University of Bergen
Department of Informatics

Properties of locally checkable vertex partitioning problems in digraphs

Author: Petter Daae
Supervisor: Jan Arne Telle

May, 2022

Abstract

While, for undirected graphs, locally checkable vertex subset and partitioning problems have been studied extensively, the equivalent directed problems have not received nearly as much attention yet. We take a closer look at the relationship between undirected and directed problems considering hardness. We extend some properties that have already been shown for undirected graphs to directed graphs. Furthermore, we explore some of the trivialities in directed problem definitions that do not appear in undirected ones. And finally, we construct and visualize digraph coverings to achieve a deeper understanding of their structure.

Acknowledgements

First and foremost, I would like to thank my supervisor, Jan Arne, for invaluable guidance and feedback. I would also like to thank my friends and family for their support during the entire degree and enjoyable conversations during lunch.

`Petter Daae`
Friday 20th May, 2022

Contents

Chapter 1

Introduction

Graphs are interesting both from a theoretical and practical perspective. They are suitable for modeling a huge variety of real-world situations and problems. Some examples are *maps*, *social networks*, *digital circuits* and *molecules*. Some less obvious examples are *dictornaries*, *music theory*, *law* and *climbing*. The only thing that stops you from modeling anything as a graph is your imagination. The problems that appear in the real world are sometimes simple. For example, finding the shortest path between two cities on a map, checking how many friends people have in common, or calculating the voltage or current at different points in a digital circuit. We refer to such problems as tractable. They are easy for computers to solve and have good theoretical (algorithmic) solutions. However, we also run into harder problems. The interesting part is that many of these sound like straightforward problems. Consider finding the largest group among your friends that do not know each other. In graph theory, we refer to this problem as MAX INDEPENDENT SET, and it is part of a rich group of problems known as vertex partitioning problems, which is the group of problems we focus on in this thesis. More specifically, we focus on such problems in digraphs. An example of a digraph in real life is a digraph that models *money transfers*. If person A transfers money to person B, there is a direction associated with their relationship as well as the edge between person A and B in the graph that models the transfers.

Many vertex partitioning problems are closely related, and in 1993, Telle [21] introduced a general framework for describing a subset of such problems, *locally checkable* ones. These are the partitioning problems where solutions can be verified by inspecting the *local* neighborhood of each vertex seperately. The most common examples of such problems are INDEPENDENT SET, DOMINATING SET and COLORING. A lot of progress has been made in the exploration of problems described by the framework for undirected graphs. However, the framework was not extended to directed graphs until Jaffke, Kwon, and Telle published their paper, *Classes of intersection digraphs with good algorithmic properties* [14], in 2021. This paper gives the formal definition of the framework for directed graphs, introduces a new width-parameter, *the bi-mim width*, and gives an algorithm that solves all such problems in XP-time parameterized by the bi-mim width, if given a decomposition.

In this thesis, we take a step back, consider a set of properties that have been shown for undirected graphs, and see if they can be extended to directed graphs. In Chapter 3, we motivate our work by comparing undirected graphs and digraphs. In Chapter 4, we extend a property for undirected graphs, from [22], to digraphs; and we explore some trivialities that were introduced when extending the framework to directed graphs. In Chapter 5, we explore the relationship between similar undirected and directed problems considering the hardness. And finally, in Chapter 6, we end our thesis by constructing and visualizing digraph coverings (which have a nice link to degree constraint matrices) for a deeper understanding of their structure.

Chapter 2

Preliminaries

2.1 Sets of numbers

Let $\mathbb{Z}$ denote the set of all integers. Let $\mathbb{Z}^*$ denote the set of all non-negative integers, $\{i : i \in \mathbb{Z} \wedge i \geq 0\}$. Let $\mathbb{Z}^+$ denote the set of all positive integers, $\{i : i \in \mathbb{Z} \wedge i > 0\}$.

2.2 Graphs

A graph G consists of a set of vertices V (also denoted as $V(G)$) and a set of edges E (also denoted as $E(G)$). In Fig. 2.1, we see an example of a graph where $V = \{a, b, c\}$ and $E = \{(a, b), (b, c)\}$. We say that two vertices are *adjacent* if there is an edge between them. The open neighbourhood of a vertex v consists of all vertices adjacent to v and is denoted as $N(v)$. The closed neighborhood of a vertex v is $N[v] = N(v) \cup \{v\}$. We use $deg(v)$ to denote how many vertices are adjacent to v, in other words $deg(v) = |N(v)|$.

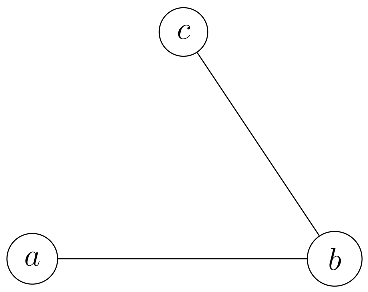


Figure 2.1: Example of an undirected graph

The graph in Fig. 2.1 is an undirected graph. This means that the edges are not associated with a direction. Graphs where the edges are associated with a direction are known as directed graphs, or digraphs. In Fig. 2.2 we see an example of a digraph, where $V = \{a, b, c\}$ and $E = \{(a, b), (b, a), (c, b)\}$. In digraphs we denote the open out-neighbourhood and in-neighbourhood of a vertex v by $N^+(v)$ and $N^-(v)$, respectively. The closed out- and in- neighbourhoods are denoted by $N^+[v]$ and $N^-[v]$. We use $deg^+(v)$ and $deg^-(v)$ to denote the size of the open out- and in-neightbourhoods of v. In other words, $deg^+(v) = |N^+(v)|$ and $deg^-(v) = |N^-(v)|$.

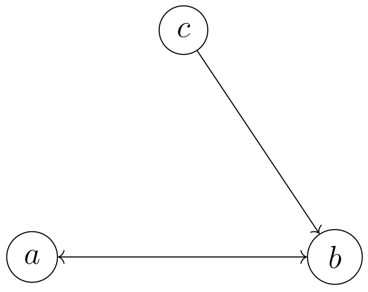


Figure 2.2: Example of a directed graph

The *biorientation* of an undirected graph is the equivalent digraph that has edges in both directions where the undirected graph has edges.

Generally, we only consider *simple graphs.* Simple graphs do not have more than one edge between two vertices (no multi-edges) and no loops (an edge that starts and ends at the same vertex). If we allow any of these, it will be clear from the context.

2.3 Vertex subset problems

A vertex subset problem asks for a subset of vertices of a graph such that the subset has some special property. One example is Independent Set. An independent set is a set of vertices of a graph such that no two vertices in the set are adjacent. We focus on *locally checkable* vertex subset problems. These are the vertex subset problems where you can check on each vertex locally if the set has the desired property. Aside from independent set, other examples of such problems are Clique, Dominating Set and Perfect Code. Examples of vertex subset problems that are not locally checkable are Feedback Vertex Set and Odd Cycle Transversal. In most cases, we are interested in maximizing or minimizing the vertex subset. This also tends to be the hard part of the problem. For some of the problems, though, the existence question is hard to answer as well.

σ	ρ	Property name
$\{0\}$	$\mathbb{Z}^*$	Independent set
$\mathbb{Z}^*$	$\mathbb{Z}^+$	Dominating set
$\{0\}$	$\mathbb{Z}^+$	Strong Stable set or 2-packing
$\{0\}$	$\{1\}$	Efficient Dominating set or Perfect Code
$\{1\}$	$\{0\}$	Independent Dominating set
$\mathbb{Z}^*$	$\{1\}$	Perfect Dominating set
$\mathbb{Z}^+$	$\mathbb{Z}^+$	Total Dominating set
$\{1\}$	$\{1\}$	Total Perfect Dominating set
$\mathbb{Z}^*$	$\{0,1\}$	Nearly Perfect set
$\{0,1\}$	$\{0,1\}$	Total Nearly Perfect set
$\{0,1\}$	$\{1\}$	Weakly Perfect set
$\{i : i \leq p\}$	$\mathbb{Z}^*$	Induced Bounded-Degree subgraph
$\mathbb{Z}^*$	$\{i : i \geq p\}$	p-Dominating set
$\{p\}$	$\mathbb{Z}^*$	Induced p-regular subgraph

Table 2.1: Examples of vertex subset properties described as (σ, ρ)-sets.

2.3.1 Characterization of LCVS problems

In this section, we describe a general framework for the characterization of locally checkable vertex subset problems, LCVS problems. It was introduced by Telle in [21] and later extended to directed graphs by Jaffke, Kwon, and Telle in [14].

Definition 1. *[21] Let $\sigma, \rho \subseteq \mathbb{Z}^*$. Let G be an undirected graph and $S \subseteq V(G)$. We say that S is a (σ, ρ)-set, if*

$$\forall v \in V(G) : |N(v) \cap S| \in \begin{cases} \sigma, & \text{if } v \in S \\ \rho, & \text{if } v \notin S \end{cases}$$

There are multiple problems related to (σ, ρ)-sets. Generally, we refer to these as (σ, ρ)-problems. In most cases, we consider the maximization or minimization of (σ, ρ)-sets, that is, finding the biggest or smallest possible set of vertices S that is a (σ, ρ)-set. We also consider the existence problem, whether there exists any set of vertices S that is a (σ, ρ)-set, but this is in many cases solvable in polynomial time. To show the expressiveness of the framework, we give an overview of well-studied vertex subset properties expressed with the (σ, ρ)-set notation, in Table 2.1 (as given in [22]).

σ^+	σ^-	ρ^+	ρ^-	Property name
$\{0\}$	$\{0\}$	$\mathbb{Z}^+$	$\mathbb{Z}^*$	Kernel [24]
$\{i : i < k\}$	$\{0\}$	$\{i : i \geq l\}$	$\mathbb{Z}^*$	(k, l)-out Kernel [19]
$\mathbb{Z}^*$	$\mathbb{Z}^*$	$\mathbb{Z}^*$	$\mathbb{Z}^+$	Dominating set [12]
$\{0\}$	$\{0\}$	$\mathbb{Z}^*$	$\mathbb{Z}^+$	Independent Dominating set [6]
$\mathbb{Z}^*$	$\mathbb{Z}^*$	$\mathbb{Z}^+$	$\mathbb{Z}^*$	Absorbing set [10]
$\mathbb{Z}^*$	$\mathbb{Z}^*$	$\mathbb{Z}^+$	$\mathbb{Z}^+$	Twin Dominating set [11]
$\mathbb{Z}^*$	$\mathbb{Z}^*$	$\mathbb{Z}^*$	$\{i : i \geq k\}$	k-Dominating set [18]
$\mathbb{Z}^*$	$\mathbb{Z}^+$	$\mathbb{Z}^*$	$\mathbb{Z}^+$	Total Dominating set [1]
$\{0\}$	$\{0\}$	$\mathbb{Z}^*$	$\{1\}$	Efficient (Closed) Dominating set [3]
$\mathbb{Z}^*$	$\{0\}$	$\mathbb{Z}^*$	$\{1\}$	Efficient Total Dominating set [20]
$\{k\}$	$\{k\}$	$\mathbb{Z}^*$	$\mathbb{Z}^*$	k-Regular Induced Subdigraph [5]

Table 2.2: Examples of vertex subset properties described as (Σ, R)-sets

2.3.2 Characterization of DLCVS problems

Now, we give the equivalent categorization for digraphs, a characterization of directed locally checkable vertex subset problems, DLCVS problems.

Definition 2. *[14] Let $\sigma^+, \sigma^-, \rho^+, \rho^- \subseteq \mathbb{Z}^*$, $\Sigma = (\sigma^+, \sigma^-)$ and $R = (\rho^+, \rho^-)$. Let G be a digraph and $S \subseteq V(G)$. We say that S is a (Σ, R)-set if:*

$$\forall v \in V(G) : |N^+(v) \cap S| \in \begin{cases} \sigma^+, & \text{if } v \in S \\ \rho^+, & \text{if } v \notin S \end{cases} \quad \text{and} \quad |N^-(v) \cap S| \in \begin{cases} \sigma^-, & \text{if } v \in S \\ \rho^-, & \text{if } v \notin S \end{cases}$$

We consider the same kinds of problems for (Σ, R)-sets as for the (σ, ρ)-sets and denote them as (Σ, R)-problems. Note that, when we discuss (Σ, R)-sets in later sections, it is implied that Σ always consists of σ^+ and σ^-, and that R always consists of ρ^+ and ρ^-, as well as the other way around. Like the undirected (σ, ρ)-set properties above, we give an overview of well-studied vertex subset properties in digraphs expressed with the (Σ, R)-set notation, in Table 2.2 (as given in [14]).

2.4 Vertex partitioning problems

A vertex partitioning problem asks for a partitioning of the vertices in a graph into partition classes, such that each partition class has some special property. One example is Coloring. It is similar to the vertex subset problem Independent Set, and asks for an assignment of color to each vertex in the graph, such that a vertex with some color c is not adjacent to any other vertex with color c. The colors we give to the vertices define the partitioning. Again, as with the vertex subset problems, we focus on *locally checkable* vertex partitioning problems.

2.4.1 Characterization of LCVP problems

In this section, we look at a more general characterization that covers locally checkable vertex partitioning problems (also denoted as LCVP problems). As for the vertex subset problems, the undirected definition was introduced by Telle in [22].

Definition 3. *[22] A degree constraint matrix D_q is a q by q matrix with entries being subsets of $\mathbb{Z}^*$. A D_q-partition in a graph G is a q-partition $V_1, V_2, ..., V_q$ of $V(G)$ such that for $1 \leq i, j \leq q$ we have $\forall v \in V_i : |N_G(v) \cap V_j| \in D_q[i, j]$.*

For example, we can express 3-coloring with a 3 by 3 degree constraint matrix where the diagonal entries are $\{0\}$ while the off-diagonal entries are $\mathbb{Z}^*$.

$$\begin{bmatrix} \{0\} & \mathbb{Z}^* & \mathbb{Z}^* \\ \mathbb{Z}^* & \{0\} & \mathbb{Z}^* \\ \mathbb{Z}^* & \mathbb{Z}^* & \{0\} \end{bmatrix}$$

In contrast to vertex subset problems, where in most cases, the existence problem is easy while the optimization problems are hard, it is very often the case that the existence problem for partitioning problems is hard as well. We define optimization problems for partitioning problems in terms of minimizing or maximizing the number of partitions. The way we use *degree constraint matrices* in this context is that we give the on- and off-diagonal entries of the matrix. When we do this, for problems such as Coloring, the existence problem is often NP-hard, even for a small number of partition classes [13].

