

Coloured graph models:

Associating emitters and ships



Anne Hillebrand



Corpus Christi College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy
Trinity 2015

Voor opa

12 mei 1914 – 23 augustus 2014

Acknowledgements

First and foremost I would like to thank my supervisor professor Colin McDiarmid for the guidance, encouragement and advice he has given me throughout my DPhil. Without him this work would have not been possible. I would also like to thank my other supervisor professor Alex Scott for all the insightful discussions and helpful suggestions concerning the algorithmics. A special thanks to my college advisor Dr. Paul Dellar for his kindness and continued support. I would like to express my great appreciation to Paul Davies for setting up my studentship. Advice given by Dr. Andrew Batchelor and Dr. Andy Stove has been a great help in establishing the particulars of the project. Assistance provided by Dr. Rebecca Gower, Kevin Wallace and Fernande van Schelle was greatly appreciated.

This dissertation would not have been possible without funding from the EPSRC and Thales UK under studentship BK/11/012.

I wish to acknowledge the help provided by Ross Atkins, Dr. Matthias Lenz and Suzanne Klaver for proofreading some chapters.

I would like to express my great appreciation to Corpus Christi College for being my home and community all these years and to the Mathematical Institute Oxford, for giving me a place to work.

I would like to thank all my mathematical friends; all those conferences, meetings, lunches and workshops would have not been the same without you.

Thanks to all my friends, whether old or new, close or far away, from Corpus, from Univ, or from somewhere completely different; you have made the past five years unforgettable.

Special thanks to Fiona Skerman for keeping me company (and sane!) in the office.

Dankjewel Marloes, dat twee jaar lang naar huis gaan echt thuis komen was.

Mijn grootste dank gaat uit naar mijn familie, voor hun onvoorwaardelijke steun en vertrouwen. Mama, jij hield (en houdt!) me op de hoogte hield van alle kleine en grote gebeurtenissen, die ik anders zou missen. Papa, bedankt voor de motivatie en de moed om door te gaan. Zusje, Lieke, bedankt voor al het enthousiasme, alle gezellige bezoeken, en voor 'holding down the fort' thuis. Broertje Joppe en Carlijn, het was super fijn dat jullie zo dichtbij woonden even en mijn thuis weg van thuis konden zijn.

Mike, trust me, for soon I too will be a doctor. You have been by my side for almost my entire DPhil. You always remind me to wonder at and about the world. Thanks for your love, always being you and for being everything that I am not. I wouldn't have wanted to do it without you.

En, last but not least, bedank ik mijn rode je-weet-wel-kater Jip voor alle slapeloze nachten, voor alle knuffels (vooral als je wilt slapen), voor alle problemen met burens en huisbazen, voor de vreugde als ik thuiskom (of als meneer denkt dat ie eten krijgt), en voor nog een knuffel. Gezellig samen thesis schrijven, gezellig samen tv-kijken, gezellig samen naar de keuken; gezellig samen alles. Afgelopen jaar vanuit Nederland, maar niet minder enthousiast.

PS: Christian Wehrenfennig, because I just wanted to show you that I do know how to spell your name!

Abstract

This thesis examines graph-theoretic approaches to problems arising in emitter-to-platform association, for example in associating emitters and ships. The first part of this thesis focuses on emitters that are observed by two different sensors, which can only determine the bearing the observed signal was emitted from. The aim is to pair observations from the two sensors, that originate from the same emitter. In this thesis it is shown that different types of observations can be represented as one of c colours and each observed bearing as a row or a column in an n by n grid. Given the horizontal and vertical projections of a coloured grid, the aim is to reconstruct the original layout of the coloured squares on the grid. Deciding whether two sets of positive natural numbers are the horizontal and vertical projections of a coloured grid is NP-complete for two or more colours [24, 35, 47] and has close connections to colour degree matrix problems.

Necessary and sufficient conditions are known for a demand matrix to be a colour degree matrix of an edge-coloured forest [9, 22]. In this thesis the first step beyond forests is taken: necessary and sufficient conditions for a demand matrix to be realisable by a graph with at most one cycle are proved. As part of the proof some directly forced structures are discovered, that is, structures that must exist in every realisation. Moreover, corresponding results for multi-graphs and pseudoforests with at most k cyclic components are presented. This part concludes with $O(n^2)$ time algorithms to check these conditions and return a witness if one of the conditions is violated. Finally $O(n^3)$ time algorithms to find a realisation, if one exists, are described.

The second part of this thesis introduces the coloured L -model. This is an original idealised graph model, developed to explore the combinatorial properties of the emitter-to-platform association problem, referred to as the reverse radar problem in this thesis. The aim is to decide which groups of observed radars originate from the same ship, taking into account which combinations of radar models are known to be carried by different types of ships. It is shown what exactly makes finding a solution to this idealised version of the reverse radar problem NP-hard, and that there are tractable cases equivalent to finding (weighted) matchings in related graphs, which have bounded pathwidth. Restricting to graphs with bounded pathwidth is a reasonable simplification, as signals come in along bearings (i.e. along a cycle) and a cycle has pathwidth two. New algorithms to find (weighted) matchings in graphs with bounded pathwidth and treewidth are presented, which can be used to solve the reverse radar problem in this model.

Finally, the likelihood and two-time models are introduced, which complement the coloured L -model by generalising measurement errors and by introducing time. It is shown that the problem remains NP-hard, even for very simple cases and some tractable cases are described.

Contents

Contents	i
List of Figures	vii
List of Tables	ix
1 Introduction	1
1.1 The two-sensor problem	1
1.2 Reverse Radar	2
1.3 Outline	5
2 Preliminaries	7
2.1 Graph theoretic definitions	7
2.2 Sets and multi-sets	9
2.3 Computational complexity theory	11
2.3.1 P versus NP	13
2.3.2 NP-complete and NP-hard	14
2.4 Matchings	15
2.4.1 Matchings and augmenting paths	15
2.4.2 Matchings in bipartite graphs	16
2.5 Flow graphs	17
2.5.1 Weighted flow graphs	20
2.5.2 Ford-Fulkerson Algorithm	20
2.6 Treewidth and pathwidth	21
2.6.1 Treewidth	21
2.6.2 Some properties of tree decompositions	22
2.6.3 Treewidth and graph minors	24
2.6.4 Pathwidth	25
2.7 Outline	26
I Colour degree matrices of graphs with few cycles	27
3 Discrete tomography and realising colour demand matrices	29
3.1 Discrete tomography	30
3.2 Colour degree matrices	30
3.2.1 Demand sequences	30
3.2.2 Colour degree matrices	31
3.2.3 Connection with discrete tomography	31
3.3 Outline	32

4	Colour degree matrices of forests	35
4.1	Degree sequences of forests	35
4.2	Colour degree matrices of forests	36
4.3	Conclusion	38
5	Colour degree matrices of graphs with at most one cycle	41
5.1	Degree sequences of connected unicyclic graphs	41
5.2	The unicycle conditions	43
5.2.1	Properties of tree and cycle critical sets	44
5.2.2	The unicycle conditions are necessary	44
5.2.3	Equivalent conditions	45
5.2.4	Implications of the unicycle conditions	46
5.3	Directly forced edges	48
5.3.1	Directly forced edges are forced and can be realised	49
5.3.2	Conditions that imply a directly forced edge	50
5.4	Directly forced triangles	51
5.4.1	Properties of directly forced triangles	52
5.4.2	A directly forced triangle is forced	54
5.4.3	A directly forced triangle can be realised	54
5.4.4	Conditions that imply a directly forced triangle	55
5.5	The unicycle conditions are sufficient	56
5.5.1	Sufficiency when each total degree is two	56
5.5.2	Sufficiency in general	57
5.6	Colour degree matrices of multi-graphs with at most one cycle	63
5.6.1	Degree sequences of connected unicyclic multi-graphs	63
5.6.2	Colour degree matrices of multi-graphs with at most one cycle	64
5.7	Colour degree matrices of pseudoforests	66
5.8	Conclusion	68
5.8.1	Directed degree sequences of unicyclic graphs	68
6	Algorithms for realising colour demand matrices	71
6.1	Checking condition (U2)	72
6.1.1	Algorithm sketch	77
6.2	Finding minimal sets in intersecting families	77
6.2.1	Intersecting families	77
6.2.2	Finding minimal sets in intersecting families	78
6.3	Checking conditions (U3) and (U4)	80
6.3.1	Finding cycle critical sets	80
6.3.2	Finding minimal cycle critical sets	82
6.3.3	Checking conditions (U3) and (U4)	82
6.4	Checking other conditions	83
6.4.1	Finding a directly forced triangle	83
6.4.2	Checking the conditions from Theorem 4.2.3	83
6.4.3	Finding tree critical sets	83
6.4.4	Finding a directly forced edge	84
6.4.5	Checking the conditions from Theorem 5.7.3	84

6.5	Finding realisations	84
6.5.1	Finding a forest realisation	84
6.5.2	Finding a unicyclic realisation	85
6.5.3	Finding a unicyclic multi-graph realisation	87
6.5.4	Finding a pseudoforest realisation	88
6.6	Conclusion	88
II	Modelling the reverse radar problem	89
7	The coloured L-model	91
7.1	The coloured L -model	91
7.1.1	The coloured L -model	91
7.2	The complexity of finding L -decompositions	94
7.2.1	NP-complete cases	94
7.2.2	L is such that $ \text{col}(H) \leq 2$	96
7.3	The coloured $\{K_1, K_2\}$ -model	100
7.3.1	The coloured $\{K_1, K_2\}$ -model	100
7.4	The coloured $\{K_1, K_2\}$ -model and matchings	101
7.5	Outline	105
8	Decompositions in 2-coloured trees and cycles	109
8.1	Definitions	109
8.2	Decomposition in trees	110
8.2.1	Algorithm sketch	110
8.2.2	Details of implementation	110
8.2.3	Correctness	111
8.3	Decomposition in cycles	113
8.3.1	Breaking cycles	113
8.3.2	Algorithm Sketch	114
8.3.3	Details of implementation	114
8.3.4	Correctness	115
8.4	Conclusion	115
9	The red-blue $\{K_1, K_2\}$- model	117
9.1	Closure conditions	118
9.1.1	Weighted Hall's condition	118
9.1.2	Closure conditions	119
9.2	Defect closure conditions	120
9.3	The partially separated cycle has pathwidth 4	120
9.4	Checking weighted Hall's condition in graphs with bounded pathwidth	121
9.4.1	The strategy	122
9.4.2	Determining intermediate defects	122
9.4.3	Towards a recursive definition of $\Delta_t(A, B)$	123
9.4.4	A recursive definition of $\Delta_t(A, B)$	125
9.4.5	Algorithm	126

Contents

9.5	Checking weighted Hall's condition in graphs with bounded treewidth	126
9.5.1	The strategy	127
9.5.2	Redefining $\Delta_t(A, B)$	127
9.5.3	Towards a recursive definition of $\Delta_t(A, B)$	128
9.5.4	A recursive definition of $\Delta_t(A, B)$	130
9.5.5	Algorithmic Representation	131
9.6	Weighted matchings in bipartite graphs with bounded treewidth	132
9.6.1	d is small	132
9.6.2	d is large	132
9.7	Conclusion	134
10	The general coloured $\{K_1, K_2\}$ - model	135
10.1	Decompositions and minimum cost matchings	135
10.2	C_s has bounded pathwidth	136
10.3	Matching in graphs with bounded treewidth	136
10.3.1	Definitions	137
10.3.2	A recursive definition of $I_t(S)$	137
10.3.3	Algorithm sketch	139
10.4	Defect matching	139
10.4.1	A recursive definition of $J_t(A, B)$	140
10.4.2	Algorithm sketch	142
10.5	Conclusion	143
11	The likelihood and two-time models	145
11.1	A note on multi-set notation	145
11.2	The likelihood model	146
11.2.1	Calculating the probability of a solution	150
11.2.2	The coloured L -model versus the likelihood model	151
11.3	The formalised likelihood problem	151
11.3.1	LHP and FLH	152
11.3.2	Calculating the probability of a solution	153
11.4	The complexity of FLH	154
11.4.1	FLH is NP-hard	154
11.4.2	Tractable cases	155
11.5	The two-time model	158
11.5.1	Calculating the probability of a solution	160
11.6	The formalised two-time problem	161
11.6.1	TTP and FTT	161
11.6.2	Calculating the probability of a solution	162
11.7	The complexity of FTT	163
11.7.1	One colour per ship is tractable	163
11.7.2	FTT is NP-hard	165
11.7.3	FTT is NP-hard, even if all ships have two colours	165
11.8	Conclusion	168

12 Concluding remarks	169
12.1 Colour degree matrices of graphs with few cycles	169
12.2 Modelling the reverse radar problem	170
12.3 Future directions for Thales UK	170
Bibliography	173
Appendices	183
A Nederlandse samenvatting	185
A.1 Het twee-sensoren probleem	186
A.2 Het omgekeerde radar probleem	189
A.3 Conclusie	193

List of Figures

1.1	The current situation and two sensors observing it, then an abstraction of the current situation. Finally the output of the sensors. . . .	1
1.2	The two-sensor problem.	2
1.3	A (bearing-only) sensor used in the reverse radar problem, observing radar models from R . Library L shows the different types of ships. .	3
1.4	The different stages of the reverse radar problem.	4
2.1	Some examples of well known graphs.	9
2.2	Maximal matchings. The third matching is also maximum [80]. . .	16
2.3	An optimal tree decomposition of C_7	22
2.4	The forbidden minors of graphs with treewidth at most 3 [37]. . . .	25
2.5	A graph with a tree and path decomposition.	26
3.1	(a) An instance of two dimensional discrete tomography with a solution. (b) Same instance as a bipartite demand sequence, with a bipartite graph realisation.	29
3.2	(a) An instance of two dimensional discrete tomography with solution. (b) Same instance as a bipartite demand matrix, with a bipartite graph realisation.	30
6.1	Example G_D graph.	72
6.2	A visual guide to the proof of Lemma 6.1.2.	73
6.3	A visual guide to the proof of Lemma 6.1.2.	74
6.4	A visual guide to the proof of Lemma 6.1.3.	75
6.5	A visual guide to the proof of Lemma 6.3.1.	81
7.1	An L -decomposition of an R -coloured graph G	93
7.2	The vertex set of F	96
7.3	Augmenting paths in F	98
7.4	R -coloured graph G , $R = \{\text{orange}, \text{blue}, \text{pink}\}$, and the fully separated graph G_s . .	101
7.5	A 2-coloured graph G and the partially separated graph G'_s	104
8.1	Counterexample: the edge between v_1 and v_6 cannot be removed. .	114
9.1	2-coloured graph G and the partially separated graph G'_s	118
9.2	A partially separated cycle and its path-decomposition.	121
9.3	Example pairs $(A', B') \subseteq D_{t-1}$ and $(A, B) \subseteq D_t$	122
9.4	Parent node v with children u and w	127
9.5	Example pair of pairs $((A_u, B_u), (A_v, B_v)) \in \mathcal{C}_t(A, B)$	129
9.6	Vertex weighted graph G and $G(u, v, \alpha, \beta)$	133
10.1	Node t , with children u and v and the subsets S , A_u and A_v	137

List of Figures

10.2	Node t , with children u and v and the subsets A and B , A_u and B_u , and A_v and B_v	140
11.1	G_Λ from L -colouring Λ and R -colouring P in Example 5. The matching in red corresponds to a possible dispersion function.	152
11.2	G_C from R -colouring P in Example 5. The matching in red corresponds to Λ	157
11.3	The two-time problem.	159
11.4	The graph G_{12} corresponding to $\Lambda = \Lambda_1$ in Example 5 and Λ_2 in Example 6. A possible transition is shaded in red.	162
11.5	Example G_f with P_1 and P_2 from Example 6 and library $L = R = \{a, b, c\}$	164
A.1	Links sensor 1 en 2 die een gebied observeren, in het midden het gekleurde raster in het gezichtsveld van sensor 1 en 2 en rechts wat sensor 1 en 2 detecteren.	186
A.2	Het twee-sensoren probleem.	187
A.3	Verschillende grafen.	187
A.4	(a) Een discrete tomografie probleem met oplossing. (b) Hetzelfde probleem als een bipartiete graadrij met oplossing.	188
A.5	The sensor in het omgekeerde radar probleem ‘luistert’ naar radar modellen. Bibliotheek L geeft de verschillende schepen weer.	189
A.6	De verschillende stadia van het omgekeerde radar probleem.	190
A.7	Een L -decompositie van een graaf.	191
A.8	Koppelingen in verschillende grafen [80].	191
A.9	Een graaf links met boomdecompositie rechts.	192
A.10	Een maximale stroming in een gerichte graaf	193

List of Tables

3.1	An overview of time complexities for the demand matrix decision problem and for finding the required realisation.	33
6.1	An overview of the different algorithms in this chapter.	88
7.1	Overview of the complexity of finding a $\{K_1, K_2\}$ -decomposition in different R -coloured graphs.	107
9.1	Complexities for finding a $\{K_1, K_2\}$ -decomposition in different 2-coloured graphs G , where $d = \max_{v \in V(G)} \text{col}(v) $	134
10.1	Complexities for finding a $\{K_1, K_2\}$ -decomposition in an R -coloured graph G , when G_s has treewidth at most $p - 1$	143
11.1	Overview of the complexity of different FLH and FTT instances. . .	168

Introduction

In this thesis we investigate problems arising from emitter-platform association, as part of research undertaken for Thales UK under studentship BK/11/012. We will focus on the two-sensor problem and the reverse radar problem. Both will be introduced in this chapter. The two-sensor problem is discussed in Part I of this thesis and the reverse radar problem is discussed in Part II.

