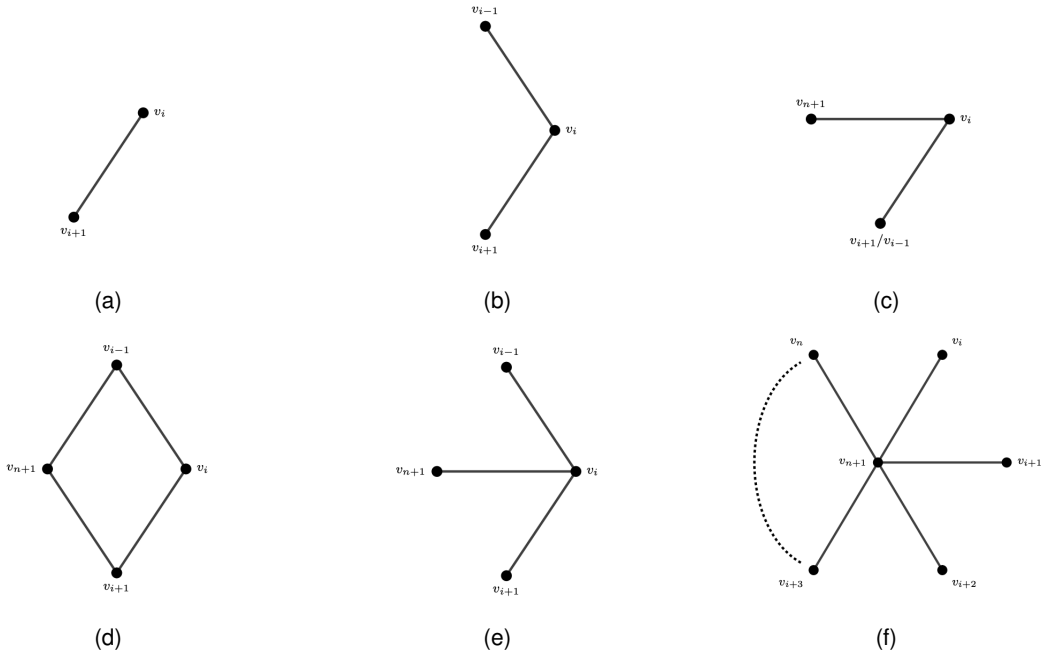



Figure 7.8: The possible complete bipartite subgraphs of the wheel $W_n$.

### 7.4.2 Proof $\kappa(W_n) = \lceil n/2 \rceil$

*Proof.* Every possible complete bipartite subgraph has a maximum of two edges in the outer cycle of the wheel. To cover every edge in the outer cycle with as few subgraphs as possible, we need to have as many subgraphs as possible with two edges in the cycle. If we want to cover these as efficiently as possible this is done with the same amount of complete bipartite subgraphs as with a cycle, $\lceil n/2 \rceil$ subgraphs.

Unless there is possible to find a decomposition of size $\lceil n/2 \rceil$ that decomposes the wheel in such a way that each spoke and each edge in the outer cycle is covered, the star-decomposition will give us as few subgraphs as possible.

The subgraphs $P_1$ will only cover one edge in the outer cycle and no spokes. So if this subgraph is included, it needs a star, and it needs as many of $P_{2_1}$ as possible. This will require at least $\lceil n/2 \rceil + 1$ subgraphs.

The subgraph $P_{2_2}$ will cover every spoke, but only one edge in the outer cycle. That will result in $n$ subgraphs, which are greater than $\lceil n/2 \rceil + 1$ as long as $n > 2$.

The subgraph $C_4$ has the subsets $I = \{v_i, v_{n+1}\}$ and $J = \{v_{i-1}, v_{i+1}\}$. It will not contain the spoke $v_i v_{n+1}$, and only two edges in the outer circle will be covered. Hence, if $C_4$ is one subgraph, we will always have at least one spoke left to cover, and the decomposition will at least be of size $\lceil n/2 \rceil + 1$.