Note that we can also characterize LCVS problems with degree constraint matrices. For example, the (σ, ρ)-problem on undirected graphs can be expressed with the following 2 by 2 degree constraint matrix:

$$\begin{bmatrix} \sigma & \mathbb{Z}^* \\ \rho & \mathbb{Z}^* \end{bmatrix}$$

2.4.2 Characterization of DLCVP problems

Now, we give the directed equivalent of the degree constraint matrix for undirected graphs introduced by Jaffke, Kwon, and Telle in [14]. We use this to characterize directed locally checkable vertex partitioning problems, DLCVP problems.

Definition 4. *[14] A bi-neighborhood constraint matrix is a $(q \times q)$-matrix D_q over pairs of finite or co-finite sets of natural numbers. Let G be a digraph, and $\chi = (X_1, ..., X_q)$ be a q-partition of $V(G)$. We say that χ is a D-partition if for all $i, j \in \{1, ..., q\}$ with $D_q[i,j] = (\mu^+_{i,j}, \mu^-_{i,j})$, we have that for all $v \in X_i$, $|N^+(v) \cap X_j| \in \mu^+_{i,j}$ and $|N^-(v) \cap X_j| \in \mu^-_{i,j}$.*

Note that we can also characterize directed locally checkable vertex subset problems with bi-neighborhood constraint matrices. For example, the (Σ, R)-problem on digraphs can be expressed with the following 2 by 2 bi-neighborhood constraint matrices:

$$\begin{bmatrix} (\sigma^+, \sigma^-) & (\mathbb{Z}^*, \mathbb{Z}^*) \\ (\rho^+, \rho^-) & (\mathbb{Z}^*, \mathbb{Z}^*) \end{bmatrix}$$

Chapter 3

Motivation

3.1 Classes of intersection digraphs with good algorithmic properties

In [14], Jaffke, Kwon, and Telle explore several concepts from the undirected graph literature that have not previously been discussed for digraphs. They start by introducing a new width parameter, the bi-mim-width, which is an extension of the mim-width introduced for undirected graphs by Vatshelle in [23]. They proceed to introduce several new classes of intersection digraphs, as well as bounds for their bi-mim-width. Finally, they define directed locally checkable vertex problems (which is the main focus of this thesis) and algorithms that solve them in XP time, parameterized by the bi-mim-width of the input graph, given a decomposition.

3.2 The space of graphs

Some of the results we explore in this thesis are results for digraphs inspired by similar results for undirected graphs. Although a lot of the properties of undirected graphs directly translate to digraphs, they appear to be more complex. In later sections, we prove that a lot of problems are at least as hard for directed graphs as for similar problems on undirected graphs.

In Fig. 3.1, we see all non-isomorphic simple undirected graphs with 4 vertices (11), and all non-isomorphic simple digraphs with 4 vertices (218). The space of digraphs is clearly a lot richer than the space of undirected graphs.

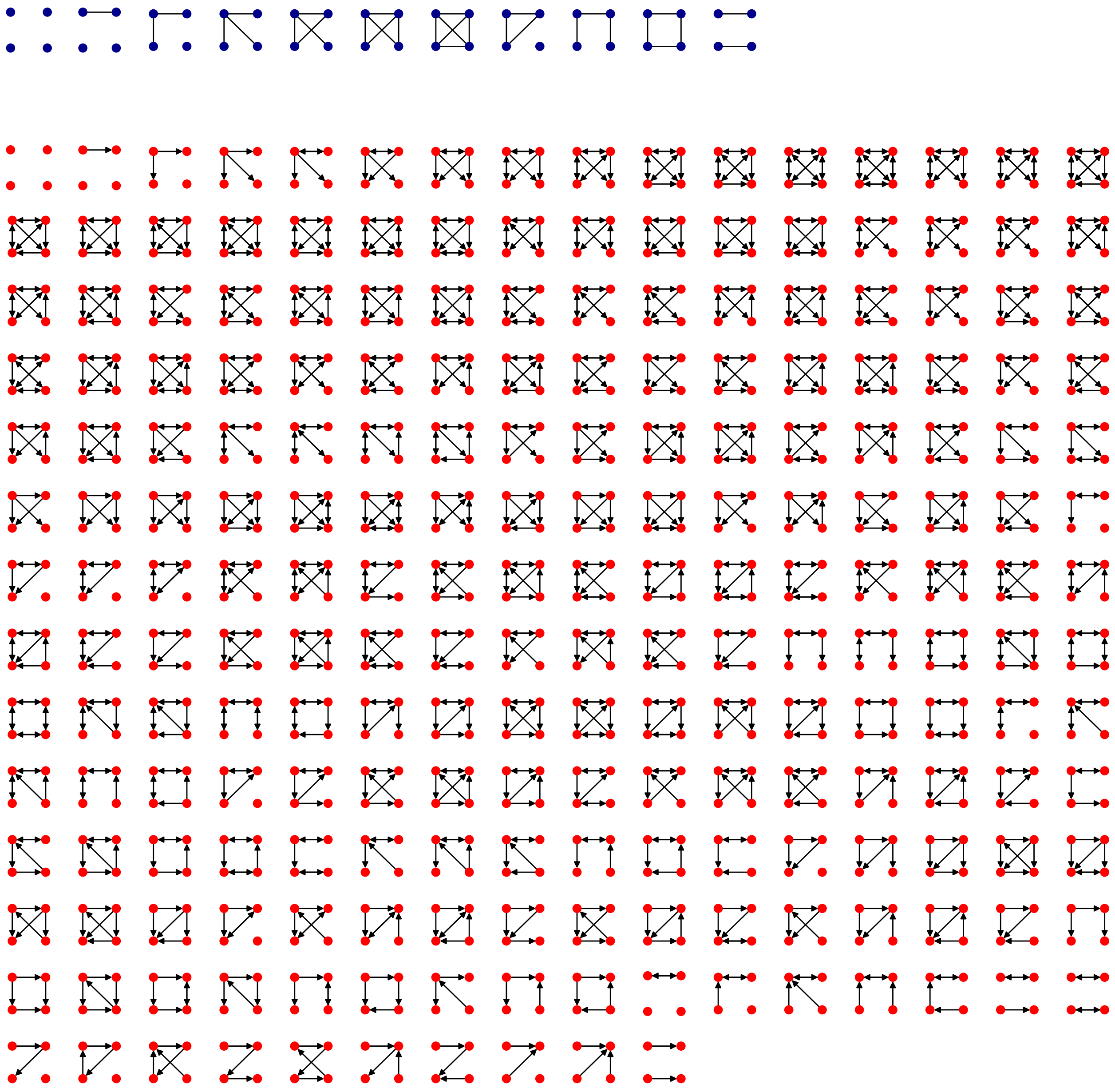


Figure 3.1: All non-isomorphic, simple undirected graphs (in blue) and digraphs (in red), with 4 vertices.

3.3 Vertex partitioning problems on digraphs

One reason partitioning problems are interesting is that many of them are hard to solve. Many of them are also closely related to each other, which means that if we find a good solution to one of them, it can lead us to good solutions for other ones as well. Vertex partitioning problems on undirected graphs is a very popular research area. A huge amount of terminology and definitions have been introduced, and many approaches to solving the problems have been explored. Similar problems on digraphs are not so well studied, but have started to receive attention. We refer to Bang-Jensen and Gutin [2] from 2018, which at the moment is one of the few books covering a considerable amount of theory for digraphs.

3.3.1 Equivalence of direction

An interesting property of problems on digraphs is that the same problem is sometimes known under different names. Consider KERNEL, that we mention in the next section, and INDEPENDENT DOMINATING SET. A set of vertices that forms a kernel in some digraph is also an independent dominating set in the same digraph, but with all edges reversed.

3.3.2 An example from game theory

In 1953, von Neumann and Morgenstern introduced the notion of a *kernel* [24]. In a digraph, it is defined as an independent that is reachable from every vertex outside the set (INDEPENDENT ABSORBING SET). It is particularly interesting in the context of combinatorial games. In *Games of No Chance*[17], *Nowkowski* give examples of many such games as well as the standard definition, which is the following:

1. there are two players moving alternately;
2. there are no chance devices and both players have perfect information;
3. the rules are such that the game must eventually end; and
4. there are no draws, and the winner is determined by who moves last.

We can define such a game on a directed acyclic graph as Boros and Gurvich do in *Perfect graphs, kernels and cores of cooperative games* [4]. Given such a digraph, two players and a token on one of the vertices, the players alternate on moving the token along the edges of the graph. One of the players win if the other player is not able to make a move, in other words, if the vertex that the token is on has no outgoing edges.

The interesting property of a kernel in this context, is that if the token is on a vertex outside the kernel and you are the one to make a move, you have a winning strategy. The strategy is to move the token into the kernel. Since the kernel is *absorbing*, that is, all vertices outside the kernel has an edge into the kernel, you will always have this option. After this, the token is on a vertex inside the kernel. Because the kernel is an independent set, it is not possible for the other player to move the token anywhere else than outside the kernel. Then again, you can move the token back into the kernel, because of the absorbing property. This continues until the token ends up on a vertex inside the kernel that has no outgoing edges. And in this case, it is the other players turn, and you win!

Chapter 4

General properties

4.1 Size of strict (Σ, R)-sets

For undirected graphs we have the following lemma from [22]:

Lemma 1. *[22] Let $q \in \mathbb{Z}^*$, $p \in \mathbb{Z}^+$, $\sigma = \{q\}$ and $\rho = \{p\}$. If both A and B are (σ, ρ)-sets of a graph G, then $|A| = |B|$.*

We now extend this lemma to (Σ, R)-sets in digraphs. The proof we give is similar to the proof given for Lemma 1 in [22].

Lemma 2. *Let $q_{out}, q_{in}, p_{out}, p_{in} \in \mathbb{Z}^*$, $p_{out} + p_{in} \in \mathbb{Z}^+$, $\Sigma = (\{q_{out}\}, \{q_{in}\})$ and $R = (\{p_{out}\}, \{p_{in}\})$. If both A and B are (Σ, R)-sets in a digraph G, then $|A| = |B|$.*

Proof. Let A and B be (Σ, R)-sets in a digraph G. Let $X_I = A \cap B$, $X_A = A \setminus B$ and $X_B = B \setminus A$. We define two subdigraphs, F and H, of $A \cup B$ that both contain all vertices of $A \cup B$. F is the subdigraph that contains the edges with one endpoint in X_A and another endpoint in X_B (see Fig. 4.1). H is the subdigraph that contains the edges with one endpoint in X_I and another endpoint in X_A or X_B (see Fig. 4.2). We now proceed to show that $(p_{out} + p_{in})|X_A| = (p_{out} + p_{in})|X_B|$ which implies $|X_A| = |X_B|$ (and $|A| = |B|$), because $p_{out} + p_{in} \in \mathbb{Z}^+$.

We first observe that, for any vertex v in X_A, $deg_F^+(v) + deg_H^+(v) = p_{out}$. Because B is a (Σ, R)-set and v is outside B, we require that v has p_{out} out-neighbours in B; and these are exactly the out-neighbours of v in $F \cup H$. Note that $E(F) \cap E(H) = \emptyset$. The same

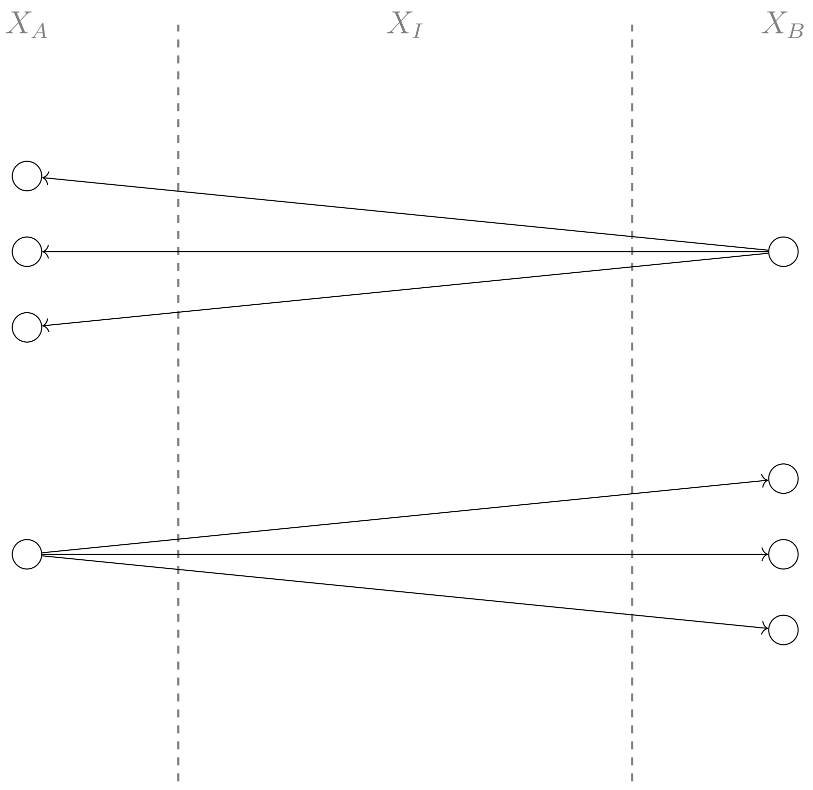


Figure 4.1: F

argument holds for $deg_F^-(v)+def_H^-(v) = p_{in}$ and also for any v in X_B because A is a (Σ, R)-set. From this we get $(p_{out} + p_{in})|X_A| = \sum_{v \in X_A} deg_F^+(v) + deg_H^+(v) + deg_F^-(v) + deg_H^-(v)$ as well as the equivalent expression for $(p_{out} + p_{in})|X_B|$.