1.1 The two-sensor problem

The two-sensor problem was raised during a project meeting with Thales in March 2013 [65]. It was discussed in the context of sensors observing radars or ships, in a way that observations have a one-to-one correspondence with platforms, e.g.

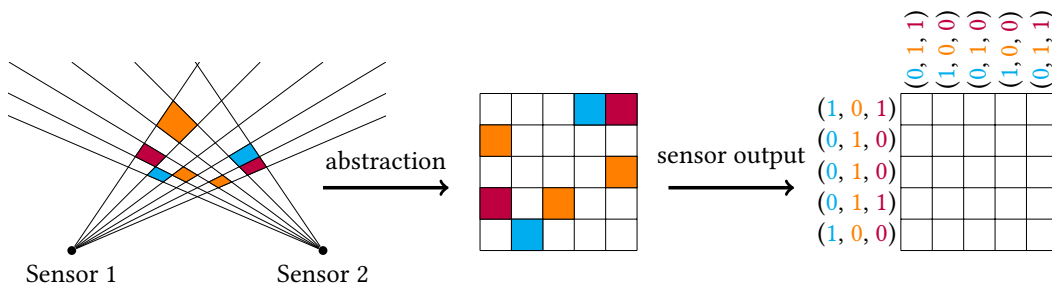


Figure 1.1: The current situation and two sensors observing it, then an abstraction of the current situation. Finally the output of the sensors.

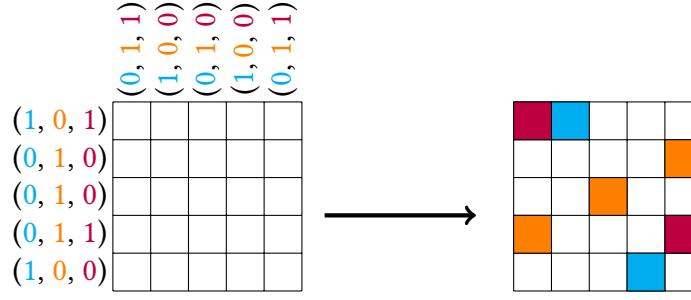


Figure 1.2: The two-sensor problem.

ships. This situation could either arise after having a successful emitter-to-platform association algorithm on the output of both sensors (see Section 1.2), or if we assume that each platform is determined by one unique type of observation. For example if each type of ship carries a unique emitter model. However, the underlying problem has much more general applications, as we will see in Chapter 3.

Suppose there are two sensors and suppose each sensor ‘observes’ exactly n bearings, that is, a sensor observes directions without a perception of depth. On each bearing there are between zero and n observations. We will only consider bearings observed by sensor 1 which intersect bearings observed by sensor 2. We will refer to each type of observation as one of c different colours, where c is a positive integer. This problem is defined by two n by c matrices, say A and B for sensor 1 and 2 respectively, containing the multiplicity of each colour along every bearing. Fig. 1.1 shows the current situation and two sensors ‘observing’ it. It then shows an abstraction of the situation and finally the output the sensors produce. This sensor output forms the input to our problem, i.e. the multiplicity of each colour in each row and column. As depicted in Fig. 1.2, we try to find a situation which corresponds to the sensor output. Essentially we have the horizontal and vertical projections of a coloured n by n grid, and we want to reconstruct the original layout of the coloured squares on the grid. This is more commonly known as discrete tomography, or colour degree matrices. Skip ahead to Chapter 3 to learn more about that.

1.2 Reverse Radar

The emitter-to-platform association problem was the initial starting point for this thesis. We will call it the reverse radar problem for the remainder of this thesis, as

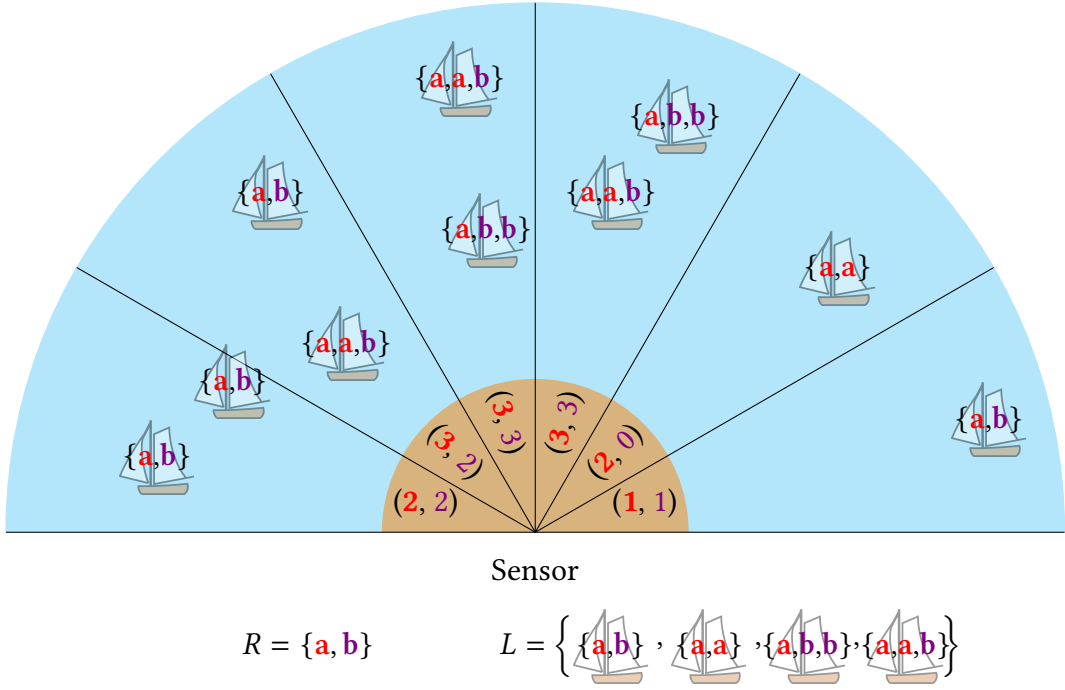


Figure 1.3: A (bearing-only) sensor used in the reverse radar problem, observing radar models from R . Library L shows the different types of ships.

in all scenarios considered emitters are radars. This problem is not restricted to just radars however; Armour [5] discusses an example of emitter-to-platform association, which also includes cell phone signals. A radar is an electromagnetic sensor used for detecting, locating, tracking, and recognising objects of various kinds. It operates by transmitting electromagnetic pulses toward objects and observing the reflected signals returned from them. These objects, also called targets, can range from birds to aircrafts, to astronomical bodies, or even terrain, or rain [84]. Radars are most commonly used to determine the presence, location, and velocity of such objects, but sometimes radars can obtain their size and shape as well. A radar is an active sensing device in that it has its own source of illumination (a transmitter) for locating targets [84, 107].

In the reverse radar problem, signals emitted by nearby radars are (passively) detected, as done by Electronic Surveillance Measures (ESM) intercept receivers [64, 95]. These signals must be processed to provide an informed view of the shipping traffic and other (radar-carrying) platforms in the area [2]. Fig. 1.3 shows an illustration of a (bearing-only) sensor used in the reverse radar problem. Each segment is a range of bearings with some radar-carrying platforms – ships in this case – from library L . Each ship carries a multi-set of radar models from R . The

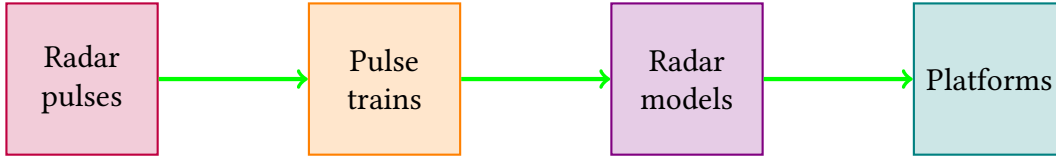


Figure 1.4: The different stages of the reverse radar problem.

sensor observes the number of instances of each radar model in each segment. From these observations we aim to reconstruct which types of ships are in each segment. In general, a platform can be anything with a radar, such as an aircraft, or a helicopter, but throughout this thesis we will assume that the platforms are all ships. This is a simplification of the problem, which makes it two dimensional. It is a passive way of gathering information, in the sense that the target to be located needs to send out pulses, before it can be noticed, but the received signal carries a lot of information such as frequency, polarisation, scan type and pulse width. We may use this information to ultimately determine the identity of the platform carrying the radar [1, 2].

At first sight this problem may not seem combinatorial in nature. This is because the problem consists of several stages, as depicted in Fig. 1.4.

1. **Pulse to pulse train:** When collecting pulses from more than one radar it is not always clear which pulses come from the same radar. Pulses likely to be from a single radar are grouped together in pulse trains.
2. **Pulse train to radar model:** When the pulse trains are constructed, the possible model(s) of radar(s) sending out this pulse train are found from different properties of the signal, such as pulse width, and frequency.
3. **Radar types to platforms:** Finally, when it is determined which radar models are where, they are associated with platforms carrying these radar models. These platforms are selected from a library, in which platforms are listed with the radars they are known to carry.

The first and second parts of the problem are addressed by Thales UK, as outlined in OCIAM workshop reports [1, 2], in a problem description by the initiator of the project [32], in internal papers [41, 115], as communicated during meetings [64, 65, 113] and as is discussed in general literature; Gavish and Weiss [49] for example give a performance analysis of two of the main methods for a bearing-only passive emitter location system and Torrieri [111] discusses the principal methods used in passive location systems. Moreover, Rose [95] and Guerri et al. [50] describe

a commercial approach to the passive emitter location problem.

Matching radar models to platforms however, is most essential for intelligence. For example, it is more useful to know that the *HrMs Rotterdam* and *HrMs Johan de Witt* are stationed nearby, than it is to know that there is a cluster of radars at that place, because given the ship types, you can find out that these are Dutch landing platform docks that can station up to 613 troops and 36 torpedoes (inspired by [3, 98]). Algorithms solving the reverse radar problem will increase situational awareness. The library of ships with the radars they are known to carry, could for example come from ‘Jane’s Fighting Ships’ [98]. This is a collection describing the world’s war ships, first published in 1898 by Fred T. Jane [71] and since then *Jane’s Information Group* publishes a new version every few years. The latest is Saunders [98].

This part of the problem is an association problem. It is closely related to matchings in vertex-coloured graphs, as will be shown in this thesis. The goal of this thesis is to understand this problem and related problems better and to find effective approaches to them. This process is two-fold: first adequate models need to be found, after which efficient algorithms can be developed. We present the coloured $\{K_1, K_2\}$ -model, and the likelihood and two-time model in Chapters 7 and 11.

The English Channel is a location in which this emitter-to-platform association might be used [1, 2, 32, 41, 64, 65, 113]. The platforms that will form the input to the problem will likely be military vessels, but we need to separate from many other commercial vessels. More details on possible scenarios can be found in OCIAM workshop reports [1, 2]. More details about this particular scenario are in Armour [5], Davies [32], Hillebrand [64], van Schelle [113], Ventress [115]. Mock scenarios can be found in Stove [108, 109].

1.3 Outline

Chapter 2 contains some preliminary definitions and results from graph theory and theoretical computer science. Readers that are already familiar with these may go straight to Part I (Chapters 3-6), which describes colour degree matrices of graphs with few cycles, or to Part II (Chapters 7-11), which introduces different models for the reverse radar problem.

Part I starts in Chapter 3, which links the two-sensor problem to discrete tomography and colour degree matrices. Chapter 4 reviews a result about colour degree

Chapter 1. Introduction

matrices of forests. The main result is in Chapter 5, where we give necessary and sufficient conditions for a colour demand matrix to be a colour degree matrix of a graph with at most one cycle. This result is generalised to pseudoforests and unicyclic multi-graphs at the end of Chapter 5. Polynomial time algorithms to check these conditions and to find a realisation are given in Chapter 6.

Part II starts in Chapter 7, where the coloured L -model is introduced, but we will focus on a special case of this: the coloured $\{K_1, K_2\}$ -model. In Chapter 8 results for 2-coloured cycles and trees are presented. In Chapter 9 general results for the 2-coloured $\{K_1, K_2\}$ -model are presented and Chapter 10 focuses on the $\{K_1, K_2\}$ -model with arbitrarily many colours. Chapter 11 describes the likelihood and two-time models, which are both extensions to the coloured L -model, extending the notion of uncertainty and introducing time.

In the final chapter we discuss some future directions for this project. Appendix A contains a (Dutch) summary of this thesis, which contains no new information.

2

Preliminaries

In this chapter some definitions and results are given, that will be used in later chapters.

2.1 Graph theoretic definitions

In this section a few graph theory definitions are given. For further references see Bollobás [17, 18], Diestel [34], Harary [57], Lovász and Plummer [77], West [116].

Definition 2.1.1. [34, 83] A **(simple) graph** $G = (V, E)$ consists of a non-empty set of elements $V(G) = V$ and a subset $E(G) = E$ of the set of unordered pairs of distinct elements of $V(G)$. The elements of $V(G)$ are called the **vertices** of G . If $e = \{x, y\} \in E(G)$, then x and y are the **endpoints** of e and e is **incident** with x and y . Moreover, if $\{x, y\} \in E(G)$, x and y are said to be **adjacent** and if $\{x, y\} \notin E(G)$, then the vertices x and y are said to be **non-adjacent**. We will often write edge $e = \{x, y\}$ as $e = xy$. G is a **finite graph** if $V(G)$ is finite. A finite graph G may be **drawn** by placing each vertex at a point and representing each edge by a curve joining the locations of its endpoints. A **path** is a sequence of alternating vertices and edges, beginning and ending with vertices, where each edge's endpoints are the preceding and following vertices in the sequence, such that no vertex (except the first and the last vertex) is ever repeated. A graph H is a **subgraph** of G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. For $A \subseteq V$, the **induced subgraph** $G[A]$ is the set of vertices A , together with any edges in E whose endpoints are both in A . The **neighbourhood** $\Gamma(v)$

or $N(v)$ of a vertex v is the set of vertices adjacent to v . More generally for $A \subseteq V(G)$, $\Gamma(A) = (\bigcup_{v \in A} \Gamma(v)) \setminus A$ is the neighbourhood of the set A in $V(G) \setminus A$. The **degree** $d_G(v) = d(v)$ of a vertex v is the number of edges incident to v in graph G . A **leaf** is a vertex with degree exactly one. An **independent set** or a **stable set** is a set of vertices in a graph, no two of which are adjacent. A graph G is **connected** if for all $u, v \in V(G)$ there is a path from u to v . Otherwise G is **disconnected**. The **connected components** of G are its maximal connected subgraphs of G . An **isolated vertex** is a vertex of degree zero. The **(graph) distance** between vertex u and v in the same component of G is the length of the shortest path between them.

Definition 2.1.2. [34] Let $e = xy$ be an edge in a graph $G = (V, E)$. By G/e we denote the graph obtained from G by **contracting** the edge e into a new vertex v_e , which becomes adjacent to all the former neighbours of x and y . Formally G/e is a graph (V', E') with vertex set $V' = v_e \cup V \setminus \{x, y\}$ and edge set

$$E' = \{vw \in E : \{v, w\} \cap \{x, y\} = \emptyset\} \cup \{v_e w : xw \text{ or } yw \in E \setminus e, w \neq x, y\}.$$

Definition 2.1.3. [116] A graph H is a **minor** of a graph G , denoted by $H \preceq G$, if a copy of H can be obtained from a subgraph of G by repeatedly deleting and/or contracting edges of G .

Definition 2.1.4. Two graphs G and H are said to be **isomorphic** (written $G \simeq H$) if there exists a bijection ϕ between $V(G)$ and $V(H)$ such that $\{x, y\} \in E(G) \Leftrightarrow (\phi(x), \phi(y)) \in E(H)$ [83]. This is also referred to as a **graph isomorphism**.

Definition 2.1.5. The following graphs are often encountered:

- A graph on n vertices is **complete** if all vertices are pairwise adjacent. It is denoted by K_n . A complete subgraph on k vertices is also referred to as a **k -clique**.
- A **path** is a non-empty graph $P_k = (V, E)$ of the form $V = \{x_0, x_1, \dots, x_k\}$, $E = \{x_0x_1, x_1x_2, \dots, x_{k-1}x_k\}$. The **length** of a path P_k is k , the number of vertices on the path.
- A **cycle** C_n is the graph $P_n + x_nx_0$, that is a path with an edge between its endpoints. The **length** of a cycle C_n is n , the number of vertices on the cycle.
- A **tree** T is a connected acyclic graph.
- A **forest** F is a graph where each connected component is a tree.
- A **rooted tree** is a tree with a special vertex, which is called the root r . The edges have a natural orientation, towards or away from the root.

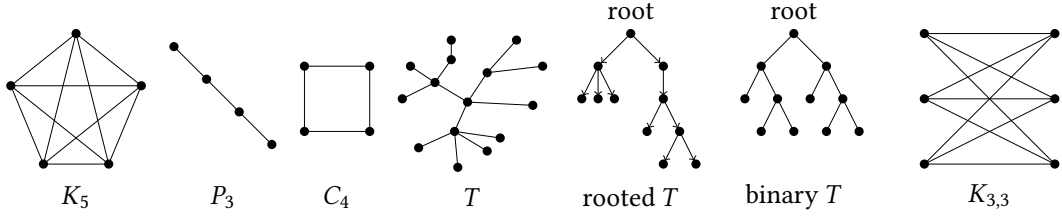


Figure 2.1: Some examples of well known graphs.