The subgraph $T_3$ will always consist of one spoke and two edges in the outer cycle. If $T_3$ is included in the decomposition, there must be one or two graphs to cover up the spokes $v_{n+1}v_{i-1}$ and $v_{n+1}v_{i+1}$. This can be done with either a star or the graph $P_{2_2}$. The star will have subsets $I = \{v_{n+1}\}$ and $J = \{v_{i-1}, v_{i+1}\}$. This will make sure that we get at least $\lceil n/2 \rceil + 1$ complete bipartite subgraphs. If $P_{2_2}$ is used, the spokes will be covered, but this will also include two extra subgraphs in the outer cycle. This will result in at least $\lceil n/2 \rceil + 1$ complete bipartite subgraphs.

Hence, a wheel $W_n$ can never be decomposed into less than $\lceil n/2 \rceil + 1$ complete bipartite subgraphs and

$$\kappa(W_n) = \left\lceil \frac{n}{2} \right\rceil + 1.$$

■

## 7.5 The $\kappa$-problem on a generalised star

An extension to this problem is to see what happens if we look at a new type of star we can call a *generalised star*. The vertex with $d_S(v) > 2$ we will call the *star center*, while the connected edges are called *star arms*. A generalised star will still have only one vertex of $d_S(v) > 2$, but each star arm does not need to be of length 1. Each star arm can be a path of arbitrary length. An *even generalised star* consists only of star arms of even length, and an *odd generalised star* consists only of star arms of odd length.

It is reasonable to think that one should always start the decomposition by choosing the vertex of degree $> 1$, the star centre, and make a complete bipartite graph. Hence, a greedy algorithm would immediately look like an effective way to find the smallest possible number of complete bipartite subgraphs.

There are a few things to notice when starting this problem. Because we want the subgraphs to be complete bipartite, vertices in different star arms will never be adjacent, and it is never possible to have any vertices from different star arms in the same subgraph. This is clearly not true for the vertices one edge away from the star centre. There are also two different methods for decomposing the star into subgraphs. We can either start with the star centre and then proceed by decomposing each star arm, or we can decompose each star arm for themselves as paths.

### 7.5.1 Notation for generalised star

To approach this problem we can start by finding a notation for each arm of the generalised star.

$$\begin{aligned} V_1(G) &= \{S, v_{11}, ..., v_{1k_1}\} \\ V_2(G) &= \{S, v_{21}, ..., v_{2k_2}\} \\ &\vdots \\ V_n(G) &= \{S, v_{n1}, ..., v_{nk_n}\} \end{aligned} \tag{7.3}$$

where $n$ is the number of star arms, $k_n$ is the number of vertices in each star arm respectively, in addition to the star centre $S$. Because each star arm is a path, and we

only take into account connected graphs, it is obvious which edges that are included in each star arm. Hence, we can write the whole star as

$$V(G) = \bigcup_{i=1}^{n} V_i(G) \tag{7.4}$$

To find the optimal way to find a complete bipartite decomposition of a generalised star that is as small as possible, it is needed some more parts of the star.

A *central star* is a star, hence also bipartite, where the star centre is in a set $I$, and every vertex adjacent to $S$ in $J$, this gives the following subgraph,

$$\text{Central star:} \qquad I = \{S\}, \qquad J = \{v_{11}, v_{21}, ..., v_{n1}\}. \tag{7.5}$$

If the central star is used in a decomposition, the part of the star arms that is left is paths one edge shorter than the original star arms. These can be called *reduced star arms* and are denoted as follows

$$\begin{aligned} R_1(G) &= V_1(G) \setminus \{S\} = \{v_{1_1}, ..., v_{1k_1}\}, \\ R_2(G) &= V_2(G) \setminus \{S\} = \{v_{2_1}, ..., v_{2k_2}\}, \\ &\vdots \\ R_n(G) &= V_n(G) \setminus \{S\} = \{v_{n_1}, ..., v_{nk_n}\}. \end{aligned}$$

Hence, $|R_i(G)| = |V_i(G)| - 1$. If $|V_i(G)|$, was even, then $|R_i(G)|$ is odd, and vice versa.

### 7.5.2 Comparison of star- and path-decomposition

As stated in the introduction to this chapter, there are two ways to begin a complete bipartite decomposition of a generalised star. We can distinguish between a greedy and a non-greedy algorithm. The *star-algorithm* starts greedy and chooses the biggest possible subgraph, which is the central star. Then it decomposes the reduced star arms as paths. The *path-algorithm* is non-greedy and decomposes each star arm as a path. Using the proof from Section 7.2 it is possible to find how many complete bipartite subgraphs each of the algorithms gives. Further follows the total number of subgraphs we get from using both algorithms.