Next, we argue that $\sum_{v \in X_A} deg_F^+(v) + deg_F^-(v) = \sum_{v \in X_B} deg_F^+(v) + deg_F^-(v)$. This follows from the fact that F is bipartite. Another way to think about it is that every edge in F adds 1 to the total degree of all the vertices in both X_A and X_B.

Our final observation is that $\sum_{v \in X_A} deg_H^+(v) + deg_H^-(v) = \sum_{v \in X_B} deg_H^+(v) + deg_H^-(v)$. We show this by considering any vertex v in X_I. Since we require that v has σ_{out} out-neighbours both in A and in B, we have that $|N^+(v) \cap A| = |N^+(v) \cap B|$. Subtracting $|N^+(v) \cap X_I|$ from both sides of the equation, we get $|N^+(v) \cap X_A| = |N^+(v) \cap X_B|$. The same reasoning holds for the in-neighbours of v. So, every vertex v in X_I has equally many neighbours in X_A and X_B. This implies that the total degree of X_A in H equals the total degree of X_B in H, or $\sum_{v \in X_A} deg_H^+(v) + deg_H^-(v) = \sum_{v \in X_B} deg_H^+(v) + deg_H^-(v)$.

Finally, we combine all our observations in the following equation: $(p_{out} + p_{in})|X_A| = \sum_{v \in X_A} deg_F^+(v) + deg_F^-(v) + deg_H^+(v) + deg_H^-(v) = \sum_{v \in X_B} deg_F^+(v) + deg_F^-(v) + deg_H^+(v) + deg_H^-(v) = (p_{out} + p_{in})|X_B|$, which implies $|X_A| = |X_B|$ and $|A| = |B|$. □

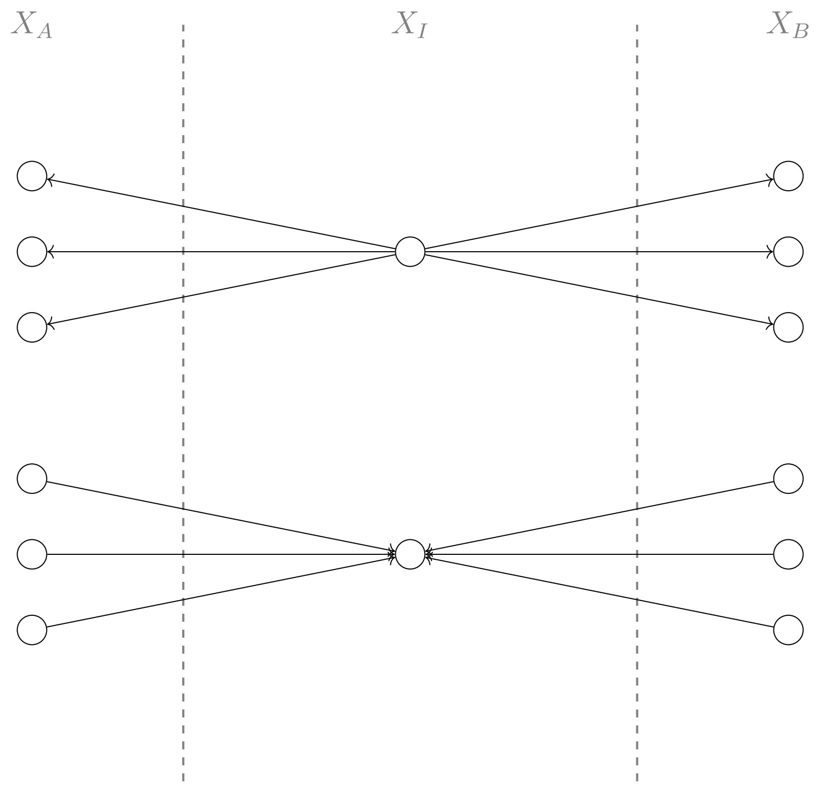


Figure 4.2: H

For undirected graphs, there is an even stronger claim in Theorem 1. Whether there is a similar property for digraphs, we leave as an open question.

Theorem 1. *[22] For any graph G, all (σ, ρ)-sets have the same size if and only if σ or ρ is the empty set or if for $q \in \mathbb{Z}^*$ and $p \in \mathbb{Z}^+$, $\sigma = \{q\}$ and $\rho = \{p\}$.*

4.2 Trivial (Σ, R)-sets

In contrast to (σ, ρ)-sets in undirected graphs, (Σ, R)-sets in digraphs do not always make sense. For example, if $\sigma^+ = \{0\}$ and $\sigma^- = \{1\}$, the only possible (Σ, R)-set is the empty set. In this section, we explore different Σ and R such that there only exists digraphs whose full or empty set of vertices are (Σ, R)-sets.

Lemma 3. *Assume that one of σ^+, σ^-, ρ^+ and ρ^- is the empty set. If there exists a (Σ, R)-set S in any digraph G, then either $S = \emptyset$ or $S = V(G)$.*

Proof. If σ^+ or σ^- is the empty set, there can be no vertices in a (Σ, R)-set because we do not allow *any* size of the out- or in-neighbourhood, respectively, of vertices inside the set into the set (not even 0 neighbours). Similarly, if ρ^+ or ρ^- is the empty set and there exists a (Σ, R)-set, then all vertices must be in the set, because we do not allow any size of the out- or in-neighbourhood of vertices outside the set into the set (not even 0 neighbours). $\square$

Because of Lemma 3, we will only consider non-empty σ^+, σ^-, ρ^+ and ρ^- in the remaining part of this section. We start by looking at $R = (\rho^+, \rho^-)$ only.

Lemma 4. *Assume that there exists a digraph G with a (Σ, R)-set S, such that $|V(G)| \in \mathbb{Z}^+$ and $S = \emptyset$. Then, $0 \in \rho^+$ and $0 \in \rho^-$.*

Proof. If there are no vertices in S, then the vertices outside S must be allowed to have 0 in- and out-neighbours in S. $\square$

Lemma 5. *Let ρ^+ and ρ^- be non-empty. Assume that there exists some digraph G with a (Σ, R)-set $S \neq \emptyset$. Then there also exists some digraph G' with a (Σ, R)-set $S' \notin \{\emptyset, V(G')\}$.*

Proof. Let $S \neq \emptyset$ be a (Σ, R)-set in some digraph G. We construct a new digraph G'. We start with a vertex v that will be the vertex outside some (Σ, R)-set. We also add a copy of G to G'. Take some number x from ρ^+ and add x out-edges from v into S (S is the (Σ, R)-set in the copy of the digraph G that we added to G'). If there are not enough vertices in S, we add another copy of G to G', and continue to do so until we are able to add x out-edges from v into one or more copies of S. We satisfy the in-neighbourhood constraints from ρ^- similarly. We now have a (Σ, R)-set $S' \notin \{\emptyset, V(G')\}$ in G', which is the union of all S (in the copies of G) that we added to G'. $S' \neq \emptyset$ because $S \neq \emptyset$ and $S' \neq V(G')$ because $v \notin S'$. $\square$

Now that we know there exist (Σ, R)-sets $\notin \{\emptyset, V(G)\}$ for any non-empty ρ^+ and ρ^- in some digraph G, if G has a (Σ, R)-set $\neq \emptyset$, we turn or attention to Σ. Hence, we are now only interested in the restrictions that σ^+ and σ^- give. Because Σ only gives restrictions on the neighbourhoods inside (Σ, R)-sets, we only consider digraphs whose full set of vertices is a (Σ, R)-set.

Lemma 6. *Let $\sigma^+ = \{a\}$ and $\sigma^- = \{b\}$, such that $a \neq b$. If there exists a digraph G with a (Σ, R)-set $S \subseteq V(G)$, then $S = \emptyset$.*

Proof. Assume, without loss of generality, that $a > b$, and assume that there exists some (Σ, R)-set of size $n \in \mathbb{Z}^+$ in some digraph. To satisfy $\sigma^+ = \{a\}$, there has to be a total out-degree of $n \cdot a$ in the set. But to satisfy $\sigma^- = \{b\}$, there has to be a total in-degree of $n \cdot b$. Because the total out-degree must equal the total in-degree and $na > nb$, we reach our contradiction. $\square$

The next lemma gives us an even stronger promise and it's argument is almost identical to the one given for Lemma 6.

Lemma 7. *Let σ^+ and σ^- be non-empty, such that $min(\sigma^+) > max(\sigma^-)$ or $min(\sigma^-) > max(\sigma^+)$. If there exists a digraph G with a (Σ, R)-set $S \subseteq V(G)$, then $S = \emptyset$.*

Proof. Assume without loss of generality, that $min(\sigma^+) > max(\sigma^-)$. Also, assume that there exists some (Σ, R)-set of size $n \in \mathbb{Z}^+$. To satisfy σ^+, there has to be a total out-degree of at least $n \cdot min(\sigma^+)$ in the set. But the maximum total in-degree is $n \cdot max(\sigma^-)$. Because the total out-degree can not be greater than the total in-degree and $n \cdot min(\sigma^+) > n \cdot max(\sigma^-)$, we reach our contradiction. $\square$

Now that we have showed that non-overlapping σ^+ and σ^- are not possible to satisfy, we move on to prove that it is always possible to construct digraphs that satisfy overlapping σ^+ and σ^-. We start by proving the following theorem:

Lemma 8. *Let $\sigma^+ = \{a, b\}$ and $\sigma^- = \{c\}$ with $a < c < b$. Then there exists a digraph G with a (Σ, R)-set $S \subseteq V(G)$, such that $S \neq \emptyset$.*

Proof. We show how to construct digraphs whose full set of vertices are such (Σ, R)-sets. We start with a claim that will be a good starting point for the construction.

Claim 1. *Let $\sigma^+ = \{a, b\}$ and $\sigma^- = \{c\}$ with $a < c < b$. Assume that there exists a digraph G with a (Σ, R)-set $S = V(G)$. Let $n = |V(G)|$ and x be the number of vertices in S that has a out-neighbours. Then, x and n have to satisfy the following equation:*

$$x = n\frac{c - b}{a - b}$$

Proof. We know that there has to be $xa + (n - x)b$ out-edges and nc in-edges in S. Solving for x in $xa + (n - x)b = nc$, we get the above equation. $\square$

We now continue the proof of Lemma 8 by giving an overview of the algorithm for the construction of the desired (Σ, R)-sets:

1. Calculate appropriate values for n and x (from Claim 1).
2. Add x vertices labeled with a out-capacity and c in-capacity, as well as $n-x$ vertices labeled with b out-capacity and c in-capacity.
3. Start with an empty set of edges and greedily add edges between the vertices until all in- and out-capacities are satisfied. Allow loops and multi-edges.
4. Fix multi-edges.
5. Fix loops.

1,2,3: Since the number of vertices we are dealing with has to be and integer, we need to be carefull when using Claim 1. In other words, we need to choose an integer n such that x is an integer. The simplest way to do this is to choose $abs(a-b)$. Note that because $a < b$, we also know that $\frac{c-b}{a-b}$ is always less than 1. After this, we construct our digraph by adding edges between the vertices greedily until all capacities are satisfied. This works because we chose n and x that satisfy the equation from Claim 1, and because we allow multi-edges and loops.

4: The constructed graph have no guarantees for not containing multi-edges (or multi-loops). The trick we use to eliminate them is to add copies of the digraph. Imagine the following example. We constructed a digraph with two vertices u and v with two edges between them, see Fig. 4.3. We start by adding a copy of this digraph, see Fig. 4.4. To remove the multi-edges, we *swap* the endpoints of one of the multi-edges in each digraph with the other graph, see Fig. 4.5. The same strategy works for any size of a multi-edge, m. When we add $m-1$ copies of the digraph, we get for each vertex, $m-1$ new pairs of vertices that have no edge between them and $m-1$ new in/out-edges from the new copies that will maintain the original neighbourhood sizes. See Fig. 4.6 for an example of a multi-edge of size 3.

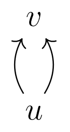


Figure 4.3: Diraph with a multi-edge of size 2

$$\begin{tikzcd}
v & v' \\
u \arrow[u, bend left] \arrow[u, bend right] & u' \arrow[u, bend left] \arrow[u, bend right]
\end{tikzcd}$$

Figure 4.4: Two copies of the digraph from Fig. 4.3

$$\begin{tikzcd}
v & v' \\
u \arrow[u, bend left] \arrow[ur] & u' \arrow[u, bend right] \arrow[ul]
\end{tikzcd}$$

Figure 4.5: Digraph from Fig. 4.4 without multi-edges

$$\begin{tikzcd}
v & v' & v'' \\
u \arrow[u] \arrow[ur] \arrow[urr] & u' \arrow[u] \arrow[ul] \arrow[ur] & u'' \arrow[u] \arrow[ul] \arrow[ull]
\end{tikzcd}$$

Figure 4.6: Multi-edge of size 3 fixed

$$\begin{tikzcd}
u \arrow[loop below] & u' \arrow[loop below] \\
u \arrow[r, bend right] & u' \arrow[l, bend right]
\end{tikzcd}$$

Figure 4.7: Loops fixed

5: Now we have a digraph with no multi-edges, but we might still have loops. We fix this with the same idea as in step *4*. But this time we only have to add one copy of the digraph to fix all loops. See Fig. 4.7.

□

We have proved Lemma 8 by designing an algorithm that constructs digraphs whose full set of vertices is a (Σ, R)-set. See Fig. 4.8 for a (Σ, R)-set constructed by our algorithm where $a = 2$, $b = 5$ and $c = 3$.