- A **binary tree** is a rooted tree such that each vertex except the root and the leaves have degree three. The root has degree two. A **perfect binary tree** is a tree for which all the leaves are at the same graph distance from the root.
- A graph G is **k -partite** if $V(G)$ can be expressed as the union of k (possibly empty) disjoint independent sets (so each edge has its endpoints in two distinct independent sets). Each independent set is called a **vertex class**. When $k = 2$, such a graph is called **bipartite**. If $G = (V, E)$ is a bipartite graph with vertex classes X and Y , we will write this as $V = X + Y$ or $V = X \cup Y$. The **complete bipartite graph** $K_{n,m}$ is a bipartite graph with two vertex classes of sizes n and m respectively and all edges between the two vertex classes.

Examples can be seen in Fig. 2.1.

Lemma 2.1.6 (Handshaking Lemma [40]). For any graph $G = (V, E)$,

$$\sum_{v \in V} d(v) = 2|E|.$$

For further reading see Bollobás [18], Bondy and Murty [19], Diestel [34], Harary [57], West [116].

2.2 Sets and multi-sets

Definition 2.2.1. We will denote the **natural numbers with zero appended** by $\mathbb{N}_0$, that is $\mathbb{N}_0 = \{0, 1, 2, \dots\}$.

Definition 2.2.2. Given a function f on a domain X ,

$$\arg \max_{x \in X} f(x) = \{x : \forall y \in X : f(y) \leq f(x)\}$$

is the set of elements $x \in X$ for which $f(x)$ attains its largest value. This set may be empty, have one element, or have multiple elements.

Definition 2.2.3. Let A be a boolean statement (i.e. true or false), then

$$1_A = \begin{cases} 1 & \text{if } A \text{ is true,} \\ 0 & \text{otherwise.} \end{cases}$$

Theorem 2.2.4 (Strong form of pigeon hole principle [23]). Let $q_1, q_2, \dots, q_n$ be positive integers. If

$$q_1 + q_2 + \dots + q_n - n + 1$$

objects are put into n boxes, then there is some $i \in [n]$ such that the i^{th} box contains at least q_i objects.

Definition 2.2.5. Given a set S , the **power set** $\mathcal{P}(S)$ of S is the set of all subsets of S , including the empty set and S itself.

Definition 2.2.6. Let $\{A_i : i \in I\}$ be a family of sets. The **disjoint union** of this family is the set

$$\bigcup_{i \in I} A_i = \bigcup_{i \in I} \{(x, i) : x \in A_i\}.$$

The elements of the disjoint union are ordered pairs (x, i) , where i refers to the set A_i the element x originated from.

Definition 2.2.7. A family of sets P is a **partition** of X if and only if all of the following conditions hold:

1. $\emptyset \notin P$,
2. $\bigcup_{A \in P} A = X$, and
3. if $A, B \in P$ and $A \neq B$ then $A \cap B = \emptyset$,

where $\emptyset$ is the empty set.

Definition 2.2.8. Suppose that S is a set and $\bullet$ is some binary operation $S \times S \rightarrow S$, then S with $\bullet$ is a **monoid** if it satisfies the following:

1. **Associativity:** For all $a, b, c \in S$,

$$(a \bullet b) \bullet c = a \bullet (b \bullet c).$$

2. **Identity element:** There is an element $e \in S$ such that for every element $a \in S$,

$$e \bullet a = a \bullet e = a.$$

In other words, a monoid is a **semigroup** with an identity element. It can also be thought of as a **magma** with associativity and identity. Moreover, $\bullet$ is **commutative**

if for all $a, b \in S$

$$a \bullet b = b \bullet a.$$

A monoid with a commutative operation is called a **commutative monoid**.

Definition 2.2.9. A **multi-set** M is an extension of the concept of a set. While a set can contain only one occurrence of any given element, a multi-set may contain multiple occurrences of the same element. We will also refer to this as a **family** or a **collection**. The number of times an element $a \in M$ appears is called the **multiplicity** $m(a)$ of a in M . Union, intersection, set removal, containment and subsets are as for normal sets, but now multiplicity of elements is taken into account as well. Union of multi-sets M_i is denoted by $\uplus_i M_i$. For example

$$\{1, 1, 2, 2\} \uplus \{2, 2, 3\} = \{1, 1, 2, 2, 2, 2, 3\}$$

$$\{1, 1, 2, 2\} \cap \{2, 2, 3\} = \{2, 2\}$$

$$\{1, 1, 2, 2\} \setminus \{2, 2, 3\} = \{1, 1\}$$

$$1 \in \{1, 1, 2, 2\}$$

$$3 \notin \{1, 1, 2, 2\}$$

$$\{1, 1, 2\} \subseteq \{1, 1, 2, 2\}$$

$$\{1, 1, 1\} \not\subseteq \{1, 1, 2, 2\}.$$

Alternative notation for an element in a multi-set is as an ordered pair (a, i) , where a is an element and $1 \leq i \leq m(a)$ an index of this element. For example

$$\{a, a, b, b, b, c\} = \{(a, 1), (a, 2), (b, 1), (b, 2), (b, 3), (c, 1)\}.$$

This notation is useful if it is necessary to distinguish between elements in a multi-set.

2.3 Computational complexity theory

Computational complexity theory is the study of the fundamental capabilities and limitations of computers. It investigates what makes problems ‘hard’ or ‘easy’, by classifying them according to their computational difficulty [105].

A *problem* is a general question to be answered, usually possessing several *parameters*, or free variables, whose values are left unspecified. It is described by giving a general description of all its parameters and a statement of what properties the answer, or *solution*, is required to satisfy [48]. Given a problem, a *problem instance*

is a string over a given alphabet, usually the binary alphabet $\{0, 1\}$, specifying the particular parameters. An *algorithm* is any well-defined computational procedure that takes some value, or set of values as *input* and produces some value, or set of values, as *output* [28]. This may be formalised with the use of Turing machines, a hypothetical computational device that manipulates symbols on a strip of tape according to a table of rules. Despite its simplicity, a Turing machine is capable of expressing any computer algorithm, of simulating any programming language [88], but we will not dwell on those in this short introduction to complexity theory. An algorithm takes a problem instance as input and will produce a solution as output. For example the sorting problem has as input an unsorted list of integers and as output the list of integers sorted in increasing order. Any known sorting algorithm, such as insertion sort or quicksort, could solve this problem. A problem instance could be $(9, 2, 1, 4, 3, 6, 8)$, with output $(1, 2, 3, 4, 6, 8, 9)$.

Definition 2.3.1. A **decision problem** is a problem with binary output. More formally, a decision problem is a boolean function

$$\mathcal{L} : \mathcal{P}(\{0, 1\}) \rightarrow \{0, 1\}.$$

For all $x \in \mathcal{P}(\{0, 1\})$, if $\mathcal{L}(x) = 1$, x is an **accepting instance** of $\mathcal{L}$; otherwise it is a **rejecting instance**.

An example is: given a graph G , is G k -colourable? Other well-known decision problems are satisfiability and k -satisfiability, defined below.

Definition 2.3.2 ([28]). A **boolean formula** contains literals whose values are 0 or 1; boolean connectives such as $\wedge$ (AND), $\vee$ (OR) and $\neg$ (NOT); and parentheses. A boolean formula is **satisfiable** if there exists some assignment of 0 and 1 of the literals that causes it to evaluate to 1. A boolean formula is in **conjunctive normal form (CNF)** if it is expressed as an AND of clauses, each of which is the OR of one or more possibly negated literals. A boolean formula is in **k-conjunctive normal form**, or **k-CNF** if each clause contains exactly k literals.

For example $(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_3)$ is in 2-CNF and $x_1 = x_2 = 1$ and $x_3 = 0$ is a satisfying assignment.

Problem 1 (Satisfiability (SAT)). Given a boolean formula C on literals U , is C satisfiable?

2.3. Computational complexity theory

For example if $1 = x_1 = x_2 = x_3$, the boolean formula $x_1 \wedge ((\neg x_1 \wedge x_2) \vee x_3)$ is equal to 1.

Problem 2 (*k-Satisfiability (k-SAT)*). *Given a boolean formula C in k -CNF on literals U , is C satisfiable?*

The *input size* of an instance I is the number of bits in the input, usually denoted by n . The *time complexity* or *running time* of an algorithm is the largest amount of time, i.e. number of ‘steps’, the algorithm could take on a problem instance of size n . We will think of a step as completing one instruction [100]. Similarly, the *space complexity* of an algorithm is the largest amount of memory used at any point during an execution of the algorithm on a problem instance of size n . Both time and space complexity are usually defined in terms of time and memory usage of a Turing machine, but it is equivalent to think of this time and space usage of a computer because they will be the same up to a constant factor [88]. The time and space complexity are usually denoted in *asymptotic notation* or *big-O notation*. For a given function $g(n) : \mathbb{N} \rightarrow \mathbb{N}$, $\Theta(g(n))$, $O(g(n))$ and $\Omega(g(n))$ are sets of functions $f(n) : \mathbb{N} \rightarrow \mathbb{N}$ defined as follows:

$$\Theta(g(n)) = \left\{ f(n) : \text{there are positive constants } c_1, c_2, n_0 \text{ such that} \right. \\ \left. 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n), \text{ for all } n \geq n_0 \right\}, \quad (2.1)$$

$$O(g(n)) = \left\{ f(n) : \text{there are positive constants } c, n_0 \text{ such that} \right. \quad (2.2)$$

$$\left. 0 \leq f(n) \leq c g(n), n \geq n_0 \right\}, \quad (2.3)$$

$$\Omega(g(n)) = \left\{ f(n) : \text{there are positive constants } c, n_0 \text{ such that} \right. \quad (2.4)$$

$$\left. 0 \leq c g(n) \leq f(n), \text{ for all } n \geq n_0 \right\}. \quad (2.5)$$

When $f(n) \in O(g(n))$, $\Theta(g(n))$, or $\Omega(g(n))$ we will write this as

$$f(n) = O(g(n)), \Theta(g(n)), \text{ or } \Omega(g(n)),$$

respectively. For example an algorithm that runs in $68n^2 + 3n + 5$ steps is said to have time complexity $O(n^2)$.

2.3.1 P versus NP

Definition 2.3.3 ([105]). *We say an algorithm runs in **polynomial time** if there is a constant k such that the time complexity of the algorithm is $O(n^k)$. **P** is the*

complexity class consisting of all problems, that are decidable in polynomial time.

There are many problems known to be in P, such as: given a graph G , does G contain a perfect matching?

Definition 2.3.4 ([105]). A **verifier** V is a two-argument algorithm, which takes an input $x \in \mathcal{P}(\{0, 1\})$ and a **certificate** or **witness** $y \in \mathcal{P}(\{0, 1\})$ and outputs an element from $\{0, 1\}$. A decision problem $\mathcal{L}$ is in the complexity class **NP** if and only if there exists a polynomial time verifier V and a constant c such that for any n -bit input string $x \in \mathcal{P}(\{0, 1\})$ there is a certificate y of length $O(n^c)$, such that $V(x, y) = 1$ if and only if $\mathcal{L}(x) = 1$. Such a problem is **polynomially verifiable**.

An example of a problem in NP is k -clique: Given a graph G , does G contain a complete graph on k vertices? Given a graph G , a possible certificate for containing a k -clique is a k -clique $K \subseteq G$. It can be checked in $O(k^2)$ time whether K is a complete graph and $k \leq n$, thus k -clique is in NP.

So roughly speaking, P is the class of problems that can be decided quickly and NP is the class of problems that can be verified quickly [27]. It is easy to see that P must be a subset of NP, but whether $P=NP$ is still an unsolved problem. In fact, it is one of seven Millennium Problems stated by the Clay Mathematics Institute in 2000, awarding \$1 million for a solution to this problem [25].

2.3.2 NP-complete and NP-hard

Definition 2.3.5 ([28]). A decision problem $\mathcal{L}_1$ is **polynomially reducible** to decision problem $\mathcal{L}_2$, written $\mathcal{L}_1 \leq_p \mathcal{L}_2$, if there exists a polynomial time computable algorithm $f : \mathcal{P}(\{0, 1\}) \rightarrow \mathcal{P}(\{0, 1\})$ such that for all $x \in \mathcal{P}(\{0, 1\})$, $\mathcal{L}_1(x) = 1$ if and only if $\mathcal{L}_2(f(x)) = 1$. Function f is called a **reduction function** and algorithm F , which computes f is called a **reduction algorithm**.

Lemma 2.3.6 ([28]). If $\mathcal{L}_1, \mathcal{L}_2$ are decision problems such that $\mathcal{L}_1 \leq_p \mathcal{L}_2$, then $\mathcal{L}_2 \in P$ implies $\mathcal{L}_1 \in P$

NP-completeness was first introduced by Cook [26]. Cook proved that 3-SAT has the property that every other problem in NP is polynomially reducible to it [26]. This implies that if SAT can be solved in polynomial time, then any problem in NP can be solved in polynomial time. Finally he suggested that there might be more problems in NP with this property. Karp [72] showed that there are many well-known problems in NP with this property and named them *NP-complete problems*.

Definition 2.3.7. A decision problem L is **NP-complete** if

1. $L \in \text{NP}$, and
2. $L' \leq_p L$ for every $L' \in \text{NP}$.

A problem is said to be **NP-hard** if any problem in NP is polynomially reducible to it; in other words, if the second property of NP-completeness holds, but not necessarily the first.

Note that NP-hard does not imply NP-complete. In Garey and Johnson [48] there are many examples of NP-hard and NP-complete problems. This is very useful because of the following lemma:

Lemma 2.3.8 ([28]). If L is a decision problem such that $L' \leq_p L$ for some NP-complete problem L' , then L is NP-hard. Moreover, if $L \in \text{NP}$, then L is NP-complete.

For further reading see Garey and Johnson [48], Papadimitriou [88], Sipser [105].

2.4 Matchings

Matchings are a useful and well-studied object in graph theory and we will encounter them in several places in this thesis.

Definition 2.4.1 ([34]). A set of **independent** edges M in a graph $G = (V, E)$, is such that no two edges in M share a vertex. M is more commonly called a **matching**. M is a matching of $U \subseteq V$ if $V(M) = U$, i.e every vertex in U is incident with an edge in M . M is **maximal** if no edges of G can be added to M without violating the matching property. A **maximum matching** is a matching that contains the maximum possible number of edges in G . For $X \subseteq V$, an **X -saturating matching** is a matching that covers at least every vertex in X and we say the vertices in X are **saturated**. A **perfect matching** is a matching such that $V(M) = V$. Given a graph with a weight $w(v)$ on each vertex $v \in V$, a **weighted matching** is a function $f : E \rightarrow \mathbb{N}_0 = \{0, 1, \dots\}$, such that for each $v \in V$, $\sum_{u \in \Gamma(v)} f(uv) \leq w(v)$. A weighted matching is **perfect** if $\sum_{u \in \Gamma(v)} f(uv) = w(v)$ for each $v \in V$.

2.4.1 Matchings and augmenting paths

Definition 2.4.2. Given a matching M in a graph $G = (V, E)$, a path P in G is **alternating** if its edges are alternatingly in and out of M . An **alternating cycle** is

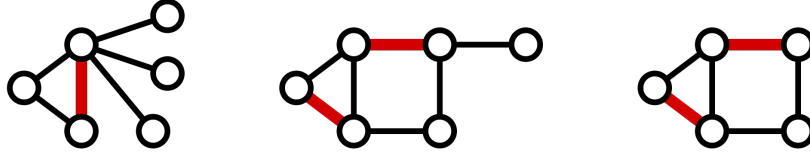


Figure 2.2: Maximal matchings. The third matching is also maximum [80].

defined similarly. An alternating path P is $(M-)$ **augmenting** if both the first and the last vertex are unsaturated.

Definition 2.4.3. If A and B are sets, we let $A \oplus B = (A - B) \cup (B - A)$ be their **symmetric difference**.

Theorem 2.4.4 (Berge [10]). A matching M in a graph G is a maximum matching in G if and only if G has no M -augmenting paths.

The Hopcroft-Karp algorithm uses this to find a maximum matching in a bipartite graph $G = (V, E)$ in $O(\sqrt{|V|}|E|)$ time [69]. Edmonds [36] extends this approach to general graphs $G = (V, E)$ with time complexity $O(|V|^2|E|)$. Micali and Vazirani [79], Vazirani [114] improve this approach to a $O(\sqrt{|V|}|E|)$ running time.

2.4.2 Matchings in bipartite graphs

A trivially necessary condition for matchings in bipartite graphs, is also sufficient, a result shown by Hall [56].

Theorem 2.4.5 (Hall's Marriage Theorem). Let $G = (V_1 + V_2, E)$ be a bipartite graph. G contains a perfect matching if and only if $|V_1| = |V_2|$ and,

$$|X| \leq |\Gamma(X)|, \quad \forall X \subseteq V_1 \quad (2.6)$$

Proof. See for example [17] or [56] for the original proof. $\square$

Eq. 2.6 is known as Hall's condition. A corollary of Hall's theorem is the so-called defect version, observed by Ore [85].

Corollary 2.4.6 (Defect version of Hall [85]). Let $G = (V_1 + V_2, E)$ be a bipartite graph and $0 \leq d \leq |V_1|$ be an integer. G contains a matching of size $|V_1| - d$ if and only if

$$|X| \leq |\Gamma(X)| + d, \quad \forall X \subseteq V_1 \quad (2.7)$$

Proof. Create G' , by adding d distinct dummy vertices D to V_2 , adjacent to all vertices in V_1 . There is a matching in G , with at most d unmatched vertices from $V_1 \Leftrightarrow$ there is a perfect matching in $G' \Leftrightarrow |X| \leq |\Gamma_{G'}(X)|, \forall X \subseteq V_1 \Leftrightarrow |X| \leq |\Gamma_G(X)| + d, \forall X \subseteq V_1. \quad \square$

Matchings are well studied objects in graph theory and there is much more known about them. See for example Bollobás [17], Lovász and Plummer [77], or any of the more general graph theory books mentioned above.