#### Star-decomposition

Each reduced star arm, $R_i(G)$, is a path on $k_n$ vertices, and

$$\kappa(R_i(G)) = \left\lceil \frac{k_i - 1}{2} \right\rceil$$

as proven in Section 7.2. It follows that if we decompose using the star-decomposition, we get the total number of complete bipartite subgraphs to be

$$\kappa\left(\sum_{i=1}^{n} R_i(G)\right) + 1 = \sum_{i=1}^{n} \left\lceil \frac{k_i - 1}{2} \right\rceil + 1. \tag{7.6}$$

This is the sum of the decompositions on each reduced star arm plus the one extra subgraphs we get from the central star. Hence,

$$\kappa_{star}(G_n) = \sum_{i=1}^{n} \left\lceil \frac{k_i - 1}{2} \right\rceil + 1.$$

### Path-decomposition

Each star arm, $V_i(G)$, is a path on $k_n + 1$ vertices, and can always give rise to

$$\kappa(V_i(G)) = \left\lceil \frac{(k_i + 1) - 1}{2} \right\rceil = \left\lceil \frac{k_i}{2} \right\rceil$$

complete bipartite subgraphs, as proven in Section 7.2. This means that if we decompose each star arm as a path, the total number of complete bipartite subgraphs will be

$$\kappa\left(\sum_{i=1}^{n} V_i(G)\right) = \sum_{i=1}^{n} \left\lceil \frac{k_i}{2} \right\rceil. \tag{7.7}$$

and hence,

$$\kappa_{path}(G_n) = \sum_{i=1}^{n} \left\lceil \frac{k_i}{2} \right\rceil. \tag{7.8}$$

We need to compare (7.7) and (7.6) to get closer to solving the $\kappa$-problem for generalised stars. Which of the decompositions gives us the fewest complete bipartite subgraphs? Both of the expressions, (7.7) and (7.6) are obviously dependent on whether $k_i$ is an even or odd number. There are three different types of generalised stars. Every star arm can be of even length, as in Figure 7.9b, though not necessarily of equal length, every star arm can be of odd length, as in Figure 7.9a, though still not necessarily of equal length, or we can have a combination of star arms that are of even and odd length, as in Figure 7.9c. We look at each case individually and decide which decomposition is the best to use when.

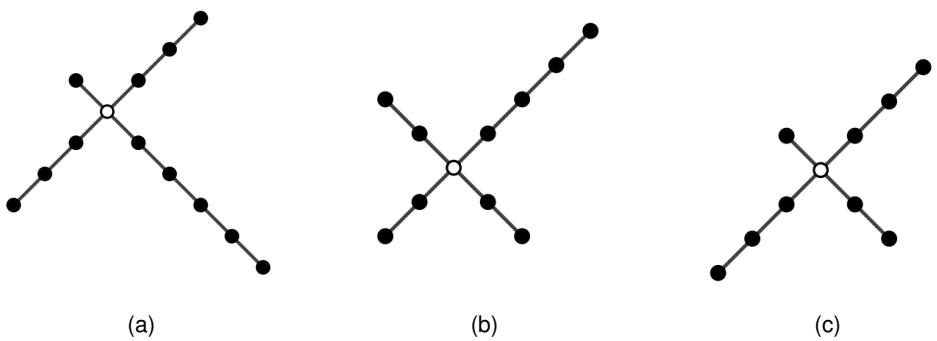



Figure 7.9: The three different types of generalised stars, (a) odd generalised star, (b) even generalised star, (c) general generalised star.

### Every $k_i$ is even

We can start by assuming every $k_i$ is even, then

$$\left\lceil \frac{k_i}{2} \right\rceil = \left\lceil \frac{k_i - 1}{2} \right\rceil .$$

This means that

$$\kappa_{path}(G_n) = \sum_{i=1}^{n} \left\lceil \frac{k_i}{2} \right\rceil = \sum_{i=1}^{n} \left\lceil \frac{k_i - 1}{2} \right\rceil . \tag{7.9}$$

If we now compare (7.9) and (7.6) we see that

$$\kappa_{path}(G_n) < \kappa_{star}(G_n)$$

Hence, it will pay off to use the non-greedy path-decomposition if every path is of even length.