Theorem 2. *Let σ^+ and σ^- be non-empty. Then the only (Σ, R)-set S in any digraph G is the empty set, if and only if $min(\sigma^+) > max(\sigma^-)$ or $max(\sigma^+) < min(\sigma^-)$.*

Proof. Theorem 2 almost follows from Lemma 7 and Lemma 8. The only thing that is missing is to state that if there is some $a \in \sigma^+$ equal to some $b \in \sigma^-$, then there clearly exists a digraph containing a (Σ, R)-set which is a clique. □

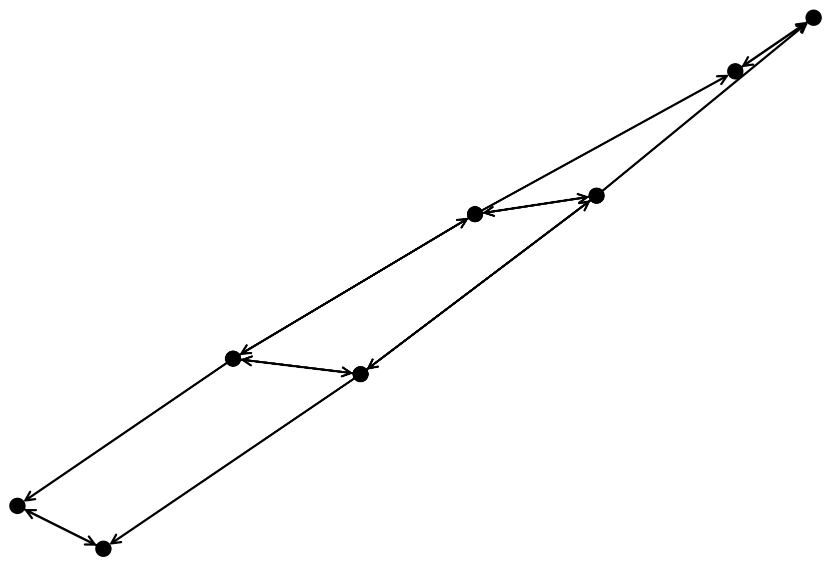


Figure 4.8: $a = 2, b = 5, c = 3$

4.3 Trivial bi-neighbourhood constraint matrices

In this section, we try to come up with some similar properties of bi-neighbourhood constraint matrices as for the trivial (Σ, R)-sets. We start by only considering the diagonal of the matrix.

4.3.1 The diagonal entries

When considering the diagonal entries of a bi-neighbourhood constraint matrix, we ask ourselves the question if there exists any digraph that has a partitioning where partition class i is not empty. Note that, in this case, we only have to consider the requirements at entry $[i, i]$ in the matrix.

Lemma 9. *Given a degree constraint matrix D_q and the problem of paritioning some digraph G into $V_1, ..., V_q$ satisfying a bi-neighbourhood constraint matrix D_q. Let $D[i, i] = (\mu_i^+, \mu_i^-)$. Then for any $1 \leq i \leq q$, there exists no partitioning of G such that $V_i \neq \emptyset$ if $min(\mu_i^+) > max(\mu_i^-)$ or $max(\mu_i^+) < min(\mu_i^-)$*

Proof. This follows directly from Theorem 2. □

Lemma 9 actually classifies a lot of the bi-neighbourhood constraint matrices where some of the partition classes will always be the empty set. In addition, if some partition class i is always the emptyset and another partition class $j \neq i$, require neighbours into partition class i, then V_j will also always be the the empty set.

4.3.2 The off-diagonal entries

Now we consider the off-diagonal entries. Although the diagonal of the degree constraint matrix loosely resembles the Σ requirements in LCVS problems, the off-diagonal entries are a bit more complex than the R requirements. To satisfy the R requirements we could just greedily increase the size of the *current* (Σ, R)-set by adding more copies of it. The problem we run into for partitioning problems is that we might have requirements for the neighbourhood between any of the partition classes in the partitioning. If we were to use the same algoritmic approach as in the previous section to construct partitions, we would quickly run into *circular dependencies* among the partition classes. Therefore, we leave it as an open problem to classify all trivial bi-neighbourhood constraint matrices.

Chapter 5

Hardness

As mentioned in the motivation, there are many LCVS and LCVP problems that have been studied extensively on undirected graphs. Many similar problems can be expressed for digraphs, so it would be useful to know if some of the hardness properties of the problems on undirected graphs are also true for the problems on digraphs.

5.1 DLCVS problems

Our first theorem shows that when we have equal requirements for neighbours in both directions, the directed problem is at least as hard as the undirected one.

Theorem 3. *Given an LCVS problem expressed by σ and ρ, and a DLCVS problem expressed by $\Sigma = (\sigma^+, \sigma^-)$ and $P = (\rho^+, \rho^-)$. If $\sigma = \sigma^+ = \sigma^-$, $\rho = \rho^+ = \rho^-$ and the LVCS problem is NP-hard, then the DLCVS problem is NP-hard as well.*

Proof. We prove this by a polynomial time reduction from the LCVS problem with input graph G to the DLCVS problem. Construct a digraph D that is the biorientation of G. If we show that a subset S in G is a (σ, ρ)-set if and only if it is a (Σ, P)-set in D, we are done.

- $\Rightarrow$ Let S be a (σ, ρ)-set in G. Now, consider the set of vertices S in D. We have to show that for every vertex v in D, $|N_D^+(v) \cap S| \in \sigma^+$ if $v \in S$ and $|N_D^+(v) \cap S| \in \rho^+$ if $v \notin S$. This follows from the fact that $N_G(v) = N_D^+(v)$ and $\sigma = \sigma^+$. The same argumentation holds for in-edges.

$\Leftarrow$ Let S be a (Σ, R)-set in D. Now, consider the set of vertices S in G. We have to show that for every vertex v in G, $|N_G(v) \cap S| \in \sigma$ if $v \in S$ and $|N_G(v) \cap S| \in \rho$ if $v \notin S$. This follows from the fact that $N_D^+(v) = N_D^-(v) = N_G(v)$ and $\sigma = \sigma^+ = \sigma^-$.

□

The next theorem gives a more general promise. The idea is that we only require one of σ^+ and σ^- to be equal to σ and one of ρ^+ and ρ^- to be equal to ρ. Imagine that we require $\sigma^+ = \sigma$. In this case, we also require σ^+ to be a subset of σ^- (similarly, if $\rho^+ = \rho$, then we require $\rho^+ \subseteq \rho^-$, or the opposite direction). The reason that we need this will be clear from the proof of the next theorem.

Theorem 4. *Given an LCVS problem expressed by σ and ρ, and a DLCVS problem expressed by $\Sigma = (\sigma^+, \sigma^-)$ and $P = (\rho^+, \rho^-)$. If any of the statements below are true and the LCVS problem is NP-hard, then the DLCVS problem is NP-hard as well.*

1. $\sigma = \sigma^+, \rho = \rho^+, \sigma^+ \subseteq \sigma^-, \rho^+ \subseteq \rho^-$
2. $\sigma = \sigma^+, \rho = \rho^-, \sigma^+ \subseteq \sigma^-, \rho^- \subseteq \rho^+$
3. $\sigma = \sigma^-, \rho = \rho^+, \sigma^- \subseteq \sigma^+, \rho^+ \subseteq \rho^-$
4. $\sigma = \sigma^-, \rho = \rho^-, \sigma^- \subseteq \sigma^+, \rho^- \subseteq \rho^+$

Proof. We prove this by a polynomial time reduction from the LCVS problem with input graph G to the DLCVS problem. Construct a digraph D that is the biorientation of G. If we show that a subset S in G is a (σ, ρ)-set if and only if it is a a (Σ, P)-set in D, we are done. We only consider the case where statement 2. is true, but the same reasoning can be extended easily to the other statements.

$\Rightarrow$ Let S be a (σ, ρ)-set in G. Now, consider the set of vertices S in D. We have to show that for every vertex v in D, $|N_D^+(v) \cap S| \in \sigma^+$ if $v \in S$ and $|N_D^+(v) \cap S| \in \rho^+$ if $v \notin S$. The first case holds because $|N_D^+(v) \cap S| = |N_G(v) \cap S| \in \sigma = \sigma^+$, and the second case holds because $|N_D^+(v) \cap S| = |N_G(v) \cap S| \in \rho = \rho^- \subseteq \rho^+$. We also have to show that for every vertex v in D, $|N^-(v) \cap S| \in \sigma^-$ if $v \in S$ and $|N^-(v) \cap S| \in \rho^-$ if $v \notin S$. The first case holds because $|N^-(v) \cap S| = |N_G(v) \cap S| \in \sigma = \sigma^+ \subseteq \sigma^-$, and the second case holds because $|N^-(v) \cap S| = |N_G(v) \cap S| \in \rho = \rho^-$.

$\Leftarrow$ Let S be a (Σ, R)-set in D. Now, consider the set of vertices S in G. We have to show that for every vertex v in G, $|N_G(v) \cap S| \in \sigma$ if $v \in S$ and $|N_G(v) \cap S| \in \rho$ if $v \notin S$. Note that $N_G(v) \cap S = N_D^+(v) \cap S = N_D^-(v) \cap S$ because D is the biorientation of G. In addition to this, the first case holds because because $\sigma = \sigma^+$, and the second case holds because $\rho = \rho^-$. The other directions holds by similar arguments.

□

This theorem makes it easy to reason about the hardness of DLCVS problems inspired by LCVS problems. It holds for existence, as well as maximization and minimization. Take for instance, dominating set (on undirected graphs). Because minimum dominating set is NP-complete, we also have that minimum dominating set (on digraphs), minimum absorbing set and minimum total dominating set are NP-complete on digraphs.

5.2 DLCVP problems

The ideas from the previous section are also applicable to DLCVP problems. We start by showing that DLCVP existence problems with fixed bi-neighbourhood constraint matrices are at least as hard as similar LCVP problems.

5.2.1 DLCVP existence

Recall the definition of a *degree constraint matrix* from Definition 3 and a *bi-neighbourhood constraint matrix* from Definition 4. We now give a theorem that is similar to Theorem 4, but for DLCVP existence problems.

Theorem 5. *Given a LCVP problem on an undirected graph expressed by a degree constraint matrix U, and a DLCVP problem on a digraph expressed by a bi-neighbourhood constraint matrix V. If for all i, j, one of the statements below is true, then if the LCVP problem is NP-hard, the DLCVP problem is NP-hard as well. Let $\chi_{i,j}$ denote $U[i,j]$ and let $V[i,j]$ consist of $(\mu^+_{i,j}, \mu^-_{i,j})$.*

1. $\chi_{i,j} = \mu^+_{i,j} \wedge \mu^+_{i,j} \subseteq \mu^-_{i,j}$

2. $\chi_{i,j} = \mu^-_{i,j} \wedge \mu^-_{i,j} \subseteq \mu^+_{i,j}$

Proof. The proof is almost identical to the proof given for Theorem 4. Similar to the other proof, we prove this by a polynomial time reduction from the LCVP problem with input graph G to the DLCVP problem. Construct a digraph D that is the biorientation of G. We only consider the case where one of the statements is true, statement 1., but the same reasoning can be extended easily to the other statement.

$\Rightarrow$ Let $P = (p_1, ..., p_n)$ be an U-partition of G. Now consider the same U-partition in D. We need to show that for all i, j and $v \in p_i$, $|N^+_D(v) \cap p_j| \in \mu^+_{i,j}$ and $|N^-_D(v) \cap p_j| \in \mu^-_{i,j}$ for P to be a V-partition in D. The first case is true because $|N^+_D(v) \cap p_j| = |N_G(v) \cap p_j| \in \chi_{i,j} = \mu^+_{i,j}$, and the second case is true because $|N^-_D(v) \cap p_j| = |N(v) \cap p_j| \in \chi_{i,j} = \mu^+_{i,j} \subseteq \mu^-_{i,j}$.

$\Leftarrow$ The other direction holds because $N(v) = N^+(v) = N^-(v)$.

$\square$

Chapter 6

Directed covering graphs

6.1 Introduction

In our last chapter, we explore and try to visualize the structure of covering digraphs generated from a set of small digraphs with simple structure. We start by giving the formal definition of covering graphs.

Definition 5. *[7] A digraph G is called a covering of a digraph H with the covering projection $p : G \to H$ if p is a surjection from $V(G)$ to $V(H)$ such that p maps the set of edges starting (and ending, respectively) at x' bijectively to the set of edges starting (and ending, respectively) at x for any vertex $x \in V(H)$ and $x' \in p^{-1}(x)$.*

The attractive property of a digraph G covering another digraph H is that the structure of G usually looks way more complicated than the structure of H. In most cases, one would try to find such underlying structures to exploit them to solve some problem. Instead, we do the opposite to understand more about such digraphs. In other words, we take some digraph H and generate a new digraph G, such that G covers H.

Graph construction, or more specifically, in our case, constructing large digraphs from smaller digraphs, is one approach that can be used to gain more insight into different problems; we refer to Miller and Sirán [15], a survey of the degree/diamater problem. In the context of coverings, we also refer to Ždímalová and Staneková [25], that, amongst other things, show that one of the graph constructions from Comellas and Fiol [8] preserves coverings. Our construction will also preserve the covering property.

The H-covering problem can have polynomial-time solutions or be NP-complete depending on the graph H. However, our interest in covering graphs, as with vertex partitioning problems, is again, that some of them are hard to solve. We refer to [22], where Telle shows the NP-completeness of H-covering problems for several infinite classes of graphs.