2.5 Flow graphs

Definition 2.5.1. A **directed graph** or **digraph** G is a triple consisting of a vertex set $V(G)$, an edge set $E(G)$ and a function assigning each edge an ordered pair of vertices. The first vertex is the **tail** of the edge and the second is the **head**. Together they are the **endpoints**. We say an edge is an edge from its tail to its head.

Definition 2.5.2. Let v be a vertex in a digraph. The **out-degree** $d^+(v)$ is the number of edges with tail v . The **in-degree** $d^-(v)$ is the number of edges with head v . The **degree** of v is $d(v) = d^+(v) + d^-(v)$

Definition 2.5.3. Let $G = (V, E)$ be a directed graph. For $v \in V$, $\Gamma_G^+(v) = \Gamma^+(v) = \{w : \overrightarrow{vw} \in E\}$ is the **out-neighbourhood** of v and $\Gamma_G^-(v) = \Gamma^-(v) = \{w : \overrightarrow{wv} \in E\}$ is the **in-neighbourhood** of v . For $A \subseteq V$ and $* \in \{+, -\}$, $\Gamma^*(A) = \bigcup_{v \in A} \Gamma^*(v) \setminus A$.

Definition 2.5.4. A **flow graph** $G = (V, E)$ is a finite directed graph in which every edge $\overrightarrow{uv} \in E$ has a non-negative, real-valued capacity $c(u, v)$. If $\overrightarrow{uv} \notin E$, we set $c(u, v) = 0$. We distinguish two vertices $s, t \in V$, called the **source** and the **sink**.

Definition 2.5.5. A **flow** in a flow graph $G = (V, E)$ is a function $f : V \times V \rightarrow \mathbb{R}$ with the following properties

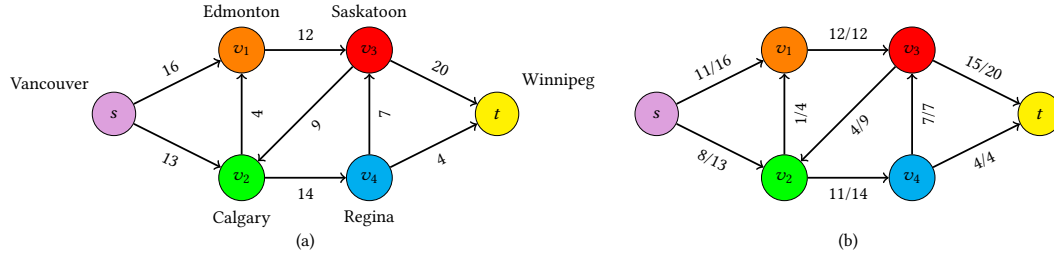
1. $f(x, y) \leq c(x, y) \quad \forall x, y \in V$,
2. $f(x, y) = -f(y, x) \quad \forall x, y \in V$,
3. $\sum_{y \in \Gamma^+(x)} f(x, y) - \sum_{y \in \Gamma^-(x)} f(y, x) = 0$ for all $x \in V \setminus \{s, t\}$, and
4. $\sum_{y \in \Gamma^+(s)} f(s, y) - \sum_{y \in \Gamma^-(s)} f(y, s) = \sum_{y \in \Gamma^-(t)} f(y, t) - \sum_{y \in \Gamma^+(t)} f(t, y)$.

Property 3 is also referred to as **conservation of flow**; every vertex except s and t have a net flow of zero. For $x \in V$, let $f^+(x) = \sum_{y \in \Gamma^+(x)} f(x, y)$ and $f^-(x) = \sum_{y \in \Gamma^-(x)} f(y, x)$, so equivalently we could write $f^+(x) = f^-(x)$. The **volume** of a flow f is $v(f) = f^+(s) - f^-(s)$.

Chapter 2. Preliminaries

Vertex s is referred to as the source, because traditionally it has a positive net flow, while vertex t is called the sink, because traditionally its net flow is negative.

Example 1 (The Lucky Puck Company [28]).



- (a) A flow graph $G = (V, E)$ for the Lucky Puck Company's trucking problem. The Vancouver factory is the source s , and the Winnipeg warehouse is the sink t . Pucks are shipped through intermediate cities, but only $c(u, v)$ crates per day can go from city u to city v . Each edge is labelled with its capacity.
- (b) A flow r in G with volume $v(f) = 19$. Only positive net flows are shown. If $f(u, v) > 0$, edge $\vec{uv}$ is labelled by $f(u, v)/c(u, v)$. (The slash notation is used merely to separate the flow and capacity; it does not indicate division.) If $f(u, v) = 0$, edge $\vec{uv}$ is labelled only by its capacity

Definition 2.5.6. Let $G = (V, E)$ be a flow graph with source s and sink t and capacity function $c : V \times V \rightarrow \mathbb{N}_0$, and let $S, T \subseteq V$ be disjoint. Then the **capacity** of the cut set $(S, T) = \{\vec{uv} \in E : u \in S, v \in T\}$ is

$$c(S, T) = \sum_{\vec{uv} \in (S, T)} c(u, v).$$

An $s - t$ **cut** is a partition of V into S and T , such that $s \in S$ and $t \in T$. Similarly, given a flow f , and cut set (S, T)

$$f(S, T) = \sum_{\vec{uv} \in (S, T)} f(u, v).$$

Ford and Fulkerson showed the following important result for the undirected case [44] and Dantzig and Fulkerson showed it for the directed case [30, 31].

Theorem 2.5.7 (Min cut/max flow theorem). *The maximum volume of an $s - t$ flow is equal to the minimum capacity over all $s - t$ cuts.*

And this implies the following

Theorem 2.5.8 (Integral flow theorem [30, 31]). *If each edge in a flow network has integral capacity, then there exists an integral maximal flow.*

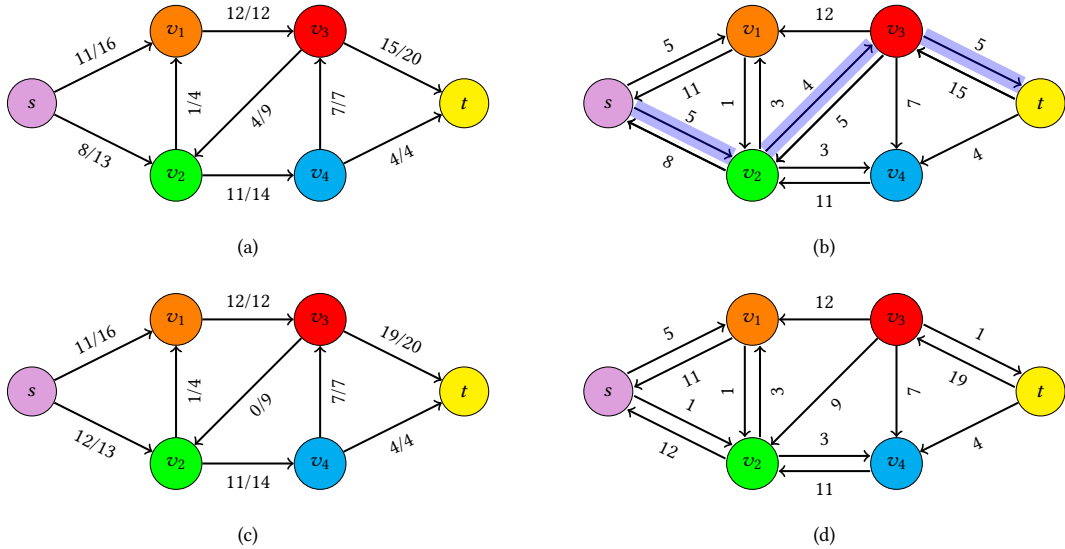
Definition 2.5.9. *Given a flow graph $G = (V, E)$ and a flow $f : E \rightarrow \mathbb{R}$, the **residual flow graph** $G_f = (V_f, E_f)$, where $V_f = V$ and for each edge $\overrightarrow{xy} \in E$,*

- (i) *if $f(x, y) < c(x, y)$, $\overrightarrow{xy} \in E_f$ with capacity $c_f(x, y) = c(x, y) - f(x, y)$, and*
- (ii) *if $f(x, y) > 0$, $\overleftarrow{xy} \in E_f$ with capacity $c_f(y, x) = f(x, y)$.*

An **augmenting path** is a path $(u_1, u_2, \dots, u_k)$ in the residual flow graph, where $u_1 = s$, $u_k = t$ and $c_f(u_i, u_{i+1}) > 0$ for $1 \leq i \leq k - 1$. The **residual capacity** of an augmenting path is $\min_i c_f(u_i, u_{i+1})$.

Given flow f and an augmenting path P in G_f with residual capacity r , note that f can be augmented by increasing the flow along each edge in P by r .

Example 2 (Augmenting the flow [28]).



- (a) *The flow network G and flow f of Example 1(b).*
- (b) *The residual network G_f with augmenting path P shaded; its residual capacity is $c_f(P) = c(v_2, v_3) = 4$.*
- (c) *The flow in G that results from augmenting along path P by its residual capacity 4.*
- (d) *The residual network induced by the flow in (c).*

We can use the residual flow graph because the following is true.

Theorem 2.5.10 ([4, 28, 100]). *A flow f in a flow graph G is maximum if and only if there is no augmenting path in the residual flow graph G_f .*

Flows are a well-studied phenomenon in combinatorial optimisation. Ahuja et al. [4] is a book dedicated to flow problems, but they also feature prominently in for example Schrijver [100] and Cormen et al. [28].

2.5.1 Weighted flow graphs

Some flow graphs also have a weight, or a cost $w(u, v) \in \mathbb{R}$ associated with each edge $\vec{uv}$.

Definition 2.5.11. *Given a weighted flow graph $G = (V, E)$ and a flow f in G the **weight** or **cost** of the flow is*

$$w(f) = \sum_{\vec{uv} \in E} f(u, v)w(u, v).$$

Definition 2.5.12. *Given a weighted flow graph $G = (V, E)$ and a flow $f : E \rightarrow \mathbb{R}$, the **residual flow graph** $G_f = (V_f, E_f)$, where $V_f = V$ and for each edge $\vec{xy} \in E$,*

- (i) *if $f(x, y) < c(x, y)$, $\vec{xy} \in E_f$ with capacity $c_f(x, y) = c(x, y) - f(x, y)$ and weight $w(x, y)$, and*
- (ii) *if $f(x, y) > 0$, $\overleftarrow{xy} \in E_f$ with capacity $c_f(y, x) = f(x, y)$ and weight $-w(x, y)$.*

*An **augmenting path** is a path $(u_1, u_2, \dots, u_k)$ in the residual flow graph, where $u_1 = s$, $u_k = t$ and $c_f(u_i, u_{i+1}) > 0$ for $1 \leq i \leq k - 1$.*

2.5.2 Ford-Fulkerson Algorithm

The Ford–Fulkerson method or Ford–Fulkerson algorithm is an algorithm which computes the maximum flow in a flow graph. It is often referred to as a method rather than an algorithm as the approach to finding augmenting paths in the residual flow graph is specified in several implementations with different running times [28]. It was published in Ford and Fulkerson [44].

In each iteration of Ford-Fulkerson, some augmenting path P is found in the residual flow graph of flow graph $G = (V, E)$ and we increase the flow f along each edge on P by the residual capacity of P . An augmenting path can be found through a depth first search or through a breadth first search, which has complexity $O(|E|)$. As each pass increases the flow by an integer amount which is at least 1, the

Ford-Fulkerson algorithm finds a maximum integer flow in flow graphs $G = (V, E)$ with integral flow in $O(|E|v(f))$ time, where $v(f)$ is the volume of the maximum flow. More details about flows can be found in for example Ahuja et al. [4], Cormen et al. [28], Schrijver [100].

2.6 Treewidth and pathwidth

The notions of pathwidth and treewidth were first introduced by Robertson and Seymour [89, 90], and played an important role in their work on graph minors. Graphs with small treewidth and pathwidth are ‘nice’, because they can be decomposed into small subgraphs that have very little overlap. By using dynamic programming algorithms, many problems that have been proven to be NP-complete/hard, such as finding a maximum independent set, can be solved efficiently, either exact or approximate, on graphs of small treewidth.

2.6.1 Treewidth

In this section vertices will be $v \in V(G)$ and nodes $i \in T(G)$ will be the elements of the decomposition tree of G . The treewidth of a graph G is so to say a measure of how much G is like a tree.

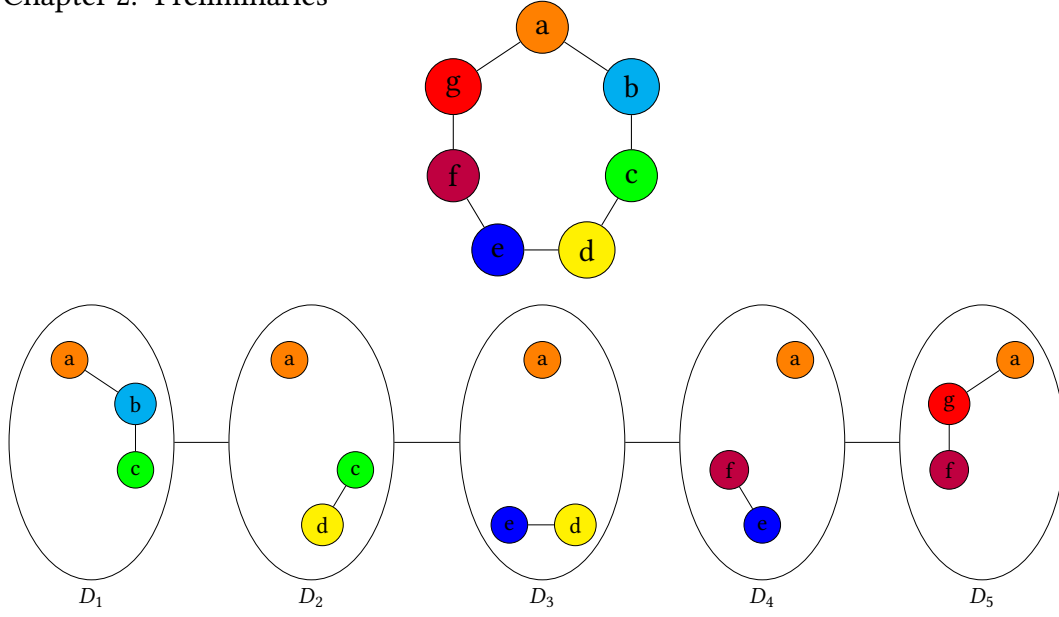
Definition 2.6.1. *Given a graph $G = (V, E)$ a **tree decomposition** is a pair $(\{X_i : i \in I\}, T)$, where T is a tree and $i \in I$ are the nodes of the tree satisfying*

1. $\bigcup_{i \in I} X_i = V$, i.e. each $v \in V$ is part of at least one node.
2. If $i, j, k \in I$ and k is on the path from i to j , then $X_i \cap X_j \subseteq X_k$.
3. For every $e = uv \in E$ there is a node i , such that X_i contains both u and v , i.e. vertices u and v are adjacent only when the corresponding subtrees $T_u = \{i : u \in X_i\}$ and $T_v = \{i : v \in X_i\}$ have at least one node in common.

For $i \in I$, X_i will also be referred to as ‘bag’ or ‘boundary’.

Given decomposition $(\{X_i : i \in I\}, T)$, we will often shorten notation by saying that node $i \in T$. See Fig. 2.3 for a tree decomposition of C_7 and Fig. 2.5 for another example of a tree decomposition.

Definition 2.6.2. *The **width** of a tree decomposition (X, T) is $\max\{|X_i| - 1 : i \in I\}$. The **treewidth** of G is the minimum width of any tree decomposition of G .*

Figure 2.3: An optimal tree decomposition of C_7 .

It is NP-complete to determine whether a graph G has treewidth at most k , when k is a variable [6]. When k is a fixed constant, Arnborg et al. [6] showed that the graphs with treewidth at most k can be recognised in polynomial time. Bodlaender [14] showed that given a constant k and a graph G , it can be decided in linear time whether G has treewidth at most k and, if so, find a tree decomposition of width at most k . In both cases there is an exponential in k hidden in the constant. Given a tree decomposition, it can be turned into a rooted binary tree decomposition in linear time [76], so in this thesis we will always assume that a tree decomposition is a rooted binary tree.