Proposition 7.5.1. *The optimal solution will be the path-decomposition if every $k_i$ is even, hence*

$$\kappa(G_n) = \sum_{i=1}^{n} \left\lceil \frac{k_i}{2} \right\rceil .$$

### Every $k_i$ is odd

If we instead assume that every $k_i$ is odd, then

$$\left\lceil \frac{k_i}{2} \right\rceil = \left\lceil \frac{k_i - 1}{2} \right\rceil + 1.$$

This means that

$$\kappa_{path}(G_n) = \sum_{i=1}^{n} \left\lceil \frac{k_i}{2} \right\rceil = \sum_{i=1}^{n} \left( \left\lceil \frac{k_i - 1}{2} \right\rceil + 1 \right) = \sum_{i=1}^{n} \left\lceil \frac{k_i - 1}{2} \right\rceil + n. \tag{7.10}$$

If we now compare (7.10) and (7.6) we see that as long as $n > 1$,

$$\kappa_{path}(G_n) > \kappa_{star}(G_n)$$

Hence, it will pay off to use the greedy star-decomposition if every path is of odd length.

Proposition 7.5.2. *The optimal solution will be the star-decomposition if every $k_i$ is odd, hence*

$$\kappa(G_n) = \sum_{i=1}^{n} \left\lceil \frac{k_i - 1}{2} \right\rceil + 1.$$

We see clearly that which decomposition to use is dependent on whether the star arms are of even or odd lengths. The final possible generalised star has a combination of even and odd lengths of the star arms.

### Combination of even and odd $k_i$'s

If there is a combination of even and odd length arms the problem is a bit more complicated. To approach this problem we need to distinguish between the star arms of respectively odd and even length. In a generalised star, we have $n$ arms, with $m$ arms of odd length and $l$ arms of even length, such that $m + l = n$ and $l = m + 1, ..., n$.

If we want to decompose the graph using the non-greedy path-decomposition, we know from (7.10) and (7.9) how many subgraphs the $m$ odd arms and $l$ even arms can be decomposed into. This results in the following

$$\kappa_{path}(G) = \underbrace{\sum_{i=1}^{m} \left\lceil \frac{k_i - 1}{2} \right\rceil + m}_{\text{odd star arms}} + \underbrace{\sum_{j=m+1}^{n} \left\lceil \frac{k_j - 1}{2} \right\rceil}_{\text{even star arms}}. \tag{7.11}$$

If we instead use the greedy star-decomposition, we can use what we found in (7.6). We can decompose each reduced star arm in addition to the extra subgraph that is the central star. We know from (7.7) how many subgraphs we get when we decompose the reduced star arms. This results in the following:

$$\kappa_{star}(G) = \underbrace{\sum_{i=1}^{m} \left\lceil \frac{k_i - 1}{2} \right\rceil}_{\text{odd reduced star arms}} + \underbrace{\sum_{j=m+1}^{n} \left\lceil \frac{k_j - 1}{2} \right\rceil}_{\text{even reduced star arms}} + \underbrace{1.}_{\text{central star}} \tag{7.12}$$

Hence, when $m \geq 1$ it will pay off to use the greedy star-decomposition.

Proposition 7.5.3. *The optimal solution will be the star-decomposition if $m \geq 1$, hence*

$$\kappa(G_n) = \sum_{i=1}^{m} \left\lceil \frac{k_i - 1}{2} \right\rceil + \sum_{j=m+1}^{n} \left\lceil \frac{k_j - 1}{2} \right\rceil + 1.$$

Thus, as long as the generalised star has at least one star arm of odd length, the star-algorithm will always make the smallest decomposition. The path-algorithm is only efficient if there is an even generalised star.

## 7.6 The $\kappa$-problem on bipartite graphs

To find a possible complete bipartite decomposition of an arbitrary bipartite graph we can systematically work our way through each vertex in one of the sets of vertices.