6.2 Degree constraint matrices

Before we discuss how to generate such digraphs, we note that covering graphs (both undirected and directed) fit pretty well into the concept of *degree constraint matrices*, which we discussed earlier. Suppose we construct a degree constraint matrix from the adjacency matrix of some graph H. Then the problem of finding a partitioning of some graph G that satisfies the matrix is equivalent to the problem of deciding if the G covers H. Below is the *degree constraint matrix* constructed from the undirected graph in Fig. 2.1. If two vertices, i and j, are adjacent in the graph, we put $\{1\}$ at entry $[i, j]$ in the matrix, and otherwise $\{0\}$. Note that degree constraint matrices work for both undirected and directed H. Thus, when H is an undirected graph, the matrix will always be symmetrical.

$$\begin{bmatrix} \{0\} & \{1\} & \{0\} \\ \{1\} & \{0\} & \{1\} \\ \{0\} & \{1\} & \{0\} \end{bmatrix}$$

6.3 Construction

Lemma 10. *Let H be a connected, simple digraph and let G be an H-covering. Consider the partition of G defined by the degree constraint matrix obtained from the adjacency matrix of H. There are equally many vertices in each partition class of the partitioning.*

Proof. Remember that H is connected. Because G has to be locally isomorphic to H, we need one vertex from each parition class to satisfy any vertex of G. We can not reuse a vertex to satisfy multiple vertices in the same partition class as this would break the local isomorphism. Therefore, G can only consist of multiple sets of vertices, where each set of vertices consist of one vertex from each partition class. □

Algorithm 1. *Our approach to generate a digraph G from some digraph H can be summarized as:*

1. *Assign unique colors to the vertices of H.*
2. *Decide on some positive integer k.*
3. *Add k vertices to G for each of the colors in H.*
4. *Add edges between the vertices in G of colors that are adjacent in H according to the python-implementation listed in Section 8.1 (we put our main focus on the vizualisations, so we do not go into the details of this)*

6.4 Vizualisation

We will use the three digraphs depicted in Fig. 6.1 as H-digraphs for our algorithm; a *three-cycle*, *a time-glass* and a *cross*. We start by choosing values of k such that we get around 25 vertices in the generated digraphs. So, for the three-cycle, we choose $k = 7$, for the time-glass, $k = 5$, and for the cross, $k = 4$.

See Fig. 6.2, Fig. 6.3 and Fig. 6.4 for the random digraphs we generated. They are vizualised with the Fruchterman-Reingold force-directed algorithm [9] (implemented in *networkx* [16]). Note that the colors in the figures correspond to which partition class each vertex belongs to. Even though these digraphs look somewhat chaotic (maybe except for Fig. 6.2), they have very simple underlying structures. We see some of this structure in the figures because of the colors, but observeing this structure without the colors is clearly a lot harder.

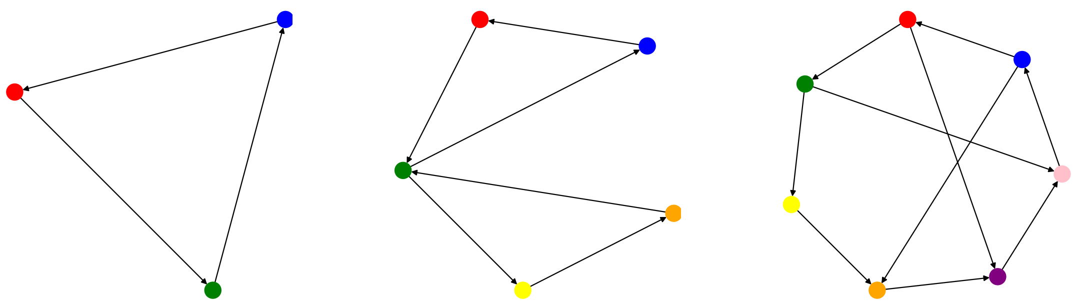
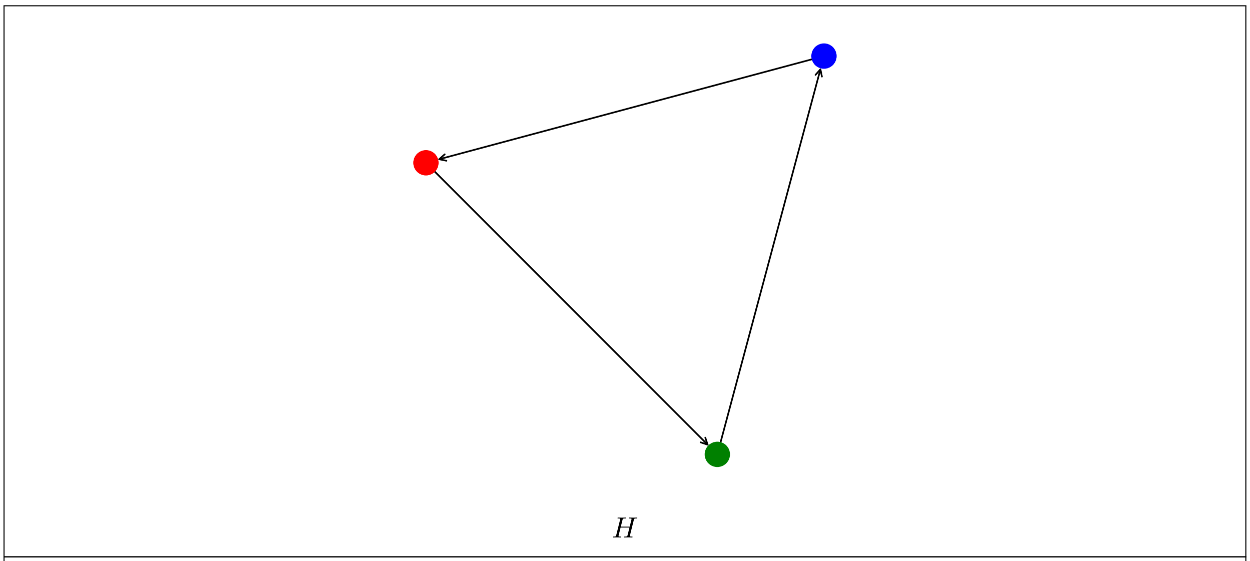


Figure 6.1: Example digraphs (a *three-cycle*, a *time-glass* and a *cross*)

We vizualise the same digraphs. But now we cluster together vertices that belong to the same partition class, put otherwise, the vertices that are locally isomorphic to the same vertex in H. We organize these vertices in circles and put the center of the circle at the same offset as where the corresponding vertices in H are in Fig. 6.1. See Fig. 6.5, Fig. 6.6 and Fig. 6.7. We can now observe the underlying digraph H very easily.

Finally, we vizualise the digraphs on a grid, such that each horizontal grid line corresponds to each parition class of the partitioning. An interesting property becomes very clear. Although, this might have been clear in previous vizualisations as well, we now see that the parition classes, that correspond to the vertex mapping, are independent sets in G. See Fig. 6.8, Fig. 6.9 and Fig. 6.10.



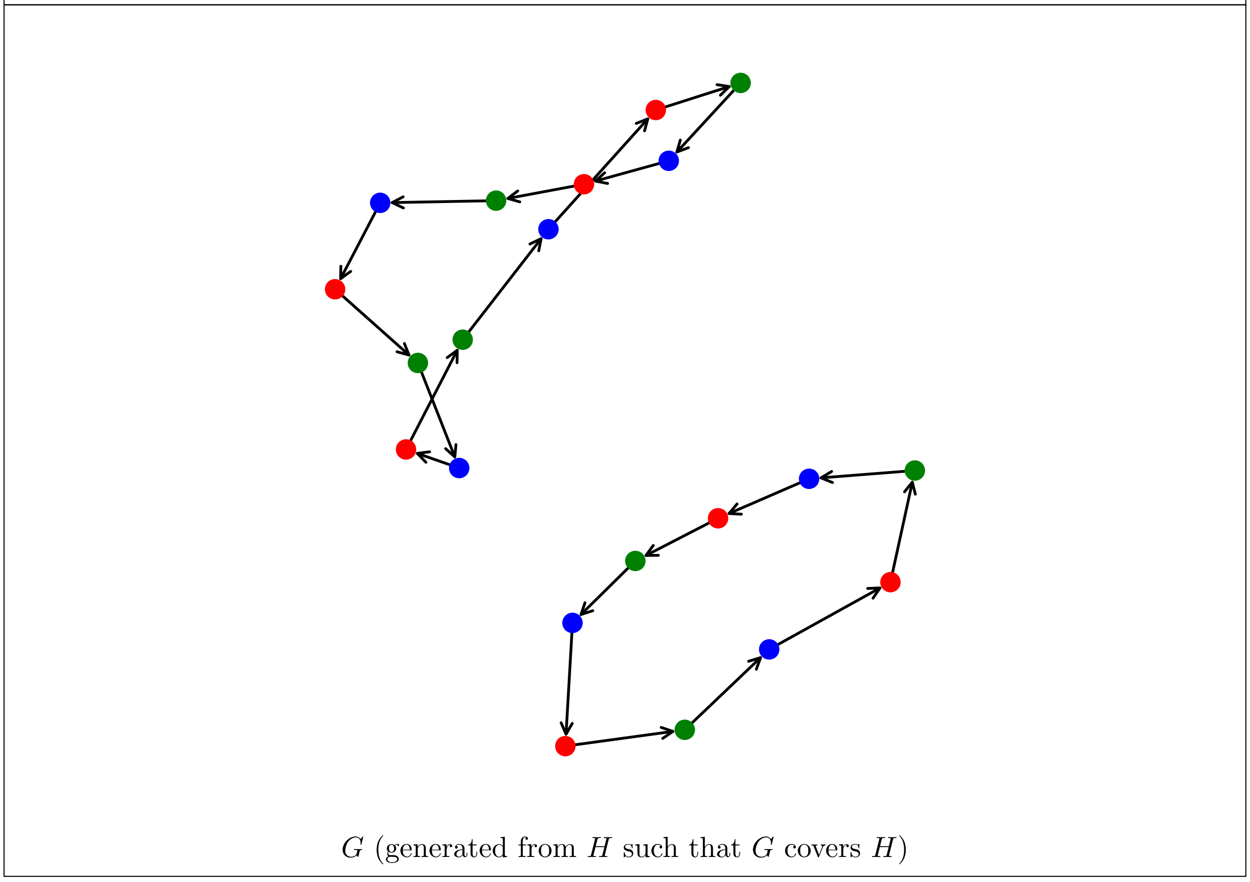


Figure 6.2: All digraphs G that cover some digraph H that is a cycle, also has to be cyclic. In this case, our algorithm generated a digraph G consisting of two seperate components that are cycles.

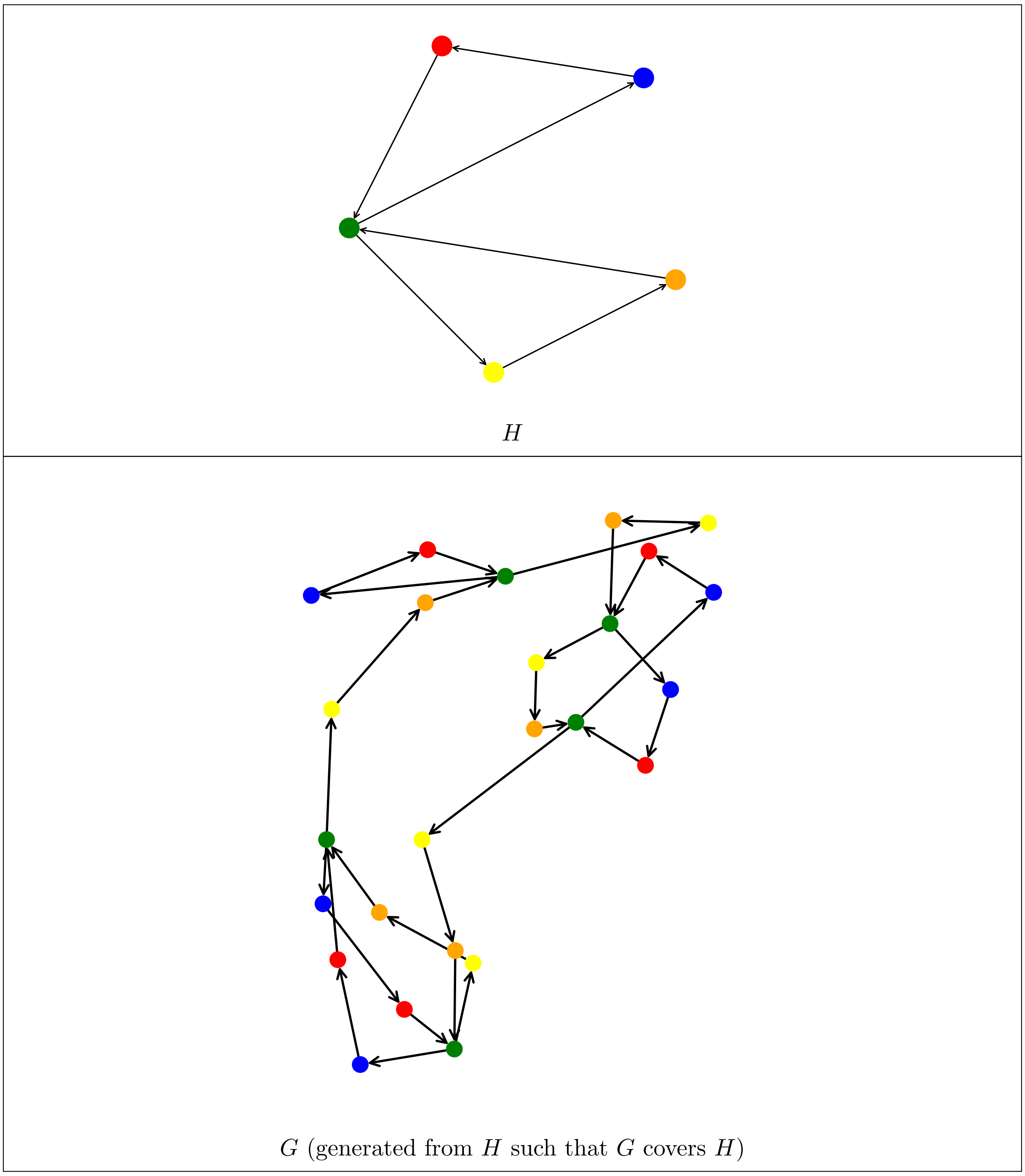


Figure 6.3: The *time-glass* digraph, similarly to the *three-cycle* digraph, is also kind of cyclic. We can observe this in the digraph G our algorithm generated. We find the *red-green-blue* and *green-yellow-orange* cycles in various shapes and lengths.