2.6.2 Some properties of tree decompositions

Lemma 2.6.3 ([34]). *Let $G = (V, E)$ be a graph that contains a clique C . In any tree decomposition $(\{X_i : i \in I\}, T)$ of G , there is a node $i \in I$, such that X_i contains C .*

Proof. Root T arbitrarily. For $w \in C$ let X_w denote the bag containing w with minimum depth (i.e. closest to the root). Let X_t be a bag from the collection $\{X_w : w \in C\}$ with maximum depth in the tree T . Suppose there is $v \in C$ such that $v \notin X_t$. Then X_v is not a descendant of X_t , because X_t has maximum depth. There exists a node t' such that $v, t \in X_{t'}$. Let $t'1, 2 \dots kv$ be the path between t' and v . Then by property 2 for all $i \in [k]$, $v \in X_i$. So $X_t \notin \{X_1, X_2, \dots, X_k\}$, because $v \notin X_t$ by assumption. This implies that t' is not a descendant of t . Let i be the first node on the path between t' and t . Then $t \in X_i$ by property 2, contradicting the definition of

X_t . Thus for all $v \in C$, $v \in X_t$ and so $C \subseteq X_t$ (modified from [59]). $\square$

Definition 2.6.4 ([60, 116]). *The class of **k-trees** is defined recursively as follows:*

1. *The complete graph on $k + 1$ vertices is a k -tree.*
2. *A graph G with $n + 1$ vertices ($n \geq k + 1$) is a k -tree, if there exists a vertex $v \in V(G)$, such that $G \setminus v$ is a k -tree and v is adjacent to the vertices of a k -clique in $G \setminus v$.*

Definition 2.6.5. *A partial k -tree is a graph that contains all the vertices and a subset of the edges of a k -tree.*

Lemma 2.6.6 ([39, 112]). *A graph has treewidth k if and only if it is a partial k -tree.*

Proof. First we will prove that every partial k -tree has treewidth at most k , by proving the stronger claim that k -trees have treewidth at most k by induction on $|V|$. Suppose G is a complete graph on $k + 1$ vertices, then G has treewidth $\leq k$. Now suppose G is a k -tree with $|V| > k + 1$ and the claim is true for smaller graphs. G is not the complete graph, i.e. there is $v \in V$ s.t. $G \setminus v$ is a k -tree and $\Gamma(v)$ is a k -clique in G . By induction $G \setminus v$ has treewidth $\leq k$ with a corresponding decomposition tree T . By Lemma 2.6.3 there is $i \in I$ such that $\Gamma(v) \subseteq X_i$. Add a new node j with $X_j = \Gamma(v) \cup \{v\}$ to T as a leaf to i . This is a tree decomposition of width $\leq k$.

Now we will prove that every graph $G = (V, E)$ with a tree decomposition $(\{X_i : i \in I\}, T)$ of width at most k is a partial k -tree, by showing that there is k -tree $H = (V, E')$, such that for each node $i \in I$, $G[X_i]$ is a subgraph of a $(k + 1)$ -clique in H . We do this by induction on $|V|$. If $|V| = k + 1$, then G is a subgraph of a complete graph on $k + 1$ vertices and a tree decomposition consists of a single bag containing all vertices. Assume $|V| > k + 1$ and that $(\{X_i : i \in I\}, T)$ is a tree decomposition of width $\leq k$ and that the claim holds for smaller graphs. Take a leaf node $l \in I$ with unique neighbour j . If $X_l \subseteq X_j$, remove l and consider $(\{X_i : i \in I \setminus l\}, T)$ instead. Let $v \in X_l \setminus X_j$. Consider a different node i , then by property 2, $v \notin X_i$, because $v \notin X_j$. Thus $\Gamma(v) \cup \{v\} \subseteq X_l$. Moreover, $|X_l| \leq k + 1$ and therefore $|\Gamma(v)| \leq k$.

The pair $(\{X_i \setminus v : i \in I\}, T)$ is a tree decomposition of $G \setminus v$ with width at most k . Induction gives a k -tree H' , such that for each node $i \in I$, $G[X_i]$ is a subgraph of a $(k + 1)$ -clique in H' . Thus $G[\Gamma(v)]$ is a subset of a k -clique $C \subseteq H'$ (as $|\Gamma(v)| \leq k$). Adding v to H' , adjacent to all $w \in C$, gives the desired k -tree H [15, 60]. $\square$

2.6.3 Treewidth and graph minors

A very deep and important result in the study of graph minors is presented in a series of papers by Robertson and Seymour entitled Graph Minors. I. [89] - Graph Minors. XX. [92]. They continued to work on this topic and have found many interesting and valuable results [93, 94].

Theorem 2.6.7 (Robertson and Seymour). *Let $\mathcal{G}$ be a class of graphs closed under taking minors. Then there exists a finite set of graphs called the obstruction set or the set of forbidden minors of $\mathcal{G}$, such that for every graph H , H belongs to $\mathcal{G}$ if and only if H does not contain a graph from the obstruction set of $\mathcal{G}$ as a minor.*

This connects to treewidth through the following lemma:

Lemma 2.6.8. [15] *If H is a minor of G , then $\text{treewidth}(H) \leq \text{treewidth}(G)$.*

Theorem 2.6.9 (Robertson and Seymour [91]). *For every finite value of k , the graphs of treewidth at most k may be characterized by a finite set of forbidden minors, each of which includes at least one planar graph.*

- For $k = 1$, the unique forbidden minor is a triangle [12].
- For $k = 2$, the unique forbidden minor is K_4 [12].
- For $k = 3$, the forbidden minors are K_5 , the octahedron, the pentagonal prism, and the Wagner graph, as depicted in Fig. 2.4 [97].

The obstruction sets are not known for larger than k . It is believed that the size of the set of forbidden minors grows exponentially in k , so it is probably more difficult to characterise for larger values of k [63].

We now have enough information to prove the following:

Lemma 2.6.10. *A connected graph G with $n \geq 2$ vertices has treewidth one if and only if G is a tree.*

Proof. First it will be shown that a tree has treewidth one, by induction on n , the number of vertices in tree G . Consider the base case $n = 2$, then one node with two vertices suffices. Now suppose $n > 2$ and the claim holds for smaller n . Remove leaf v , then $G \setminus v$ has $n - 1$ vertices and is still a tree. By the inductive assumption this tree has treewidth one. Take an optimal tree decomposition T . Let u be the unique neighbour of v in G . Then create a new bag $X_j = \{u, v\}$ and attach j to i in T , such that $u \in X_i$. We now have a tree decomposition of width one for the tree G .

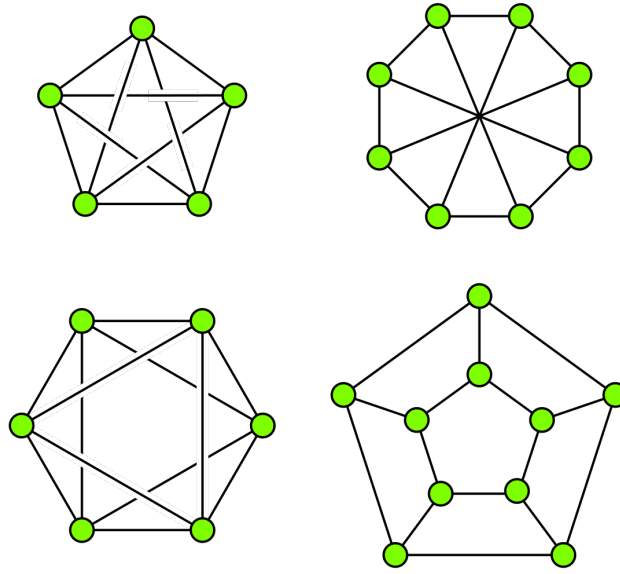


Figure 2.4: The forbidden minors of graphs with treewidth at most 3 [37].

Because a tree is maximal acyclic graph, any graph that is connected and is not a tree must contain a cycle, and thus contain a triangle as a minor. By Lemma 2.6.8 any graph with a cycle as minor has treewidth at least two. $\square$

Trees are also known as 1-trees.

2.6.4 Pathwidth

The notion of pathwidth is very similar to the notion of treewidth, except that the decomposition is a path, rather than a tree. Though that makes algorithms on this structure simpler, it makes the constraints on the graph stronger. It is easy to see that every graph with pathwidth k , has treewidth at most k . The converse is naturally not true. For example the graph in Fig. 2.5 has treewidth two and pathwidth three.

Definition 2.6.11. A **path decomposition** of a graph $G = (V, E)$ is a sequence of subsets of vertices $(D_1, D_2, \dots, D_m)$ such that

1. $\bigcup_{1 \leq t \leq m} D_t = V$.
2. for all edges $vw \in E$, there exists $t \in [m]$, such that $v, w \in D_t$.
3. for all $t, q, r \in [m]$: if $t \leq q \leq r$, then $D_t \cap D_r \subseteq D_q$. [13]

For an example of a path decomposition see Fig. 2.5.

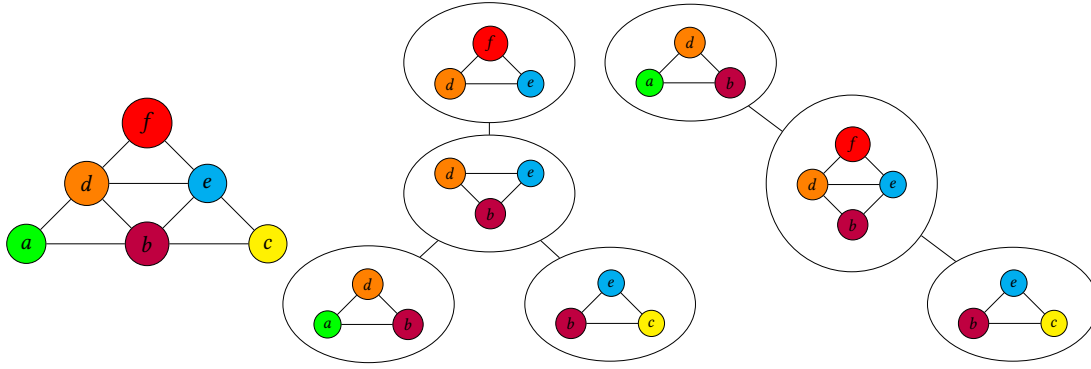


Figure 2.5: A graph with a tree and path decomposition.

Definition 2.6.12. The **width** of a path decomposition $(D_1, D_2, \dots, D_m)$ is $\max\{|D_i| - 1; i \in [m]\}$. The **pathwidth** of G is the minimum width of any path decomposition of G .

For more information on treewidth and pathwidth, see Bodlaender [14, 15] for two survey papers, Erickson [39], Fiala [43], Holzer [68], Krause [76], Thomas [110] for lecture notes on the subject and Bodlaender [12, 13], Hicks et al. [63], Noble [82] for some papers with general results on graphs with bounded treewidth.

2.7 Outline

This concludes the introductory chapters of this thesis. Part I (Chapters 3-6) describes colour degree matrices of graphs with few cycles and Part II (Chapters 7-11) describes different models for the reverse radar problem.

Part I

Colour degree matrices of graphs with few cycles

3

Discrete tomography and realising colour demand matrices

In this chapter the two-sensor problem will be linked to discrete tomography and colour degree matrices. In the two-sensor problem, we want to reconstruct the original layout of c -coloured squares on an n by n grid, given its horizontal projections A and vertical projections B . This problem is closely related to Japanese puzzles, or nonograms [81], but is more formally known as two-dimensional *discrete tomography* [9] with c colours, as will be outlined in Section 3.1. For an example see Fig. 3.2(a). The number of sensors corresponds to the dimension.

This problem is also equivalent to finding a bipartite edge-coloured graph, which realises A and B , as will be shown in Section 3.2. This is the way this problem will be represented in the remainder of this thesis.

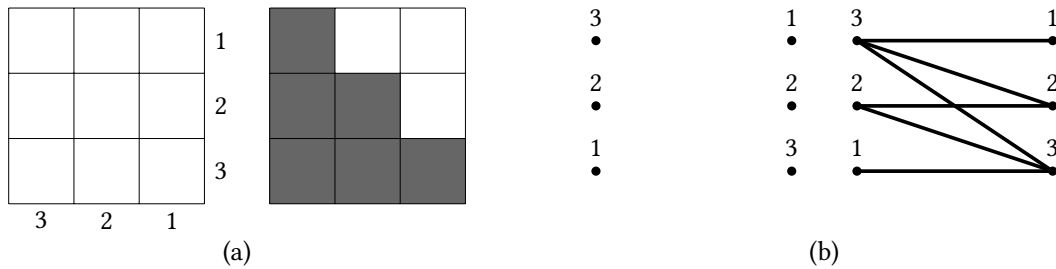


Figure 3.1: (a) An instance of two dimensional discrete tomography with a solution. (b) Same instance as a bipartite demand sequence, with a bipartite graph realisation.

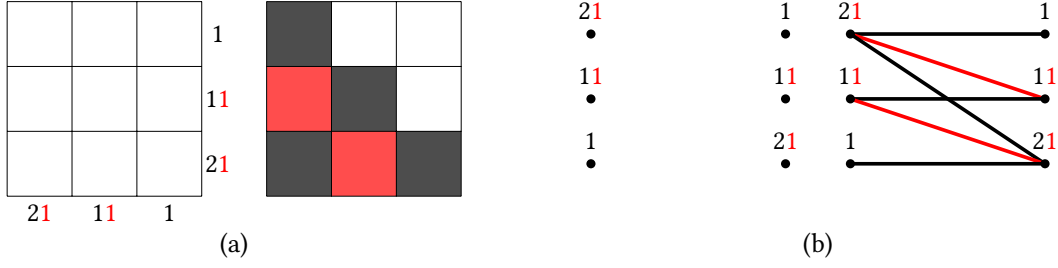


Figure 3.2: (a) An instance of two dimensional discrete tomography with solution. (b) Same instance as a bipartite demand matrix, with a bipartite graph realisation.

3.1 Discrete tomography

Discrete tomography is the problem of reconstructing finite subsets of the integer lattice from a set of projections. It has many applications, ranging from image processing to material science [62].

We will consider the following problem in 2 dimensions. Let m , n , and c be positive integers. Given $(h_{i,q} : i \in [m], q \in [c])$ and $(v_{j,q} : j \in [n], q \in [c])$, can we find an $m \times n$ grid P where each cell contains either no colour, or a colour from $[c]$ such that $h_{i,q}$ ($v_{j,q}$) is the number of cells with colour q in row $i \in [m]$ (column $j \in [n]$) [61]? See for example Fig. 3.1(a). This can be generalised to $d \geq 3$ dimensions, but for any dimension strictly greater than 2, the problem is NP-hard [24, 47]. For 2 dimensions, the case $c = 1$ can be solved in polynomial time [45–47, 96]. Recently it was shown that for any fixed $c \geq 2$ discrete tomography is NP-hard [24, 35, 47]. For an example of a discrete tomography problem with 2 colours, see Fig. 3.2.

3.2 Colour degree matrices

3.2.1 Demand sequences

Definition 3.2.1. A **demand sequence**, is a sequence $D = (d_1, d_2, \dots, d_n)$ of non-negative integers. A demand sequence is **graphical**, or a **degree sequence**, if there exists a (simple) graph G on n vertices, where vertex $i \in [n]$ has degree d_i . It is said that G **realises** D , or that it is a **realisation** of D .

See for example Fig. 3.1(b). Sufficient conditions for a demand sequence to be realisable were discovered by Senior [103], and have been restated many times since; first by Senior himself in a more mathematically oriented paper [102], then by

Bäbler [7] and later by Ore [86, 87]. In 1955, Havel gave a recursive necessary and sufficient condition [58]. Hakimi independently found the same result in 1962 [53]. This recursive condition yields a polynomial time algorithm. In 1960, Erdős and Gallai gave necessary and sufficient conditions for a sequence to be graphical [38] by giving a bound on all partial demand sums. Many equivalent criteria have been found since [74, 99, 104]. In addition to focusing on the possibility of a realisation, often properties of realisations are considered.

Definition 3.2.2. *A property of a realisation of a degree sequence D is **forcible**, if it exists in every realisation of D and **potential** if there exists some realisation of D with this property.*

3.2.2 Colour degree matrices

Definition 3.2.3. *Let c and n be positive integers. An $n \times c$ **demand matrix** is a matrix $D = (d_{v,q} : v \in [n], q \in [c])$ of non-negative integers. We call D a **colour degree matrix** if there exists a c -edge coloured graph G on $[n]$, which **realises** D ; that is, each vertex $v \in [n]$ has $d_{v,q}$ incident edges coloured q , for each $q \in [c]$. If such a graph G exists, G is a realisation of the colour degree matrix D .*

Note that if any $d_{v,q} \geq 2$, this colouring is not proper. See for example Fig. 3.2(b). Finding a realisation for a demand matrix is also known as finding *edge-disjoint realisations* [51, 52], *edge packing* [21] and *degree constrained edge-partitioning* [9].

3.2.3 Connection with discrete tomography

De Werra et al. show that the problem of discrete tomography in two dimensions is equivalent to finding an edge-coloured bipartite graph with colour degree matrices $(h_{i,q} : i \in [m], q \in [c])$ and $(v_{j,q} : j \in [n], q \in [c])$ for vertex class A and B , such that $|A| = m$ and $|B| = n$ [33]. See Fig. 3.1(b) and 3.2(b) for examples.

Bentz et al. showed that finding a general realisation of a demand matrix is at least as hard as finding a bipartite realisation of two disjoint demand matrices A and B [9]. The argument is that given an algorithm to find a general realisation of a demand matrix, a bipartite realisation can be found of $A = \{d_{v,q} : v \in [m], q \in [c]\}$ and $B = \{d_{v,q} : v \in [n], q \in [c]\}$, by ensuring that $[m]$ and $[n]$ form a clique in two new colours, say colour 1 and 2. This is easily done by setting $d_{v,1} = m - 1$ and $d_{v,2} = 0$ for each $v \in [m]$ and $d_{v,1} = 0$ and $d_{v,2} = n - 1$ for each $v \in [n]$, as then

each $v \in [m]$ ($v \in [n]$) is adjacent to all vertices in $[m] \setminus v$ ($[n] \setminus v$). Then for any realisation there is no edge coloured $q \in [c]$ in $[m]$ and $[n]$, as $[m]$ is a clique in colour 1 and $[n]$ is a clique in colour 2. It is thus enough to consider general demand matrices. Note that this means that in general deciding whether D is a colour degree matrix is NP-complete.

However, we may be interested in realisations where the graph has some special structure, and restricting the solution class may yield a tractable problem. Examples are colour degree matrices of span at most two by Guíñez et al. [52] or the existence of certain structures in the resulting graph [101]. Guíñez gives more tractable examples in his PhD thesis [51]. Useful surveys of results on colour degree matrices are in references [42, 73, 117]. Previous work has often focused on packing two or three degree sequences (fixing c at two or three) [9, 29, 52], but in this thesis we will consider colour degree matrices of graphs with few cycles for general c . This is of course still a rather restricted class of graphs for example for applications in tomography, but at least it is a step beyond forests, and we investigate a richer structure.