In a bipartite graph we can partition the vertices in the sets $I$ and $J$,

$$I = \{v_{i_1}, ..., v_{i_n}\}$$
$$J = \{v_{j_1}, ..., v_{j_m}\}$$

where $|I| = n$ and $|J| = m$.

We want to find complete bipartite subgraph and using the same notation as earlier we have

$$S_i = (I_i, J_i)$$

where there is still no need for a set of edges, it is obvious which edges that are in each subgraph, because the subgraphs always are complete bipartite.

### 7.6.1 An algorithm to find a complete bipartite decompsiton of a bipartite graph

The following algorithm can be used to find a complete bipartite decomposition of an arbitrary bipartite subgraph.

1. Choose $v_{i_1} \in I$. We make a complete bipartite subgraph

   $$S_1 = (I_1, J_1)$$

   where $v_{i_1} \in I_1$ and $v_{j_a} \in J_1$, $v_{j_a}$ is every vertex adjacent to $v_{i_1}$.

2. Choose $v_{i_2} \in I$ and find all $v_{j_{a_2}} \in J$ that are adjacent to $v_{i_2}$.

   (a) If $v_{i_2}$ is adjacent to the exact same set as $v_{i_1}$, then $v_{i_2} \in I_1$, and we still have only one subgraph.

   (b) If $v_{i_2}$ is not adjacent to the exact same set as $v_{i_1}$, then we get a new subgraph

   $$S_2 = (I_2, J_2)$$

   where $v_{i_2} \in I_2$ and $v_{j_a} \in J_2$, $v_{j_a}$ is every vertex adjacent to $v_{i_2}$.

3. In general, choose $v_{i_k} \in I$, $1 \leq k \leq n$.

   (a) If $v_{i_k}$ is adjacent to the exact same set as one of $J_1, ..., J_p$, where $p$ is the number of former subgraphs, then $v_{i_k} \in I_p$

   (b) If $v_{i_k}$ is not adjacent to the exact same set as $J_1, ..., J_p$, the former subgraphs, then $v_{i_k} \in I_{p+1}$ and $v_{j_k} \in J_{p+1}$

For each step we get maximum one new subgraph. This means that the maximum number of complete bipartite graphs we can get using this algorithm is $|I|$.

Example 7.6.1. The algorithm can be demonstrated on the bipartite graph

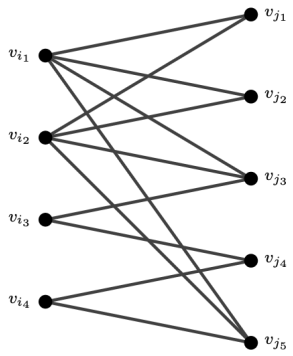



Figure 7.10: Example of a bipartite graph.

This results in the following complete bipartite subgraphs.

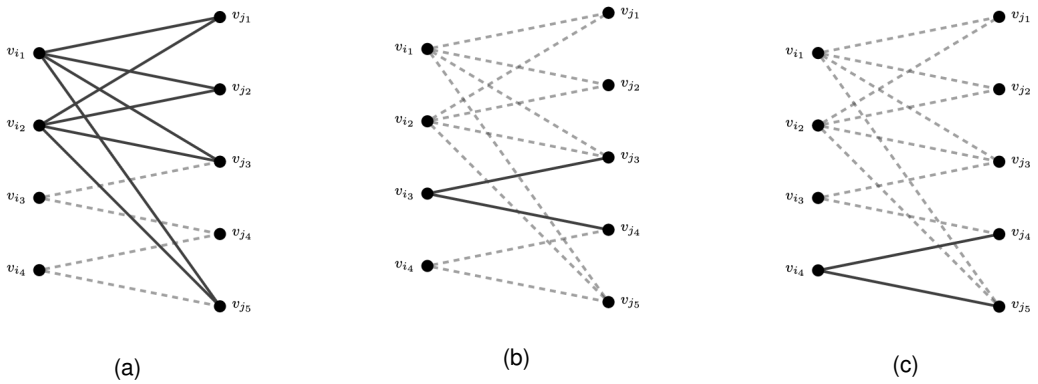



Figure 7.11: A complete bipartite decomposition of Figure 7.10.