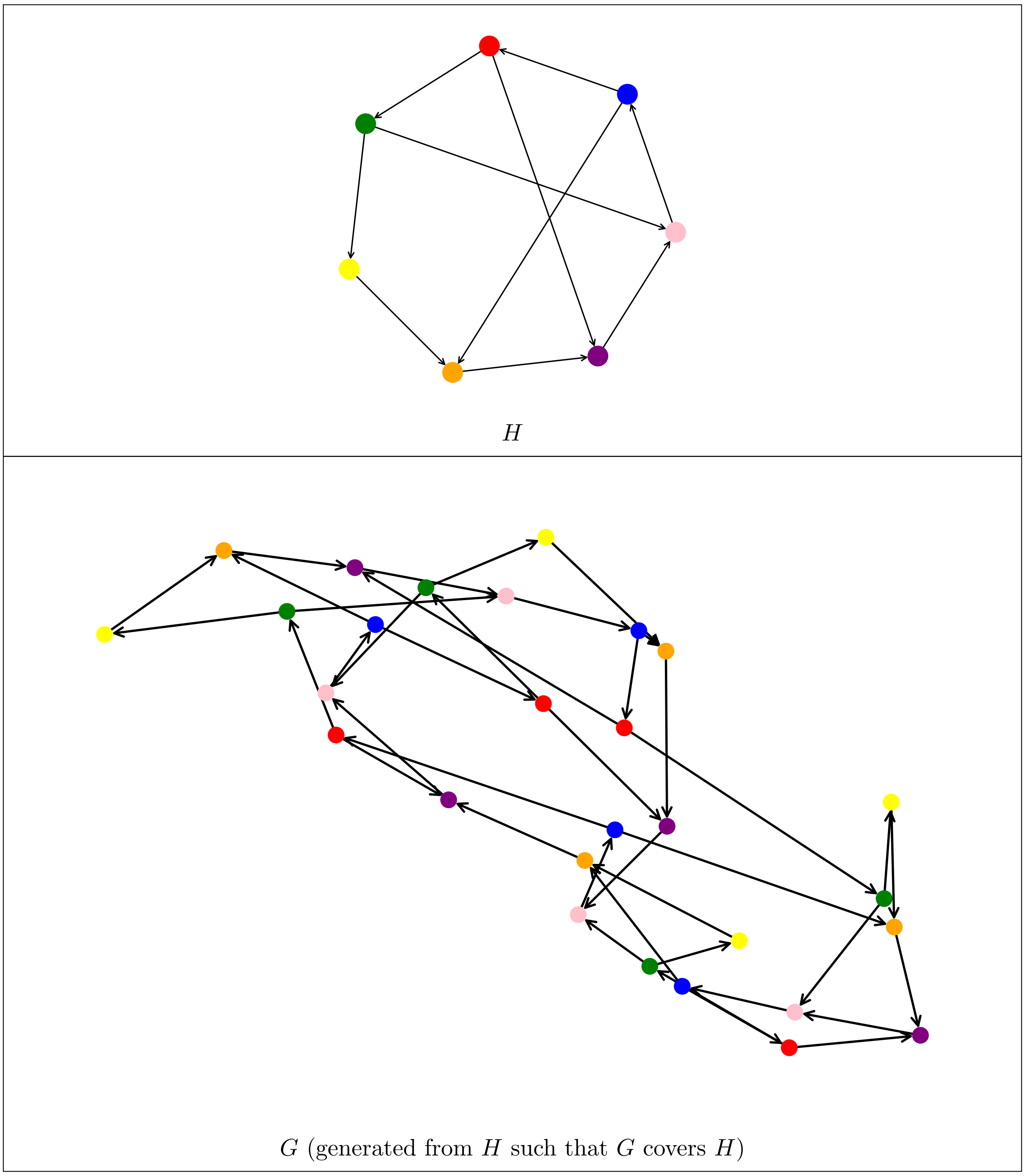


Figure 6.4: The *cross* digraph is also kind of cyclic. However, the cycle is longer, and we have three extra edges *crossing* the cycle. While H still has a fairly simple structure, it is now much harder to observe this structure in the generated digraph.

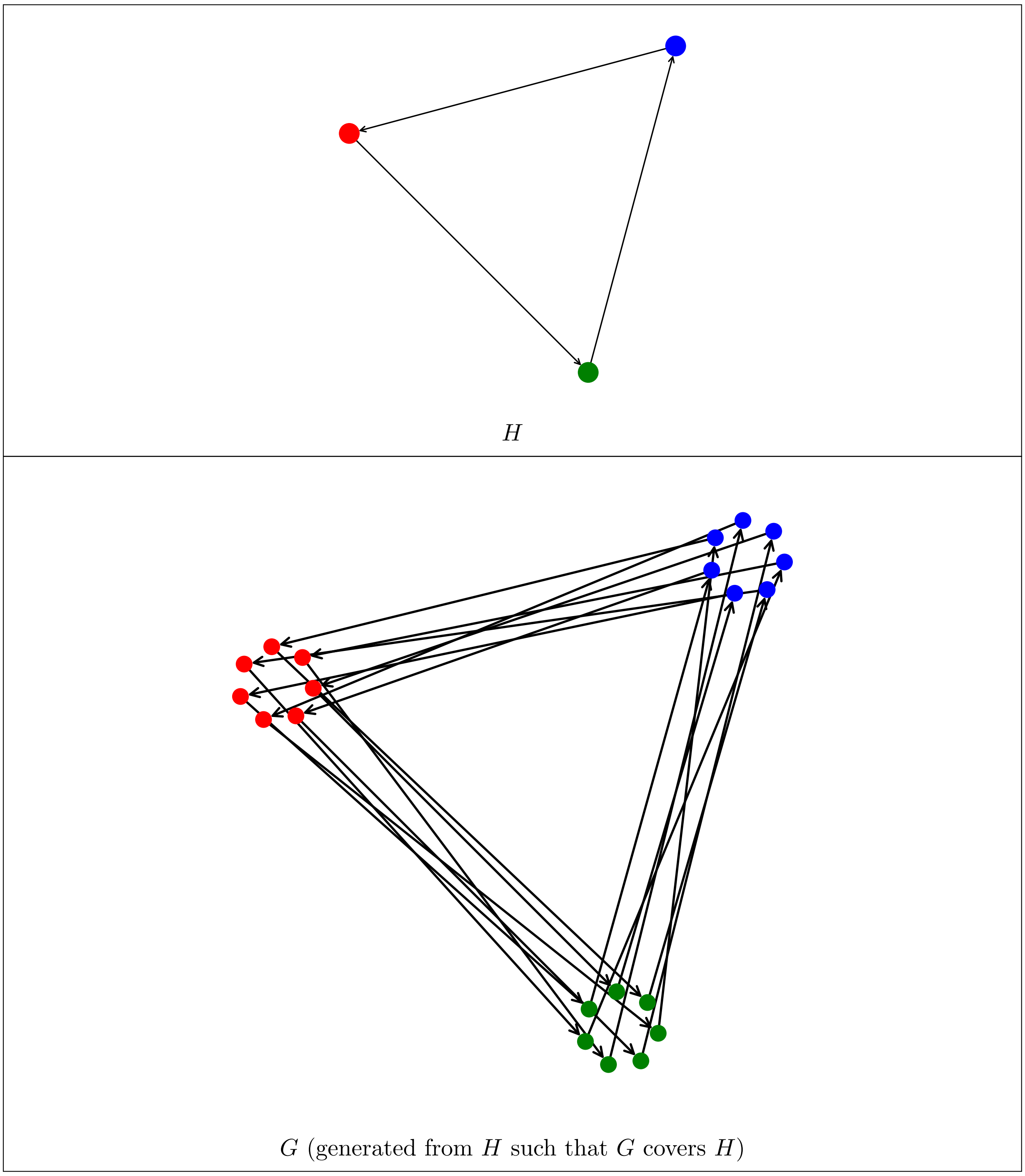


Figure 6.5: Clustering together the vertices in G that are mapped to the same vertices in H makes it easy to see that G is a H-covering; both in terms of the structure and the direction of the edges.

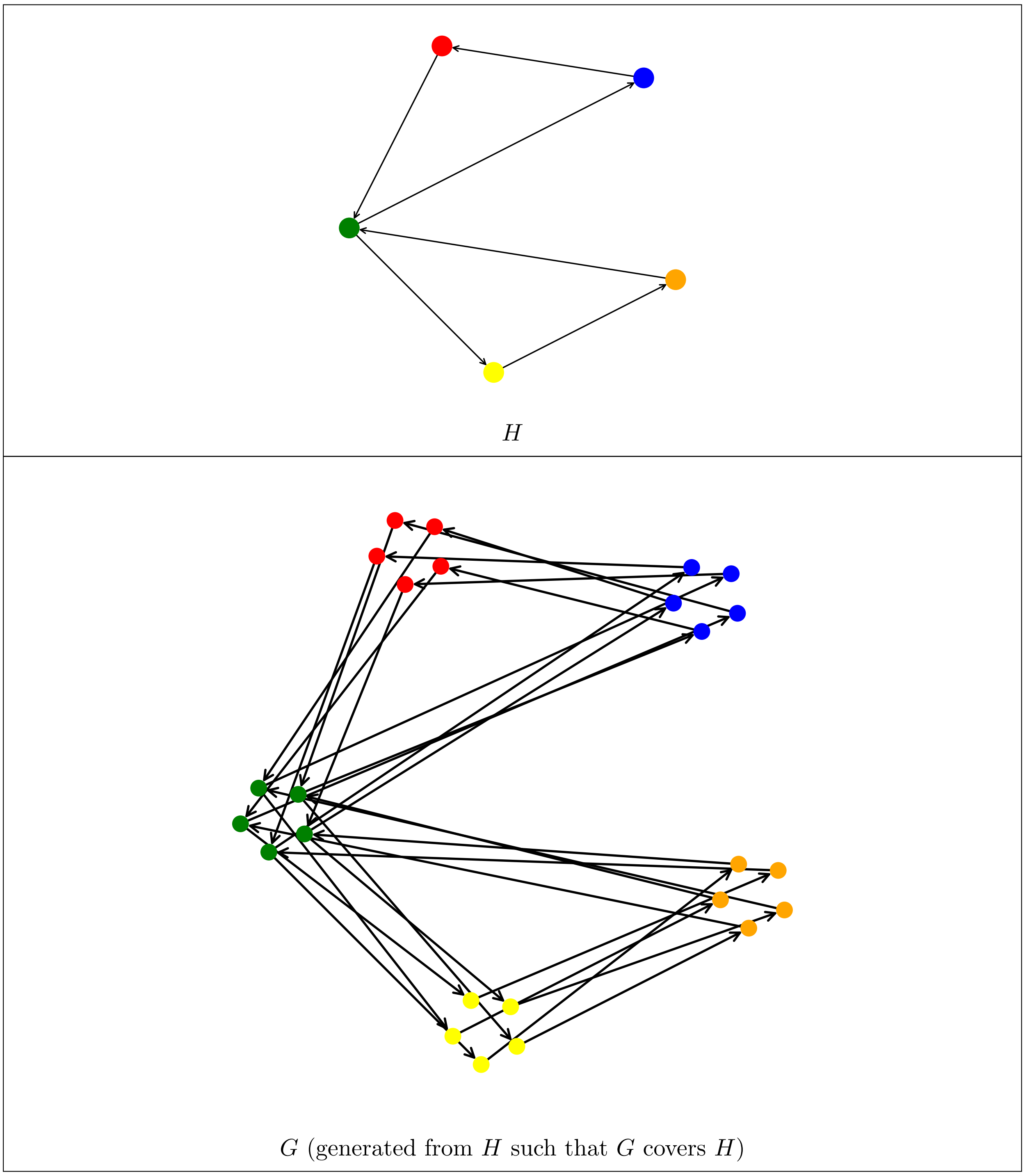
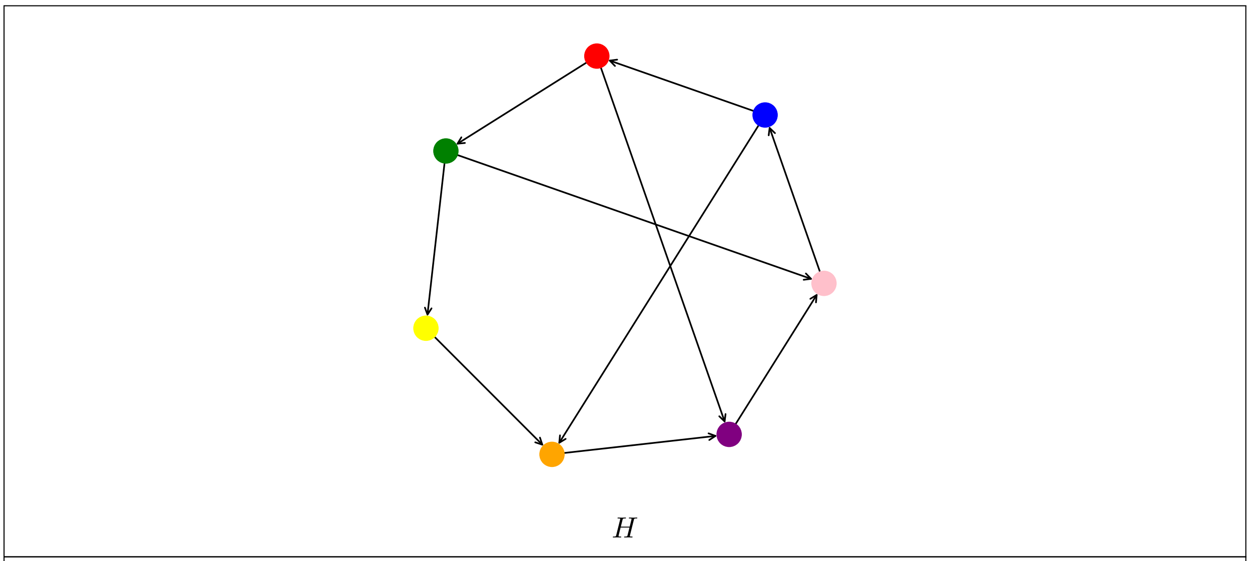


Figure 6.6: Clustering together the vertices in G that are mapped to the same vertices in H makes it easy to see that G is a H-covering; both in terms of the structure and the direction of the edges.