3.3 Outline

We will focus on colour degree matrices of graphs with at most one cycle. As there might be realisations with more than one cycle in addition to a realisation with at most one cycle, this is a potential property. In Chapter 4 we start by considering graphs without cycles: forests. We state a classic result for a sequence of non-negative integers to be the degree sequence of a forest and give our own proof. Bentz et al. gave the necessary and sufficient conditions for D to be the colour degree matrix of a 2-coloured tree [9]. Carroll and Isaak gave necessary and sufficient conditions for D to be realisable by a c -edge coloured forest for general c [22]. We give a short proof of this result, and then take the first step beyond forests in Chapter 5.

Again a classic result for a sequence of non-negative integers to be the degree sequence of a unicyclic graph is given. This is compared to a new, alternative formulation of this result. The conditions in this result are equivalent, but give a bound on the largest degrees, rather than restricting the number of vertices with degree one. A short proof is provided.

Additionally, we give necessary and sufficient conditions for D to be the colour degree matrix of a graph with at most one cycle. First we show that the ‘unicycle con-

ditions' are necessary. Then we prove some implications of the unicycle conditions. Before we proceed to the full proof we consider some 'directly forced' structures in the resulting graph, forced by certain types of subsets. We demonstrate that these structures are forcible and which conditions imply their existence. These results are needed in various places of the proof of the main theorem. We continue with results analogous the main theorem, pertaining to multi-graphs and pseudoforests with at most k cyclic components.

Next, we discuss some related algorithmic questions in Chapter 6. First we demonstrate that the conditions can be checked in polynomial time by 'translating' them to a flow problem in a related bipartite graph. We end this chapter by showing that a realisation as required can be found in polynomial time.

Table 3.1 shows, given a demand matrix the known time complexities to check for the existence of a certain realisation and to find this realisation. References to the relevant papers and section numbers are given in parentheses.

Graph Type	Uncoloured		Coloured	
	Existence	Realisation	Existence	Realisation
Forest	$O(n)$ [57, 78]	$O(n)$ [57, 78]	$O(n^2)$ (6.4.2)	$O(n^3)$ (6.5.1)
A graph with at most 1 cycle	$O(n)$ [16]	$O(n)$ [16]	$O(n^2)$ (6.1,6.3)	$O(n^3)$ (6.5.2)
Unicyclic graph $\forall v, d_{v,[c]} = 2$	$O(n)$ [16]	$O(n)$ [16]	$O(n^2)$ (6.1,6.3)	$O(n^2)$ (5.5.1)
Pseudoforest with at most k cycles	$O(n)^*[16]$	$O(n)$ [16]	$O(n^2)$ (6.4.5)	$O(n^3)$ (6.5.4)
Unicyclic multi-graph	$O(n)$ [8]	$O(n)$ [8]	$O(n^2)$ (6.5.3)	$O(n^3)$ (6.5.3)

Table 3.1: An overview of time complexities for the demand matrix decision problem and for finding the required realisation.

*Note that this is the same as an uncoloured graph with at most one cycle.

4

Colour degree matrices of forests

In this chapter we will give necessary and sufficient conditions for a demand matrix to be realisable by a forest. We briefly consider the monochromatic case and then state a result by Bentz et al. [9] and Carroll and Isaak [22]. We give (a tidied up version of) the proof in [22], as we will use similar arguments for graphs with at most one cycle in Chapter 5.

4.1 Degree sequences of forests

The monochromatic case ($c = 1$) is a degree sequence. Necessary and sufficient conditions for a degree sequence to be realisable by a forest have been found independently many times. See for example Harary [57], Menon [78].

Lemma 4.1.1. *Let $D = (d_1, d_2, \dots, d_n)$, $n \geq 1$, be a sequence of strictly positive integers such that $d_1 \geq d_2 \geq \dots \geq d_n \geq 1$, then D is the degree sequence of a forest if and only if*

$$\sum_{i=1}^n d_i \leq 2(n-1), \text{ and}$$
$$\sum_{i=1}^n d_i \equiv 0 \pmod{2}.$$

Proof. The conditions are clearly necessary, since a forest on n vertices has at most $n - 1$ edges. We will show sufficiency by induction on n . If $n = 1, 2$, it is easy to

check that the statement holds. Suppose $n \geq 3$, $\sum_{i=1}^n d_i = 2n - 2k$, $k \geq 1$ and that the statement is true for smaller values. Note that $d_n = 1$. If $d_1 = 1$, $d_i = 1$ for all $i \in [n]$ and D can be realised by a disjoint union of edges. If $d_1 \geq 2$, let $D' = (d'_1, \dots, d'_{n-1})$, with $d'_1 = d_1 - 1$ and $d'_i = d_i$ for $2 \leq i \leq n-1$. Then $\sum_{i=1}^{n-1} d'_i = 2n - 2k - 2 = 2(n-1) - 2k$. By induction there exists a forest realisation F of D' . Appending a new vertex to a vertex with degree d'_1 extends F to a forest realisation of D . $\square$

4.2 Colour degree matrices of forests

When is a demand matrix the colour degree matrix of a forest? There are simple necessary conditions. Let c and n be positive integers, and consider an $n \times c$ demand matrix $D = (d_{v,q} : v \in [n], q \in [c])$. For each $I \subseteq [c]$, let $d_{v,I} = \sum_{q \in I} d_{v,q}$ be the demand of v for the set of colours I . Observe that $(d_{1,I}, \dots, d_{n,I})$ is the demand sequence we obtain when we ignore colours not in I and identify colours in I . Clearly it is necessary that each such sequence $(d_{1,I}, \dots, d_{n,I})$ is the degree sequence of a forest. These conditions were shown to be sufficient by Bentz et al. in [9] for the case $c = 2$, and by Carroll and Isaak in [22] for the general case.

Theorem 4.2.1. [9, 22] *The demand matrix $D = (d_{v,q} : v \in [n], q \in [c])$ is the colour degree matrix of a forest if and only if, for each $I \subseteq [c]$, the sequence $(d_{1,I}, \dots, d_{n,I})$ is the degree sequence of a forest.*

We may restate this theorem in an apparently more quantitative way.

Definition 4.2.2. For $I \subseteq [c]$, let $S(I) = S_D(I) = \{v : d_{v,q} > 0 \text{ for some } q \in I\}$ be the **support** of I and $s(I) = s_D(I) = |S_D(I)|$. Let $t(I) = t_D(I) = \sum_{q \in I} \sum_{v \in [n]} d_{v,q}$ be the **demand sum** of I . For a single element $q \in [c]$ we will write $s(q)$ and $t(q)$. Given $V \subseteq [n]$, let $D_{V,I} = (d_{v,q} : v \in V, q \in I)$ be the **demand** of V for I and $d_{V,I} = |D_{V,I}| = \sum_{v \in V} \sum_{q \in I} d_{v,q}$.

Theorem 4.2.3. [9, 22] *The demand matrix $D = (d_{v,q} : v \in [n], q \in [c])$ is the colour degree matrix of a forest if and only if*

- (1) $t(q)$ is even for each $q \in [c]$, and
- (2) $t(I) \leq 2s(I) - 2$ for each $I \subseteq [c]$ with $t(I) > 0$.

We will give a tidied up version of the proof of [22].

Definition 4.2.4. Let the **total demand** of a vertex $v \in [n]$ be $d_{v,[c]}$. Let I be **tree critical** with respect to D if $t_D(I) = 2s_D(I) - 2$.

4.2. Colour degree matrices of forests

Proof of Theorem 4.2.3. The conditions are obviously necessary, as condition (1) is the Handshaking Lemma and condition (2) follows, since a forest on k vertices has degree sum at most $2k - 2$. If $c = 1$, the theorem is true by Lemma 4.1.1. We proceed to prove sufficiency by induction on n . If $n = 1$ or $n = 2$, the conditions are obviously sufficient. Now assume $n \geq 3$ and that the conditions are sufficient for smaller values.

Without loss of generality assume there is no vertex with total demand zero and no colour with an empty support. There are two cases to consider.

The first case is that there exists a tree critical set $I \subsetneq [c]$. Let $I^c = [c] \setminus I$ and let $V = \{[n] \setminus S_D(I)\}$. Suppose $V = \emptyset$, then $t([c]) = t(I) + t(I^c) = 2n - 2 + t(I^c) > 2n - 2$, which contradicts condition (2). Thus $V \neq \emptyset$. Let $D_1 = (d_{v,q} : v \in S(I), q \in I)$. Let a be a new vertex with demands $d_{a,q} = \sum_{v \in S(I)} d_{v,q}$, for each $q \in I^c$ and $D_2 = (d_{v,q} : v \in V \cup \{a\}, q \in I^c)$. All conditions hold for D_1 by assumption and if the same holds for D_2 we can find a forest realisation of D : there exist realisations T_1 and T_2 of D_1 and D_2 respectively. The new vertex a is T_1 contracted to one vertex. Let T be the graph with a replaced by T_1 . All edges incident with a should be connected to any $a_i \in S_D(I)$, such that they all have the required degrees. It is easy to check that this can be done and that T realises D . Moreover, any cycle in T would yield either a cycle in T_1 or T_2 , so T is acyclic.

It is left to check that all conditions hold for D_2 . Condition (1) has not changed, so only condition (2) needs to be checked. Let $B \subseteq I^c$. If $S_D(B) \cap S_D(I) = \emptyset$, then $a \notin S_{D_2}(B)$ and $t_{D_2}(B) = t_D(B) \leq 2s_D(B) - 2 = 2s_{D_2}(B) - 2$. Suppose $S_D(B) \cap S_D(I) \neq \emptyset$. Then $s_{D_2}(B) = s_D(I \cup B) - s_D(I) + 1$, as a has non-zero demand for B . As I and B are disjoint and $t_D(I) = 2s_D(I) - 2$, we obtain

$$t_{D_2}(B) = t_D(B) = t_D(I \cup B) - t_D(I) \leq 2s_D(I \cup B) - 2 - 2s_D(I) + 2 = 2s_{D_2}(B) - 2.$$

The second case is that no tree critical set smaller than $[c]$ exists. There exists a vertex with total demand one. Suppose not, then each vertex would have total demand at least two and by condition (2)

$$2n \leq t([c]) \leq 2n - 2,$$

which is a contradiction. So without loss of generality, let $d_{v,1} = 1$ and $d_{v,q} = 0$ for $q \neq 1$. By condition (1) there exists $w \neq v$, such that $d_{w,1} \geq 1$. Consider the following possibilities:

Chapter 4. Colour degree matrices of forests

- (i) $d_{x,[c]} = 1$ for all $x \in S(1)$.
- (ii) If there is a w with $d_{w,1} > 1$, pick such w .
- (iii) If $d_{w,1} = 1$ for all $w \in S_D(1)$ and there is $w \in S_D(1)$ such that $d_{w,q} > 0$ for some $q \neq 1$, pick such w .

It is easy to show these events are disjoint and cover all possible outcomes.

Suppose (i) holds, then $S(1) \cap S([c] \setminus 1) = \emptyset$ and $S(1)$ can be realised by a disjoint union of edges E . Let $D' = (d_{v,q} : q \in [c] \setminus 1, v \in [n] \setminus S(1))$. By induction a forest realisation F' of D' exists and $F' \cup E$ is a forest realisation of D .

Suppose (ii) or (iii) holds and we picked $w \in S(1) \setminus v$. Let $\tilde{D}(w)$ be obtained from D by removing v and reducing $d_{w,1}$ by one. Condition (1) obviously still holds. If condition (2) holds for all $I \subseteq [c]$, induction gives a forest realisation of $\tilde{D}$ and appending v to w with colour 1 gives a forest realisation of D . We have for all $I \subseteq [c] \setminus 1$

$$\tilde{t}(I) = t(I) \leq 2s(I) - 2 = 2\tilde{s}(I) - 2$$

by assumption. So it is enough to consider $I \subseteq [c]$ for which $1 \in I$.

If (ii) is true, for all $I \subseteq [c]$ for which $1 \in [c]$, $s_{\tilde{D}}(I) = s_D(I) - 1$ and we get

$$t_{\tilde{D}}(I) = t(I) - 2 \leq (2s_D(I) - 2) - 2 = 2s_{\tilde{D}}(I) - 2.$$

If (iii) is true and $I \subsetneq [c]$ contains 1, then $t_D(I) < 2s_D(I) - 2$ and because the demand sum is always even $t(I) \leq 2s_D(I) - 4$. Moreover, as $s_{\tilde{D}}(I) \geq s_D(I) - 2$:

$$t_{\tilde{D}}(I) = t_D(I) - 2 \leq (2s_D(I) - 4) - 2 \leq 2s_{\tilde{D}}(I) - 2.$$

Finally, $s_{\tilde{D}}([c]) = s_D([c]) - 1$ as $d_{w,[c]} \geq 2$, and

$$t_{\tilde{D}}([c]) = t_D([c]) - 2 \leq (2s_D([c]) - 2) - 2 \leq 2s_{\tilde{D}}([c]) - 2.$$

Thus the conditions hold for all $I \subseteq [c]$ and there is a forest realisation of D . $\square$

4.3 Conclusion

Chapter 5 gives corresponding results for graphs with at most one cycle, unicyclic multi-graphs and pseudoforests with at most k cyclic components. Chapter 6 outlines algorithms to check these conditions and to find a realisation if one exists.

Another extension to consider is demand sequences of directed graphs. Directed demand sequences can easily be linked to bipartite demand sequences, by letting

all vertices in one part have in-degree zero and all vertices in the other part have out-degree zero. The monochromatic case has been shown in for example [11] and Bentz et al. [9] showed a necessary and sufficient conditions for a demand matrix to be realisable by directed forest for $c = 2$. It is straightforward to extend Theorem 4.2.3 to directed graphs also, but that is outside the scope of this thesis.

5

Colour degree matrices of graphs with at most one cycle

In this chapter we will show necessary and sufficient conditions for a demand matrix to be realisable by a graph with at most one cycle. This result is also published in Hillebrand and McDiarmid [67]. First we consider the monochromatic case. In Section 5.2 we introduce the unicycle conditions. We show first that the unicycle conditions are necessary, then we show some equivalent conditions and finally we show some useful implications of the unicycle conditions. In Sections 5.3 and 5.4 we discuss some structures that are forcible and show what conditions imply them. These structures are needed in various places when proving that the unicycle conditions are sufficient in Section 5.5. In Sections 5.6 and 5.7 we consider corresponding results for multi-graphs and pseudoforests with at most k cyclic components.

5.1 Degree sequences of connected unicyclic graphs

A unicyclic graph is a graph with exactly one cycle. Necessary and sufficient conditions for a sequence of non-negative integers to be the degree sequence of a connected unicyclic graph were found by Boesch and Harary [16]:

Theorem 5.1.1. *Necessary and sufficient conditions that a sequence of non-negative integers $D = (d_1, d_2, \dots, d_n)$, $n \geq 1$, is the degree sequence of a connected unicyclic graph are*

- I $\sum_{i \in [n]} d_i = 2n$,
- II $d_i \geq 1$ for all $i \in [n]$, and
- III there are at least three $i \in [n]$ with $d_i > 1$.

Independently we found some equivalent conditions, which are given here with a short proof:

Lemma 5.1.2. *Let $D = (d_1, d_2, \dots, d_n)$, $n \geq 3$ and $d_1 \geq d_2 \geq \dots \geq d_n \geq 1$, be a sequence of integers. D is the degree sequence of a connected unicyclic graph if and only if*

- (I) $\sum_{i \in [n]} d_i = 2n$, and
- (II) $d_1 + d_2 \leq n + 1$.

They are indeed equivalent as condition II of Theorem 5.1.1 is included in the definition of Lemma 5.1.2. Moreover conditions (I) and (II) of Lemma 5.1.2 $\Leftrightarrow$ condition I and III of Theorem 5.1.1. Thus Theorem 5.1.1 is equivalent to Lemma 5.1.2.

Proof of Lemma 5.1.2. The necessity of condition (I) is clear. Suppose $d_1 + d_2 \geq n + 2$ and let $G = (V, E)$ be a realisation of D , where vertex $v_i \in V$ has degree d_i . Suppose v_1 and v_2 are adjacent in G : by the pigeon hole principle $|\Gamma(v_1) \cap \Gamma(v_2)| \geq 2$, which constitutes at least two cycles in G . Suppose v_1 and v_2 are not adjacent in G : $|\Gamma(v_1) \cap \Gamma(v_2)| \geq 4$ and G has at least two cycles. Thus condition (II) is necessary.