### 7.6.2 Implementing the algorithm in Python

Using this algorithm to find a possible decomposition of a bipartite graph may be time consuming if given large bipartite graphs. I have therefore implemented this algorithm in Python using a part of the adjacency matrix of the graph.

In an adjacency matrix of a bipartite graph, there will always be lots of zeros, as it is never any edges connecting the vertices in the same set. This means that the upper left $|I| \times |I|$-matrix and the lower right $|J| \times |J|$-matrix will both be the null matrix with corresponding dimensions. Thus, the interesting parts of an adjacency matrix of a bipartite graph is the upper right $|I| \times |J|$-matrix or the lower left $|J| \times |I|$-matrix. They will both sufficiently describe a bipartite graph and are the transposed matrix of the other.

Using Figure 7.10 as an example, we have the adjacency matrix in Figure 7.12. The interesting parts of the matrix in Figure 7.12 is, as mentioned, the upper right $4\times5$-matrix and lower left $5\times4$-matrix, shown with the dashed lines. Both of the matrices in Figure 7.13 are bipartite adjacency matrices and sufficiently describe the bipartite graph.

| | $v_{i_1}$ | $v_{i_2}$ | $v_{i_3}$ | $v_{i_4}$ | $v_{j_1}$ | $v_{j_2}$ | $v_{j_3}$ | $v_{j_4}$ | $v_{j_5}$ |
|---|---|---|---|---|---|---|---|---|---|
| $v_{i_1}$ | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| $v_{i_2}$ | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| $v_{i_3}$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| $v_{i_4}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| $v_{j_1}$ | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $v_{j_2}$ | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $v_{j_3}$ | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| $v_{j_4}$ | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| $v_{j_5}$ | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

Figure 7.12: The adjacency matrix of the bipartite graph in Figure 7.10.

| | $v_{j_1}$ | $v_{j_2}$ | $v_{j_3}$ | $v_{j_4}$ | $v_{j_5}$ |
|---|---|---|---|---|---|
| $v_{i_1}$ | 1 | 1 | 1 | 0 | 1 |
| $v_{i_2}$ | 1 | 1 | 1 | 0 | 1 |
| $v_{i_3}$ | 0 | 0 | 1 | 1 | 0 |
| $v_{i_4}$ | 0 | 0 | 0 | 1 | 1 |

(a) A matrix showing the edges connecting the vertices in $I$ and $J$.

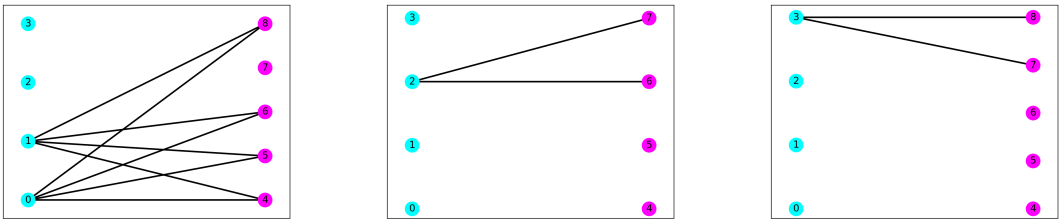
| | $v_{i_1}$ | $v_{i_2}$ | $v_{i_3}$ | $v_{i_4}$ |
|---|---|---|---|---|
| $v_{j_1}$ | 1 | 1 | 0 | 0 |
| $v_{j_2}$ | 1 | 1 | 0 | 0 |
| $v_{j_3}$ | 1 | 1 | 1 | 0 |
| $v_{j_4}$ | 0 | 0 | 1 | 1 |
| $v_{j_5}$ | 1 | 1 | 0 | 1 |

(b) Another matrix showing the edges connecting the vertices in $I$ and $J$.

Figure 7.13: Two different ways to describe Figure 7.10 using matrices.

It is possible to use either of Figure 7.13a and Figure 7.13b to find a possible decomposition of the bipartite graph. A complete bipartite graph can be represented by a row in either of the matrices, where $v_i \in I$ or $v_j \in I$ the columns containing 1 are the set of vertices J. Similarly, each column in either of the matrices can also describe a complete bipartite graph, where $v_i \in I$ or $v_j \in I$ and the rows containing 1 is the set of vertices $J$. For instance, $I = \{v_{i_3}\}$ and $J = \{v_{j_3}, v_{j_4}\}$, also shown in Figure 7.11b.