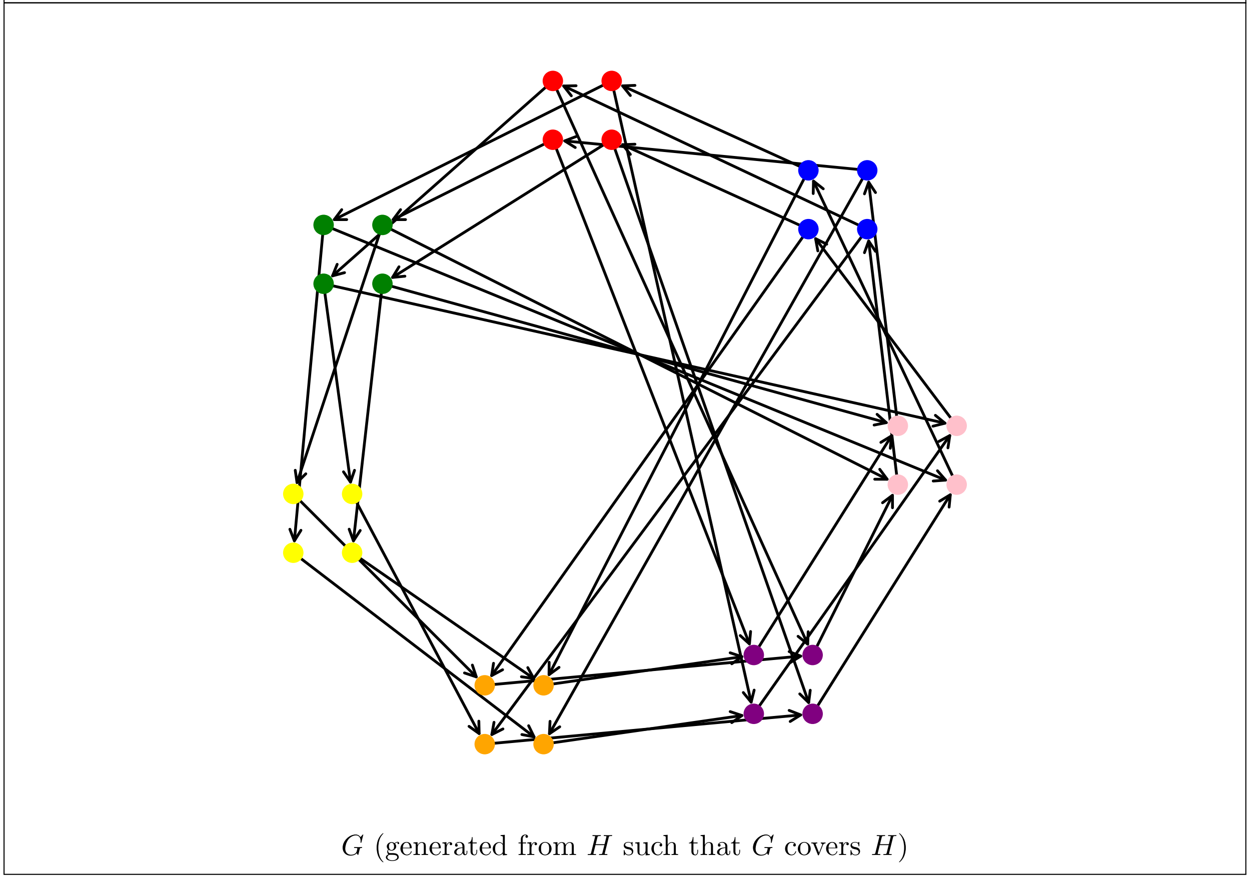
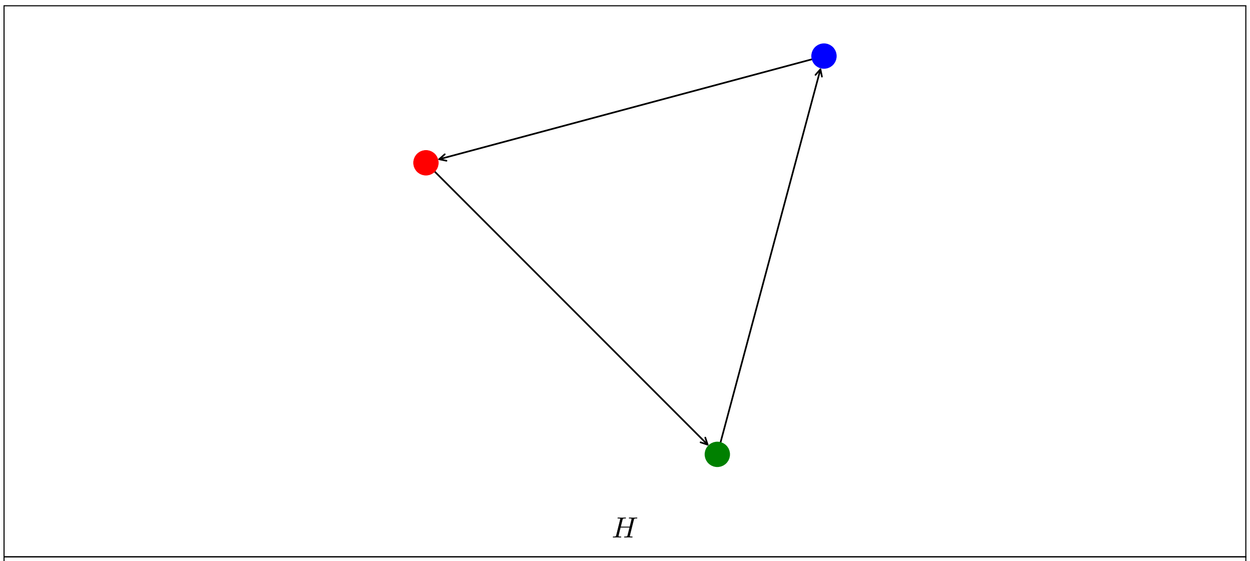


Figure 6.7: Clustering together the vertices in G that are mapped to the same vertices in H makes it easy to see that G is a H-covering; both in terms of the structure and the direction of the edges.



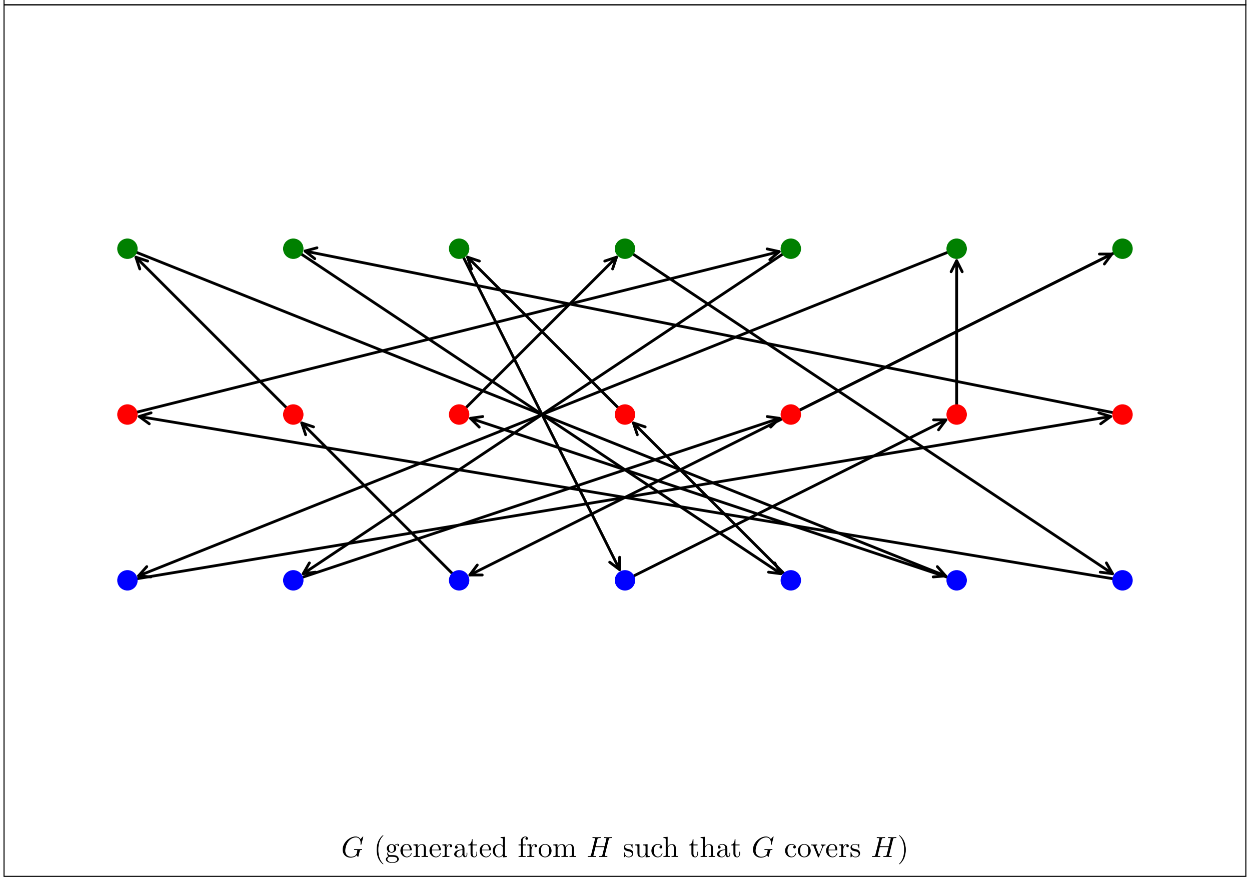
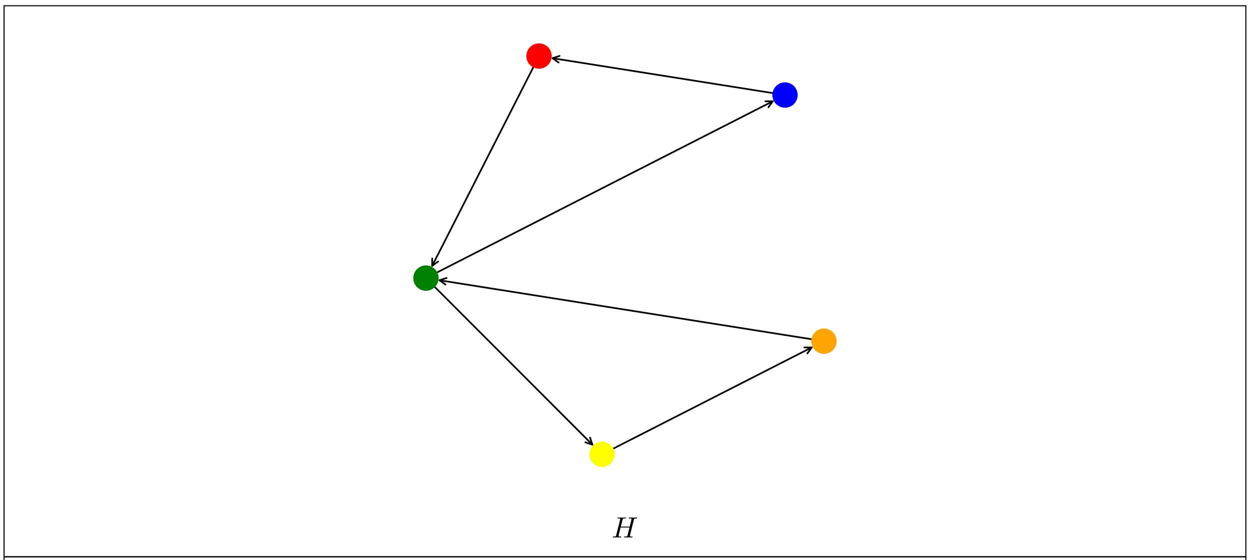


Figure 6.8: It is still very managable to see the cycle-pattern in the digraph G generated from H. In addition, the fact that the partition classes are independent sets, is emphasized when grouping them together on each row in the grid.



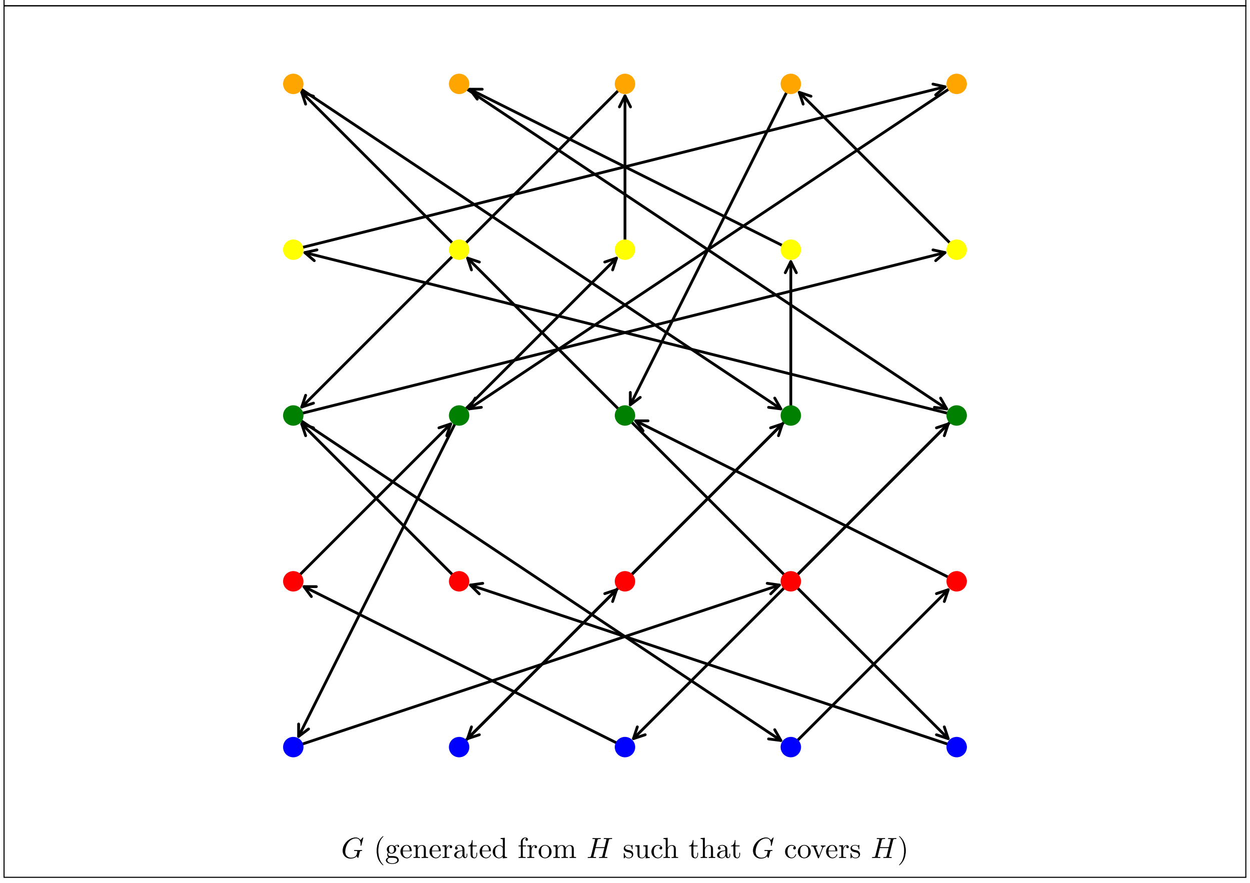
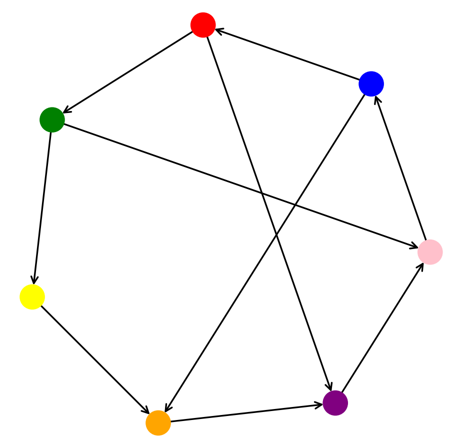