To establish sufficiency suppose conditions (I) and (II) hold. We proceed by induction on n . If $n = 3$ a triangle realises D . Suppose $n > 3$ and the conditions are sufficient for $\tilde{n} < n$. If $d_n \geq 2$, then $d_i = 2$ for all $i \in [n]$ by condition (I) and there is a realisation of D that is just a cycle. Otherwise $d_n = 1$. Let $\tilde{D}$ be a sequence on $\tilde{n} = n - 1$ vertices, where

$$\begin{aligned} \tilde{d}_1 &= d_1 - 1, \text{ and} \\ \tilde{d}_i &= d_i, \quad 2 \leq i \leq n - 1. \end{aligned}$$

Then $\sum_i \tilde{d}_i = 2n - 2 = 2\tilde{n}$. It is left to check that $\tilde{d}_1 + \tilde{d}_2 \leq \tilde{n} + 1$ and $\tilde{d}_2 + \tilde{d}_3 \leq \tilde{n} + 1$. In fact $\tilde{d}_1 + \tilde{d}_2 \leq n = \tilde{n} + 1$ by (II). Suppose $\tilde{d}_2, \tilde{d}_3 > \tilde{d}_1$ and $\tilde{d}_2 + \tilde{d}_3 > \tilde{n} + 1$. This implies $d_2 + d_3 \geq n + 1$ and $d_1 = d_2 = d_3 = \frac{n+1}{2}$ as $d_1 \geq d_2 \geq d_3$. Moreover n is odd, because this is not an integer otherwise. So $n \geq 5$ and

$$\sum_{i=1}^n d_i \geq 3 \left(\frac{n+1}{2} \right) + n - 3 = 2n + \frac{n-3}{2} > 2n.$$

Therefore conditions (I) and (II) hold for $\tilde{D}$ and a connected unicyclic realisation G can be found by induction. A connected unicyclic realisation of D is G with v_n attached to v_1 . $\square$

5.2 The unicycle conditions

When is a demand matrix D the colour degree matrix of a graph with at most one cycle? This is the question on which we focus in this section. As with forests, there are simple necessary conditions. Clearly, for each $I \subseteq [c]$, it is necessary that $(d_{1,I}, \dots, d_{n,I})$ is the degree sequence of a graph with at most one cycle. Similar to the definition of tree critical sets, we can define cycle critical:

Definition 5.2.1. Let $I \subseteq [c]$ be **cycle critical** with respect to D if $t_D(I) = 2s_D(I) > 0$.

If $I \subseteq [c]$ is cycle critical then in any realisation G there must be a cycle where each edge has a colour in I . Thus another necessary condition is that there are no two disjoint cycle critical (colour) sets. We shall see that these conditions are also sufficient.

Theorem 5.2.2. The demand matrix $D = (d_{v,q} : v \in [n], q \in [c])$ is the colour degree matrix of a graph with at most one cycle if and only if

- (a) for each $I \subseteq [c]$, the sequence $(d_{1,I}, \dots, d_{n,I})$ is the degree sequence of a graph with at most one cycle, and
- (b) there are no two disjoint cycle critical sets.

Let us restate Theorem 5.2.2, much as we restated Theorem 4.2.1 for forests. Theorems 5.2.2 and 5.2.3 together form the main result of this chapter.

Theorem 5.2.3. The demand matrix $D = (d_{v,q} : v \in [n], q \in [c])$ is the colour degree matrix of a graph with at most one cycle if and only if

- (U1) $t(q)$ is even for each $q \in [c]$,
- (U2) $t(I) \leq 2s(I)$ for each $I \subseteq [c]$,
- (U3) there are at least 3 vertices v with $d_{v,I} \geq 2$ for each cycle critical set $I \subseteq [c]$, and
- (U4) there are no two disjoint cycle critical sets.

We will refer to conditions (U1)-(U4) as the *unicycle conditions*.

5.2.1 Properties of tree and cycle critical sets

First we will consider tree and cycle critical sets of colours and their properties.

Lemma 5.2.4. *Suppose that for all non-empty $I \subseteq [c]$, $t(I) \leq 2s(I) - 2$. Let $I, J \subseteq [c]$ be tree critical sets such that $s(I \cap J) > 0$. Then $I \cup J$ and $I \cap J$ are tree critical sets too.*

Proof. It is easy to check that $s(A)$ is submodular for $A, B \subseteq [c]$:

$$s(A) + s(B) \geq s(A \cap B) + s(A \cup B), \quad (5.1)$$

and

$$t(A) + t(B) = t(A \cap B) + t(A \cup B). \quad (5.2)$$

Now set $\forall A \subseteq [c]$ with $s(A) > 0$,

$$f(A) = 2s(A) - t(A) - 2. \quad (5.3)$$

So $f(A) \geq 0$ if $A \neq \emptyset$ and $f(A) = 0$ if and only if A is a tree critical set. Then

$$\begin{aligned} 0 &= f(I) + f(J) = 2s(I) + 2s(J) - t(I) - t(J) - 4 \\ &\geq 2s(I \cap J) + 2s(I \cup J) - t(I \cap J) - t(I \cup J) - 4 \\ &= f(I \cap J) + f(I \cup J) \geq 0. \end{aligned}$$

Thus $I \cup J$ and $I \cap J$ are tree critical sets. □

Lemma 5.2.5. *Suppose that for all $I \subseteq [c]$, $t(I) \leq 2s(I)$. Let $I, J \subseteq [c]$ be cycle critical sets. Then $I \cup J$ and $I \cap J$ are either cycle critical sets or empty.*

Proof. As Lemma 5.2.4. □

5.2.2 The unicycle conditions are necessary

Lemma 5.2.6. *If $D = (d_{v,q} : v \in [n], q \in [c])$ is the colour demand matrix of a graph with at most one cycle, then*

(U1) $t(q)$ is even for each $q \in [c]$,

(U2) $t(I) \leq 2s(I)$ for each $I \subseteq [c]$,

(U3) there are at least 3 vertices v with $d_{v,I} \geq 2$ for each cycle critical set $I \subseteq [c]$, and

(U4) there are no two disjoint cycle critical sets.

Proof. Condition (U1) is the handshaking lemma. Condition (U2) follows, since a unicyclic graph on k vertices has degree sum at most $2k$. Condition (U3) excludes

forcible multi-edges and self-loops. Condition (U4) is necessary because if I is a cycle critical set, then in any realisation there must be a cycle with colours from I . Thus for any two disjoint cycle critical sets, there must be at least two cycles. $\square$

5.2.3 Equivalent conditions

Let $D = (d_{v,q} : v \in [n], q \in [c])$ be a demand matrix. Some other necessary conditions for D to be a simple unicyclic graph are:

Definition 5.2.7. For $I \subseteq [c]$, let $\Delta_{j,I}, j \in [s_D(I)]$, denote the j^{th} largest demand $d_{v,I}$ for I .

- (U5) $s(I) \geq 2$ and $\Delta_{1,I} + \Delta_{2,I} \leq s(I) + 1$, for each cycle critical set $I \subseteq [c]$.
- (U6) Let $I_1, I_2, \dots, I_k \subseteq [c]$ be all cycle critical sets in D . If $k \geq 1$, there is a unique minimal cycle critical set, which is $I = \bigcap_{i=1}^k I_i$.
- (U7) Let $\mathcal{J}$ be the set of all cycle critical sets in D . Then $\mathcal{J} = \emptyset$ or $(\mathcal{J}, \cap)$ is a commutative monoid.

We denote the restriction of condition (Ui), $i = 2, 3, 5$, to a given $I \subseteq [c]$ by $(Ui)_I$. For example, for a fixed $I \subseteq [c]$, we denote $t(I) \leq 2s(I)$ by $(U2)_I$ holds.

Lemma 5.2.8. $(U3) \Leftrightarrow (U5)$

Proof. Let I be a cycle critical set.

Suppose there at least three vertices $v \in S(I)$ such that $d_{v,I} \geq 2$, then $s(I) \geq 3$ and

$$2s(I) = t(I) \geq \Delta_{1,I} + \Delta_{2,I} + s(I) - 1,$$

and so $\Delta_{1,I} + \Delta_{2,I} \leq s(I) + 1$

Conversely, suppose $\Delta_{1,I} + \Delta_{2,I} \leq s(I) + 1$ and that there at most two vertices $v \in S(I)$ that have $d_{v,I} \geq 2$, then

$$2s(I) = t(I) = \Delta_{1,I} + \Delta_{2,I} + s(I) - 2,$$

and so $\Delta_{1,I} + \Delta_{2,I} = s(I) + 2 > s(I) + 1$, contradicting the assumption. Thus there are at least three vertices such that $d_{v,I} \geq 2$. $\square$

Lemma 5.2.9. $(U2) \text{ and } (U4) \Leftrightarrow (U2) \text{ and } (U6)$.

Proof. Suppose condition (U2) holds. If there is at most one cycle critical set the statement is easily verified. Suppose $I_1, \dots, I_k$ are all cycle critical sets in D and that $k \geq 2$.

If condition (U4) holds. Without loss of generality let I_1 be a minimal cycle critical set and let $I = \bigcap_{i=1}^k I_i$. By induction on k and Lemma 5.2.5, I is cycle critical and thus $I \neq \emptyset$. By construction $I \subseteq I_1$ and therefore by minimality $I_1 = I$.

Conversely, condition (U6) implies (U4) directly, because the intersection of all cycle critical sets is non-empty. $\square$

Lemma 5.2.10. *(U2) and (U4) $\Leftrightarrow$ (U2) and (U7).*

Proof. Suppose conditions (U2) and (U4) hold and that $\mathcal{J} \neq \emptyset$. Then $\mathcal{J}$ is closed under intersection by (U2) and (U4) and Lemma 5.2.5. Moreover, intersection is associative and commutative. Finally $I = \bigcup_{J \in \mathcal{J}} J$ is a cycle critical set by Lemma 5.2.5 and thus $I \in \mathcal{J}$ and I is the identity element in $\mathcal{J}$.

Now suppose that conditions (U2) and (U7) hold and that $\mathcal{J}$ is non-empty. Then, as $\mathcal{J}$ is closed under intersection and $\emptyset \notin \mathcal{J}$, $I \cap J \neq \emptyset$ for all $I, J \in \mathcal{J}$. $\square$

We will use these conditions interchangeably.

5.2.4 Implications of the unicycle conditions

Throughout this section let $D = (d_{v,q} : v \in [n], q \in [c])$ be a demand matrix. Note that without loss of generality we can assume

$$\begin{aligned} \sum_{v \in [n]} d_{v,q} &> 0, & \forall q \in [c], \text{ and} \\ \sum_{q \in [c]} d_{v,q} &> 0, & \forall v \in [n]. \end{aligned} \tag{5.4}$$

An important relation is the following; for $I \subseteq [c]$ and any partition of I into I_1, I_2 ,

$$s(I) = s(I_1) + s(I_2) - |S(I_1) \cap S(I_2)|. \tag{5.5}$$

Lemma 5.2.11. *Suppose $I \subseteq [c]$ is such that $t(I) \leq 2s(I) - 2$. Then for all $v, w \in S(I)$,*

$$d_{v,I} + d_{w,I} \leq s(I).$$

Proof. For all $v, w \in S(I)$, $d_{v,I} + d_{w,I} + s(I) - 2 \leq 2s(I) - 2$. Rearranging gives the result. $\square$

Lemma 5.2.12. *Let $\mathcal{J}$ be the set of all cycle critical sets in D . Then $(U2) \Rightarrow (\emptyset \cup \mathcal{J}, \cup)$ is a commutative monoid.*

5.2. The unicycle conditions

Proof. If $\mathcal{J} = \emptyset$, the statement is trivially true, so without loss of generality, assume $\mathcal{J} \neq \emptyset$. Condition (U2) and Lemma 5.2.5 imply that $\mathcal{J}$ is closed under union. Moreover, union is associative and commutative. Finally, $\emptyset$ is the identity element. $\square$

Lemma 5.2.13. *For $I \subseteq [c]$*

$$\Delta_{1,I} \leq s(I) - 1$$

.

Proof. If I is cycle critical, condition (U3) gives $\Delta_{2,I} \geq 2$. Then condition (U5) states $\Delta_{1,I} + 2 \leq \Delta_{1,I} + \Delta_{2,I} \leq s(I) + 1$. Rearranging gives the result. If I is not cycle critical and $s(I) > 0$, then condition (U1) implies that $s(I) \geq 2$, and so

$$\Delta_{1,I} + s(I) - 1 \leq t(I) \leq 2s(I) - 2.$$

Rearranging gives the required inequality. $\square$

Proposition 5.2.14. *Suppose that condition (U3) holds for D , $I \subseteq [c]$ is tree critical and $d_{v,I} + d_{w,I} = s(I)$ for some distinct $v, w \in S(I)$. Then for all $J \subseteq I$,*

$$d_{v,J} + d_{w,J} \geq s(J) - |S(J) \cap \{v, w\}|.$$

Proof. Set $S = S(I) \setminus \{v, w\}$. The assumptions give

$$\begin{aligned} 2s(I) - 2 = t(I) &= \sum_{x \in S(I)} d_{x,I} = d_{v,I} + d_{w,I} + d_{S,I} \\ &\geq d_{v,I} + d_{w,I} + s(I) - 2 = 2s(I) - 2. \end{aligned} \tag{5.6}$$

This implies $d_{S,I} = |S| = s(I) - 2$ and thus for every $x \in S$, $d_{x,I} = 1$. Then by condition (U3), no $J \subseteq I$ is cycle critical. Suppose $J \subseteq I$ and $s(J) - (r + 1) \geq d_{v,J} + d_{w,J}$, where $r = |S(J) \cap \{v, w\}|$. Then $J \neq I$ by assumption. Set $K = I \setminus J$. Note that $S(K) \cap S(J) \subseteq \{v, w\}$ and thus $|S(K) \cap S(J)| \leq r$. This gives

$$s(K) = s(I) - s(J) + |S(K) \cap S(J)| \leq s(I) - s(J) + r,$$

and combining this with the assumptions gives

$$d_{v,K} + d_{w,K} = d_{\{v,w\},I} - d_{\{v,w\},J} \geq s(I) - s(J) + (r + 1) \geq s(K) + 1,$$

which is a contradiction with Lemma 5.2.11, because K is not cycle critical. $\square$

Corollary 5.2.15. *Suppose $I \subseteq [c]$ is tree critical and $d_{v,I} + d_{w,I} = s(I)$ for some distinct $v, w \in S(I)$. Then for all $J \subseteq I$ and $S = S(I) \setminus \{v, w\}$,*

$$d_{\{v,w\},J} \geq d_{S,J}.$$

Proof. Let $J \subseteq I$ and set $r = |S(J) \cap \{v, w\}|$. By Proposition 5.2.14, $r = 0$ implies $d_{v,J} + d_{w,J} = 0 \geq s(J) - 0$, i.e. $s(J) = 0$, which contradicts Eq. 5.4. Consequently $r = 0$ does not happen. If $r = 1, 2$, $d_{x,J} \in \{0, 1\}$ for every $x \in S$, because for every $x \in S$, $d_{x,I} = 1$. This implies $d_{S,J} = |S(J) \cap S| = s(J) - r$ and the result follows from Proposition 5.2.14, as

$$d_{\{v,w\},J} = d_{v,J} + d_{w,J} \geq s(J) - |S(J) \cap \{v, w\}| = s(J) - r = d_{S,J}. \quad \square$$

Lemma 5.2.16. *Suppose that condition (U2) holds for D . Then for all distinct $x, y, z \in S(I)$,*

$$d_{x,I} + d_{y,I} + d_{z,I} \leq s(I) + 3. \quad (5.7)$$

Proof. Observe that by condition (U2), we have for all $I \subseteq [c]$ and for all distinct $x, y, z \in S(I)$

$$2s(I) \geq t(I) \geq d_{x,I} + d_{y,I} + d_{z,I} + s(I) - 3.$$

Rearranging gives Eq. 5.7. $\square$

5.3 Directly forced edges

Before we consider realisations of the entire degree sequence, it is worthwhile looking at partial realisations that appear in every realisation. These parts are directly forced by critical subsets of colours with certain demand conditions.

We will consider directly forced edges first and directly forced triangles in the next section. We will show that directly forced edges are forced and can be realised. Next we establish some conditions that imply a directly forced edge, illustrated by Example 3.

Throughout this section let $D = (d_{v,q} : v \in [n], q \in [c])$ be a demand matrix.

Definition 5.3.1. *For distinct $v, w \in [n]$ and $q \in [c]$, vw is a **directly forced edge in colour q** if $t(q) \leq 2s(q) - 2$ and $d_{v,q} + d_{w,q} = s(q)$.*

5.3. Directly forced edges

Set $S = S(q) \setminus \{v, w\}$:

$$\begin{aligned} 2s(q) - 2 \geq t(q) &= \sum_{x \in S(q)} d_{x,q} = d_{v,q} + d_{w,q} + d_{S,q} \\ &\geq d_{v,q} + d_{w,q} + s(q) - 2 = 2s(q) - 2. \end{aligned} \quad (5.8)$$

Thus the condition “ $t(q) \leq 2s(q) - 2$ ” could be replaced by “ q is tree critical” and all $x \in S$ are such that $d_{x,q} = 1$. Moreover observe that $d_{v,q} + d_{w,q} = s(q) \geq 1$ by Eq. 5.4. Lemma 5.2.13 implies $d_{v,q}, d_{w,q} \leq s(q) - 1$, thus $d_{v,q}, d_{w,q} \geq 1$.

Example 3 (Directly forced edge). *Below is a demand matrix of a directly forced edge with a unique realisation.*

$D =$

	q	r
v	3	0
w	1	2
x_1	1	0
x_2	1	0
x_3	0	1
x_4	0	1

Let $I = \{q, r\}$, $B = \{v, w\}$ and $S = \{x_1, x_2, x_3, x_4\}$. Then

$$\begin{aligned} s(I) &= 6 & t(I) &= 10 = 2s(I) - 2 \\ s(q) &= 4 & t(q) &= 6 = 2s(q) - 2 \\ s(r) &= 3 & t(r) &= 4 = 2s(r) - 2 \\ d_{B,q} &= 4 = s(q) & d_{S,q} &= 2 = d_{B,q} - 2 \\ d_{B,r} &= 2 = s(r) - 1 & d_{S,q} &= 2 = d_{B,q} \\ d_{B,I} &= 6 = s(I) & d_{S,I} &= 4 = d_{B,I} - 2, \end{aligned}$$

as expected. Note that in fact every edge in this realisation is directly forced.