The Python code finds a possible decomposition of a bipartite subgraph by using a bipartite adjacency matrix, given by the user. It then finds the number of unique rows in the matrix and then draws every complete bipartite subgraph, see Appendix A for the complete code.




(a) (b) (c)

Figure 7.14: The subgraphs found using the algorithm in Python.

# References


[AMO93] Ahuja, R. K., Magnanti, T. L. and Orlin, J. B. *Network flows*. Theory, algorithms, and applications. Prentice Hall, Inc., Englewood Cliffs, NJ, 1993, pp. xvi+846.

[AZ18] Aigner, M. and Ziegler, G. M. *Proofs from The Book*. Sixth. See corrected reprint of the 1998 original [ MR1723092], Including illustrations by Karl H. Hofmann. Springer, Berlin, 2018, pp. viii+326.

[Bar23] Barazzutti, R. *Co-authorship distance computation*. https://www.csauthors.net/distance/geir-dahl/paul-erdos. [Online; accessed 14-May-2023]. Feb. 2023.

[BK02] Balaban, A. T. and Klein, D. J. 'Co-authorship, rational Erdős numbers, and resistance distances in graphs'. In: *Scientometrics* 55 (2002), pp. 59–70.

[BM08] Bondy, J. A. and Murty, U. S. R. *Graph theory*. Vol. 244. Graduate Texts in Mathematics. Springer, New York, 2008, pp. xii+651.

[Dah13] Dahl, G. 'Network flows and combinatorial matrix theory'. In: 2013.

[Fou12] Foulds, L. R. *Graph theory applications*. Springer Science & Business Media, 2012.

[MKR16] Moore, K., Khim, J. and Ross, E. *Greedy algorithms*. https://brilliant.org/wiki/greedy-algorithm/. [Online; accessed 29-April-2023]. 2016.

[Wei23] Weisstein, E. *Wheel graph*. https://mathworld.wolfram.com/WheelGraph.html. [Online; accessed 14-May-2023]. Apr. 2023.

[Wik23a] Wikipedia. *Erdős number — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/wiki/ErdÅŚs_number. [Online; accessed 14-May-2023]. 2023.

[Wik23b] Wikipedia. *Graham–Pollak theorem — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/wiki/GrahamâĂŞPollak_theorem. [Online; accessed 24-April-2023]. 2023.

[Wik23c] Wikipedia. *Star (graph theory) — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/wiki/Star_(graph_theory). [Online; accessed 15-May-2023]. 2023.

Appendix A

# Decomposition of bipartite graphs into complete bipartite subgraphs

```
import networkx as nx
import matplotlib.pyplot as plt
from networkx.algorithms import bipartite
import numpy as np

row = int(input("Enter the number of rows:"))
column = int(input("Enter the number of columns:"))

matrix = []
print("Enter the entries rowwise:")

for i in range(row):
    list = []
    for j in range(column):
         list.append(int(input()))
    matrix.append(list)


subs = []

for i in matrix:
    sub = []
    for j in matrix:
        if i==j:
            sub.append(i)
        else:
            sub.append([0]*len(matrix[0]))
    subs.append(sub)

unique_sub = []

for i in subs:
    if i not in unique_sub:
        unique_sub .append(i)
```

```
for i in unique_sub:
    print (np.matrix(i))

print("unique_sub = ", unique_sub)


for k in unique_sub:

    kant = []
    vertex_i = []
    vertex_j = []
    print(k)
    for i in range(len(k)):
        vertex_i.append(i)
        vertex_j.append(len(k) + i)
        for j in range(len(k[i])):
            if k[i][j] == 1:
                kant.append((i,len(k) +j))

    B = nx.Graph()

    B.add_nodes_from(vertex_i, bipartite=0)
    B.add_nodes_from(vertex_j, bipartite=1)
    B.add_edges_from(kant)

    colour = []

    for vertex in B:
        if vertex < len(k):
            colour.append('red')
        else:
            colour.append('green')

    nx.draw_networkx(B, node_color = colour, pos =
    ↪  nx.drawing.layout.bipartite_layout(B, vertex_i), width = 2)

    plt.show()
```