Figure 6.9: Note that the two cycles in H are grouped together such that the vertices corresponding to the *green-yellow-orange* cycle are on the first three rows and the vertices corresponding to the *green-blue-red* cycle are on the last three rows. We see that there are no edges going from the bottom of the grid, past the green row, to the top of the grid. The green vertices *cut* the graph in two pieces.



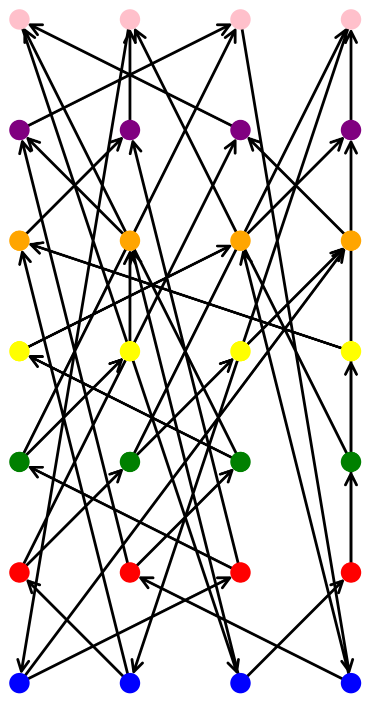


Figure 6.10: Because the rows in G are ordered in the order of the cycle in H, we see that most of the edges point *upwards*. However, again, the length of the cycle and the crossing edges in H makes it difficult to interpret the graph even on a grid.

Chapter 7

Conclusions

In this thesis, we have made several small contribution regarding properties of locally checkable vertex partitioning problems on digraphs.

We started by giving a lemma for when two vertex subsets must have the same cardinality, Lemma 2, inspired by similar properties of undirected graphs. Though, we where not able to give as strong a claim as Theorem 1 from [22]. We leave this as an open problem.

Then we moved on to explore trivial classes of (Σ, R)-problem definitions in Lemma 3, Lemma 4, Lemma 5, Lemma 6, Lemma 7, Lemma 8 and finally Theorem 2. We also constructed an algorithm in the proof of Lemma 8, and gave an implementation of it in Section 8.2. We tried to do the same for trivial *bi-neighbourhood constraint matrices*, in Lemma 9, but where not able to classify as many as for the (Σ, R)-problems. Therefore, we leave as an open problem, to classify all trivial degree constraint matrices, both for undirected graphs and digraphs.

Furhtermore, we gave Theorem 4 and Theorem 5 that lets us easily observe the hardness (*NP-completeness*) of directed problems that are similar to undirected ones.

Finally, we constructed an algorithm, Algorithm 1 (with its implementation in Section 8.1), that takes as input a digraph H, and constructs a larger digraph G that covers H. We vizualised some of the graphs generated with the algorithm to get a deeper understanding of covering digraphs.

Bibliography

[1] S. Arumugam, K. Jacob, and Lutz Volkmann. Total and connected domination in digraphs. *Australian Journal of Combinatorics*, 39, 2007. 6

[2] Jørgen Bang-Jensen and Gregory Gutin. *Classes of directed graphs*. Springer, 2018. 11

[3] David W. Bange, Anthony E. Barkauskas, Linda H. Host, and Lane H. Clark. Efficient domination of the orientations of a graph. *Discrete Mathematics*, 178(1):1–14, Jan 1998. doi: 10.1016/S0012-365X(97)81813-4. 6

[4] E. Boros and V. Gurvich. Perfect graphs, kernels, and cores of cooperative games. *Discrete Mathematics*, 306(19):2336–2354, 2006. 12

[5] Domingos M. Cardoso, Marcin Kamiński, and Vadim Lozin. Maximum k-regular induced subgraphs. *Journal of Combinatorial Optimization*, 14(4):455–463, Nov 2007. doi: 10.1007/s10878-007-9045-9. 6

[6] Michael Cary, Jonathan Cary, and Savari Prabhu. Independent domination in directed graphs. *arXiv:1909.05121 [cs, math]*, Sep 2019. arXiv: 1909.05121. 6

[7] Young-Bin Choe, Jin Ho Kwak, Yong Sung Park, and Iwao Sato. Bartholdi zeta and l-functions of weighted digraphs, their coverings and products. *Advances in Mathematics*, 213(2):865–886, Aug 2007. doi: 10.1016/j.aim.2007.01.013. 26

[8] F. Comellas and M. A. Fiol. Vertex-symmetric digraphs with small diameter. *Discrete Applied Mathematics*, 58(1):1–11, Mar 1995. doi: 10.1016/0166-218X(93)E0145-O. 26

[9] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164, 1991. 28

[10] Yumin Fu. Dominating set and converse dominating set of a directed graph. *The American Mathematical Monthly*, 75(8):861–863, 1968. doi: 10.2307/2314337. 6

[11] Chartrand G. Twin domination in digraphs. *Ars Combin.*, 67:105–114, 2003. 6

[12] J. Ghoshal, R. Laskar, and D. Pillone. *Topics on Domination in Directed Graphs.* Routledge, 1998. ISBN 9781315141428. 6

[13] Pinar Heggernes and Jan Telle. Partitioning graphs into generalized dominating sets. *Nordic J. Comput.*, 5, Sep 2001. 7

[14] Lars Jaffke, O-joung Kwon, and Jan Arne Telle. Classes of intersection digraphs with good algorithmic properties, 2021. 2, 5, 6, 8, 9

[15] Mirka Miller and Jozef Sirán. Moore graphs and beyond: A survey of the degree/-diameter problem. *The Electronic Journal of Combinatorics*, pages DS14: May 16–2013. doi: 10.37236/35. 26

[16] Networkx. Networkx home page, May 2022.
URL: `https://networkx.org`. 28

[17] Richard J. Nowakowski. *Games of No Chance.* Cambridge University Press, 1994. 11

[18] Lyes Ouldrabah, Mostafa Blidia, and Ahmed Bouchou. On the k-domination number of digraphs. *Journal of Combinatorial Optimization*, 38(3):680–688, Oct 2019. doi: 10.1007/s10878-019-00405-1. 6

[19] Amina Ramoul and Mostafa Blidia. A new generalization of kernels in digraphs. *Discrete Applied Mathematics*, 217:673–684, Jan 2017. doi: 10.1016/j.dam.2016.09.048. 6

[20] Oliver Schaudt. Efficient total domination in digraphs. *Journal of Discrete Algorithms*, 15:32–42, Aug 2012. doi: 10.1016/j.jda.2012.02.003. 6

[21] JA Telle. Characterization of domination-type parameters in graphs. *Congressus Numerantium*, pages 9–9, 1993. 1, 5

[22] Jan Arne Telle. *Vertex Partitioning Problems: Characterization, Complexity and Algorithms on Partial k-Trees.* PhD thesis, University of Oregon, 1994. 2, 5, 7, 13, 15, 27, 39

[23] Martin Vatshelle. *New Width Parameters of Graphs.* PhD thesis, University of Bergen, 2012. 9

[24] John von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior.* Princeton University Press, 1953. 6, 11

[25] Mária Ždímalová and Lubica Staneková. Large digraphs of given diameter and degree from coverings. page 373–378. Iniciativa Digital Politècnica, 2011. ISBN 9788476535653. 26

Chapter 8

Appendices

8.1 Implementation of Algorithm 1

```python
import matplotlib.pyplot as plt

def random_covering_map(covering_graph, k, directed=False):
    nn, edges = covering_graph

    new_nn = k * nn
    new_edges = set()

    for u in range(new_nn):
        original_u = original_node(u, nn)
        original_adj = adj(original_u, edges, directed)

        for a in original_adj:
            for _k in np.random.permutation(k):
                v = a + (_k * nn)
                if valid_edge(u, v, k, nn, new_edges, directed):
                    new_edges.add((u, v))

    return (new_nn, list(new_edges))

def valid_edge(u, v, k, nn, new_edges, directed):
    original_v = original_node(v, nn)
    for _k in range(k):
        if (u, original_v + (_k * nn)) in new_edges:
            return False
        if not directed and (original_v + (_k * nn), u) in new_edges:
            return False

    original_u = original_node(u, nn)
    for _k in range(k):
        if (original_u + (_k * nn), v) in new_edges:
            return False
        if not directed and (v, original_u + (_k * nn)) in new_edges:
            return False

    return True

def original_node(u, nn):
    return u % nn

def adj(u, edges, directed):
    adjacent = [v for _u, v in edges if _u == u]
    if not directed:
        adjacent += [_u for _u, v in edges if v == u]
    return list(set(adjacent))
```

8.2 Implementation of algorithm from Lemma 8

```python
def generate(_sigmap, _sigmam):
    valid, sigmap, sigmam = validate_sigma(_sigmap, _sigmam)

    if not valid:
        print("not possible to construct non-empty (", sigmap, ",", sigmam, ")-set")
        return

    if len(sigmap) == 1 and len(sigmam) == 1:
        edges = clique(sigmap[0] + 1)
        draw_graph(sigmap[0], edges)
        return

    flip = len(sigmam) == 2
    if flip:
        sigmap, sigmam = sigmam, sigmap

    a, b = sigmap[0], sigmap[1]
    c = sigmam[0]
    n = abs(a-b)
    x = int(n * ((c - b) / (a - b)))
    out_capacities = [b for _ in range(n-x)] + [a for _ in range(x)]
    in_capacities = [c for _ in range(n)]

    edges = []
    n, edges = greedy(out_capacities, in_capacities, edges)
    n, edges = fix_multi_edges(n, edges)
    n, edges = fix_loops(n, edges)

    if flip:
        edges = [(v, u) for v, u in edges]

    return n, edges

def validate_sigma(sigmap, sigmam):
    # a = b, a \in \sigma^+, b \in \sigma^-
    for a in sigmap:
        for b in sigmam:
            if a == b:
                return True, [a], [b]

    # a < c < b, a,b \in \sigma^+, c \in \sigma^-
    for ai, a in enumerate(sigmap):
        for bi, b in enumerate(sigmap):
            if ai == bi:
                continue
            for c in sigmam:
                if a < c and c < b:
                    return True, [a, b], [c]

    # a < c < b, a,b \in \sigma^-, c \in \sigma^+
    for ai, a in enumerate(sigmam):
        for bi, b in enumerate(sigmam):
            if ai == bi:
                continue
            for c in sigmap:
                if a < c and c < b:
                    return True, [c], [a, b]

    return False, None, None

def greedy(out_capacities, in_capacities, edges):
    n = len(out_capacities)

    for i in range(n):
        while out_capacities[i] > 0:
            for j in range(n):
                if in_capacities[j] > 0:
                    edges.append([i, j])
                    in_capacities[j] -= 1
                    out_capacities[i] -= 1
                    break

    return n, edges

def fix_multi_edges(n, edges):
    max_multi_edge = most(edges)

    new_edges = []
    for m in range(1, max_multi_edge):
        new_edges += [[i + n*m, j + n*m] for [i, j] in edges]

    edges += new_edges

    for i in range(n):
        for j in range(n):

            multi_edge_count = edges.count([i, j])

            if multi_edge_count > 1:
                for m in range(max_multi_edge):
                    im, jm = i + m*n, j + m*n
                    for mm in range(1, multi_edge_count):
                        edges.remove([im, jm])
                        edges.append([im, (jm + mm*n) %
                            ↪ (max_multi_edge*n)])

    return n*max_multi_edge, edges

def fix_loops(n, edges):
    edges_copy = [[i + n, j + n] for [i, j] in edges]
    edges += edges_copy

    for i in range(n):
        if [i, i] in edges:
            edges.remove([i, i])
            edges.remove([i + n, i + n])
            edges.append([i, i + n])
            edges.append([i + n, i])

    return 2*n, edges

def clique(n):
    return list(sum([[[u, v] for v in range(n) if u != v] for u in
        ↪ range(n)], []))

def count(l, e):
    c = 0
    for em in l:
        if e == em:
            c += 1
    return c

def most(l):
    m = 0
    for e in l:
        m = max(m, count(l, e))
    return m
```