5.3.1 Directly forced edges are forced and the demands for colour q can be realised

Proposition 5.3.2. *Let vw be a directly forced edge in colour q , then vw is an edge with colour q in every realisation of D .*

Proof. Eq. 5.8 implies that $d_{x,q} = 1$ for all $x \in S = S(q) \setminus \{v, w\}$. Suppose there exists a realisation in which v, w are not adjacent in colour q , then v and w have $s(q)$ neighbours in S in colour q , but $|S| = s(q) - 2$. Therefore v and w must have at least 1 common neighbour in q , which contradicts $d_{x,q} = 1$ for all $x \in S(q)$. Thus vw are adjacent in colour q in every realisation of D . $\square$

It is easy to see that the demands for colour q can be realised, by connecting v to w in q , since $d_{v,q}, d_{w,q} \geq 1$, and all x in S to one of v, w in colour q , as their demand requires.

5.3.2 Conditions that imply a directly forced edge

Lemma 5.3.3. *Let D satisfy the unicycle conditions. Suppose that $I \subseteq [c]$ is such that $t(I) \leq 2s(I) - 2$, and v and w are distinct and satisfy $d_{v,I} + d_{w,I} = s(I)$. Then there exists a unique colour $q \in I$, such that vw is a directly forced edge in colour q , and each $x \in S(q) \setminus \{v, w\}$ satisfies $d_{x,q} = 1$.*

Proof. By assumption $t(I) \leq 2s(I) - 2$. In fact we have equality here, since

$$\begin{aligned} 2s(I) - 2 &\geq t(I) = d_{v,I} + d_{w,I} + \sum_{x \in S(I) \setminus \{v, w\}} d_{x,I} \\ &\geq d_{v,I} + d_{w,I} + s(I) - 2 = 2s(I) - 2. \end{aligned}$$

Moreover, v and w must be in $S(I)$, and for every $x \in S = S(I) \setminus \{v, w\}$, we must have $d_{x,I} = 1$ and also $d_{x,q} \in \{0, 1\}$ for each $q \in I$. Thus by condition (U3), I has no cycle critical subsets.

Suppose that $d_{v,q} + d_{w,q} < s(q)$ for all $q \in I$. If $v, w \in S(q)$, condition (U1) implies that

$$t(q) = d_{v,q} + d_{w,q} + d_{S,q} = d_{v,q} + d_{w,q} + s(q) - 2 \equiv 0 \pmod{2}.$$

Therefore $d_{v,q} + d_{w,q} \neq s(q) - 1$. Thus $d_{v,q} + d_{w,q} \leq s(q) - 2$ if $v, w \in S(q)$ and $d_{v,q} + d_{w,q} \leq s(q) - 1$ otherwise; and so $d_{v,q} + d_{w,q} \leq |S(q) \setminus \{v, w\}|$. Hence

$$s(I) = \sum_{q \in I} (d_{v,q} + d_{w,q}) \leq \sum_{q \in I} |S(q) \setminus \{v, w\}| = |S| = s(I) - 2, \quad (5.9)$$

because each $x \in S$ appears in $S(q)$ for exactly one $q \in I$. This contradiction shows that there exists $q \in I$ such that $d_{v,q} + d_{w,q} \geq s(q)$. By Lemma 5.2.13, $d_{v,q}, d_{w,q} \leq s(q) - 1$, thus $d_{v,q}, d_{w,q} \geq 1$, and $d_{v,q} + d_{w,q} \geq s(q) = |S(q) \setminus \{v, w\}| + 2$. Thus by

Eq. 5.9, q is unique. Now

$$t(q) \geq d_{v,q} + d_{w,q} + s(q) - 2 \geq 2s(q) - 2 \geq t(q) \quad (\text{since } \{q\} \text{ is not critical})$$

and it follows that $t(q) = 2s(q) - 2$ and $d_{v,q} + d_{w,q} = s(q)$, as required. We already saw that for all $x \in S(I) \setminus \{v, w\}$ and $r \in I$, $d_{x,r} \in \{0, 1\}$, so in particular $d_{x,q} = 1$ for all $x \in S(q) \setminus \{v, w\}$. $\square$

Two important examples of a directly forced edge vw in colour q are

1. $S(q) = \{v, w\}$
2. $q \in I \subseteq [c]$, I tree cycle critical, $d_{v,I} = s(I) - 1$ and $d_{w,q} = 1$.

5.4 Directly forced triangles

In this section we will consider directly forced triangles: triangles that must exist in every (simple) realisation, due to certain constraints on the demands. We will first show some properties of directly forced triangles, then we will show that a directly forced triangle is forced and can be realised. Finally we find some conditions that imply a directly forced triangle. See Example 4 for an illustration of a directly forced triangle.

We will suppose throughout this section that the unicycle conditions hold for demand matrix D .

Definition 5.4.1. For distinct $u, v, w \in [n]$, $J \subseteq [c]$, uvw is a **directly forced triangle with colours J** if $d_{u,J} + d_{v,J} + d_{w,J} = s(J) + 3$ and J is minimal with respect to this property.

Example 4 (Directly forced triangle). Below is the demand matrix of a directly forced triangle with a realisation:

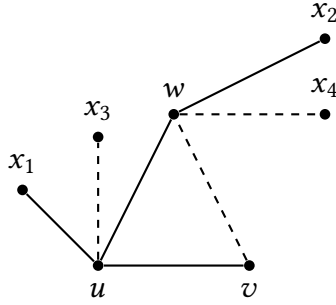
$D =$

	q	r
u	3	1
v	1	1
w	2	2
x_1	1	0
x_2	1	0
x_3	0	1
x_4	0	1

Let $I = \{q, r\}$, $B = \{u, v, w\}$ and $S = \{x_1, x_2, x_3, x_4\}$. Then

$$\begin{aligned} s(I) &= 7 & t(I) &= 14 = 2s(I) \\ s(q) &= 5 & t(q) &= 8 = 2s(q) - 2 \\ s(r) &= 5 & t(r) &= 6 = 2s(r) - 4 \\ d_{B,q} &= 6 = s(q) + 1 & d_{S,q} &= 2 = d_{B,q} - 4 \\ d_{B,r} &= 4 = s(r) - 1 & d_{S,r} &= 2 = d_{B,r} - 2 \\ d_{B,I} &= 10 = s(I) + 3 & d_{S,I} &= 4 = d_{B,I} - 6, \end{aligned}$$

as expected. Note that the realisation in above is not unique. An alternative realisation is shown below:



5.4.1 Properties of directly forced triangles

Let uvw be a directly forced triangle with colours J and let $B = \{u, v, w\}$ and $S = S(J) \setminus B$.

Lemma 5.4.2. *Let B and J be as above. Then J is cycle critical, $d_{x,J} = 1$ for all $x \in S$ and for all $I \subseteq J$, $d_{S,I} = |S \cap S(I)| = s(I) - |S(I) \cap B|$.*

Proof. We note that

$$\begin{aligned} 2s(J) &\geq t(J) = \sum_{x \in S(J)} d_{x,J} = d_{u,J} + d_{v,J} + d_{w,J} + \sum_{x \in S} d_{x,J} \\ &\geq d_{u,J} + d_{v,J} + d_{w,J} + s(J) - 3 = 2s(J). \end{aligned} \tag{5.10}$$

This gives equality in both cases, and so J is cycle critical and $\sum_{x \in S} d_{x,J} = s(J) - 3 = |S|$, implying that $d_{x,J} = 1$ for all $x \in S$. As $d_{x,J} = 1$ for all $x \in S$, $d_{x,I} \leq 1$ for all $I \subseteq J$. Thus

$$\begin{aligned} d_{S,I} &= |\{x \in S : d_{x,I} = 1\}| = |\{x \in S : d_{x,I} \neq 0\}| \\ &= |S \cap S(I)| = |S(I) \setminus B| = s(I) - |S(I) \cap B|. \end{aligned} \quad \square$$

Lemma 5.4.3.

$$d_{B,J} = d_{S,J} + 6.$$

Proof. As J is cycle critical and $d_{x,J} = 1$ for all $x \in S$ by Lemma 5.4.2, $d_{u,J}, d_{v,J}, d_{w,J} \geq 2$ by condition (U3) and so $|S| = s(J) - 3 = d_{S,J}$. Lemma 5.4.2 gives

$$d_{B,J} = s(J) + 3 = d_{S,J} + 6. \quad \square$$

Lemma 5.4.4. *No $I \subsetneq J$ is cycle critical.*

Proof. Suppose there exist a cycle critical $I \subsetneq J$. By condition (U3), $u, v, w \in S(I)$, as $d_{x,J} = 1$ for all $x \in S$ by Lemma 5.4.2. Then

$$2s(I) = d_{B,I} + d_{S,I} = d_{B,I} + |S \cap S(I)| = d_{B,I} + s(I) - 3.$$

Rearranging gives $d_{B,I} = s(I) + 3$, contradicting minimality of J . $\square$

Lemma 5.4.5. *For each $\emptyset \neq I \subseteq J$, $d_{B,I} \geq d_{S,I} + 2$.*

Proof. Suppose for a contradiction that $\emptyset \neq I \subseteq J$ satisfies $d_{B,I} \leq d_{S,I}$. Then $I \neq J$ by Lemma 5.4.3. Set $K = J \setminus I$ (so $\emptyset \neq K \subsetneq J$). As $d_{S,I} = |S \cap S(I)| = s(I) - |B \cap S(I)|$,

$$\begin{aligned} d_{B,K} &= d_{B,J} - d_{B,I} \geq s(J) + 3 - d_{S,I} \\ &= s(J) - (s(I) - |B \cap S(I)|) + 3 \\ &= |S \cap S(K)| + |B \setminus S(I)| + |B \cap S(I)| + 3 \\ &= |B| + |S \cap S(K)| + 3 \geq s(K) + 3. \end{aligned}$$

Thus by Lemma 5.2.16, $d_{B,K} = s(K) + 3$, contradicting minimality of J . Hence for each $\emptyset \neq I \subseteq J$, we have $d_{B,I} > d_{S,I}$ and since their sum is even we have $d_{B,I} \geq d_{S,I} + 2$. $\square$

Corollary 5.4.6.

$$1 \leq |J| \leq 3.$$

Proof. J is cycle critical, so non-empty. By Lemma 5.4.3 and 5.4.5,

$$6 = d_{B,J} - d_{S,J} = \sum_{q \in J} (d_{B,q} - d_{S,q}) \geq \sum_{q \in J} 2 = 2|J|.$$

Rearranging gives $|J| \leq 3$ as required. $\square$

Corollary 5.4.6 and Lemma 5.4.2 imply that there are three different cases for a directly forced triangle:

1. $J = \{q\}$ and $d_{B,q} = d_{S,q} + 6$: we require a uni-coloured triangle,
2. $J = \{q, r\}$ and $d_{B,q} = d_{S,q} + 4$ and $d_{B,r} = d_{S,r} + 2$: we require a triangle with two sides in colour q and one side in colour r ,
3. $J = \{q, r, s\}$ and $d_{B,i} = d_{S,i} + 2$ for all $i \in J$: we require a triangle with each side in a different colour.

5.4.2 A directly forced triangle is forced

Proposition 5.4.7. *Suppose that conditions (U1)-(U4) hold for D and that uvw is a directly forced triangle with colours J . Then in every realisation of D uvw is a triangle and J is the set of colours appearing on its edges.*

Proof. Set $S = S(J) \setminus \{u, v, w\}$ and $d_{u,J} \geq d_{v,J} \geq d_{w,J}$. Moreover $d_{u,J} \geq d_{v,J} \geq d_{w,J} \geq 2$, as $d_{u,J} + d_{v,J} \leq s(J) + 1$ by condition (U5). Suppose a pair in $\{u, v, w\}$ is non-adjacent in J in some realisation of D . Then $\{u, v, w\}$ have at least $s(J) - 1$ neighbours in S in colours J , but $|S| = s(J) - 3$ and therefore there must be at least one vertex $x \in S$ that is incident with two edges in a colour from J . This contradicts Lemma 5.4.3, which states $d_{x,J} = 1$ for all $x \in S$. Thus uvw is a triangle with colours J in every realisation in D . $\square$

5.4.3 The demand for the colours of a directly forced triangle can be realised

Lemma 5.4.8. *Let D satisfy the unicycle conditions. Suppose that uvw is a directly forced triangle with colours J , and let $D' = (d_{t,q} : t \in S(J), q \in J)$. Then D' is the colour degree matrix of a unicyclic graph.*

Proof. We have $d_{u,J}, d_{v,J}, d_{w,J} \geq 2$ (since J is cycle critical). Set $B = \{u, v, w\}$ and $S = S(J) \setminus B$ and by Corollary 5.4.3, $|J| \leq 3$. If we can form a triangle in u, v, w to absorb excess demands $d_{B,q} - d_{S,q}$ (each at least 2 and summing to 6), then connecting $x \in S$ to u, v or w appropriately gives the required realisation.

There are three cases:

1. $|J| = 1$, say $J = \{q\}$, and $d_{B,q} = d_{S,q} + 6$. Then the required (uni-coloured) triangle can be formed, as $d_{u,q}, d_{v,q}, d_{w,q} \geq 2$.
2. $|J| = 2$, say $J = \{q, r\}$, and $d_{B,q} = d_{S,q} + 4$ and $d_{B,r} = d_{S,r} + 2$. We are looking for a two-coloured triangle with two edges in q and one in r . If we have $d_{x,q} \geq 2$ and $d_{y,q}, d_{z,q} \geq 1$, and $d_{y,r}, d_{z,r} \geq 1$, for distinct $x, y, z \in B$, the required

triangle can be formed. Suppose there is $x \in B$, such that $d_{x,q} = 0$. Then setting $B' = B \setminus x$ gives $d_{B',q} = d_{S,q} + 4 \geq s(q) + 2$, contradicting condition Lemma 5.2.11 or (U5). Thus $d_{x,q} \geq 1$ for all $x \in B$. Suppose now that $d_{x,r} = 0$ for some $x \in B$, and set $B' = B \setminus x = \{y, z\}$. Then

$$\begin{aligned} d_{y,r} + d_{z,r} &= d_{B,r} \geq d_{S,r} + 2 \quad \text{by Lemma 5.4.3,} \\ &= s(r) - |S(r) \cap B| + 2 \quad \text{by Lemma 5.4.2.} \\ &\geq s(r) - 2 + 2 = s(r). \end{aligned}$$

But $d_{y,r}, d_{z,r} < s(r)$ by Lemma 5.2.13, so $d_{y,r}, d_{z,r} \geq 1$. Thus we can form the required triangle, as any $x \in B$ for which $d_{x,r} = 0$, must have $d_{x,q} \geq 2$, as $d_{x,J} \geq 2$, for each $x \in B$.

3. $|J| = 3$, say $J = \{q, r, s\}$ and $d_{B,q} = d_{S,q} + 2$, $d_{B,r} = d_{S,r} + 2$ and $d_{B,s} = d_{S,s} + 2$. We are looking for a three-coloured triangle with colours q, r , and s . We need for all $i \in \{q, r, s\}$ that there exists a distinct pair $x, y \in B$ such that $d_{x,i}, d_{y,i} \geq 1$. If $d_{x,i} = d_{x,j} = 0$, for some distinct $i, j \in J$, then letting $B \setminus x = \{y, z\}$, we have $d_{y,i} + d_{z,i} \geq s(i)$ and $d_{y,i}, d_{z,i} \geq 1$ by Lemma 5.2.13. Similarly $d_{y,j} + d_{z,j} \geq s(j)$ and $y, z \in S(j)$. Then $\{i, j\}$ is cycle critical:

$$\begin{aligned} t(\{i, j\}) &= d_{y,i} + d_{z,i} + d_{y,j} + d_{z,j} + d_{S,\{i,j\}} \\ &\geq s(i) + s(j) + d_{S,\{i,j\}} \\ &= s(i) + s(j) + s(\{i, j\}) - 2 \\ &= s(i) + s(j) - |S(i) \cap S(j)| + s(\{i, j\}) = 2s(\{i, j\}). \end{aligned}$$

This contradicts condition (U3), as there are at most 2 vertices $x \in S(\{i, j\})$ such that $d_{x,\{i,j\}} \geq 2$. So every $x \in B$ misses at most one colour. Moreover these colours are distinct, for if x is the only vertex in B with demand for colour i , then $d_{x,i} = d_{B,i} = d_{S,i} + 2 = s(i) + 1$, contradicting Lemma 5.2.13. Consider the bipartite graph with parts B and J (each of size 3) and an edge xj whenever $d_{x,j} \geq 1$. The edges missing form a matching, so the edges present must contain a 6-cycle, which specifies a coloured triangle as required. $\square$

5.4.4 Conditions that imply a directly forced triangle

Lemma 5.4.9. *Suppose that the unicycle conditions hold for D , that $I \subseteq [c]$ is cycle critical and $u, v, w \in S(I)$ are distinct and satisfy $d_{u,I} + d_{v,I} = s(I) + 1$ and $d_{w,I} = \Delta_{3,I}$.*

Then there exists $J \subseteq I$, such that uvw is a directly forced triangle with colours J .

Proof. Let $B = \{u, v, w\}$ and $S = S(I) \setminus B$. Note that for all $x \in S$, $d_{x,I} = 1$ and that $d_{w,I} = 2$. Moreover, for all $J \subseteq I$,

$$s(J) \leq |S \cap S(J)| + 3 = d_{S,J} + 3,$$

with equality for I . Let $H = \{q \in I : d_{B,q} \leq d_{S,q}\}$. Suppose $q \in H$ is such that $d_{B,q} \leq d_{S,q} - 1$, then by Lemma 5.2.16,

$$\begin{aligned} s(I \setminus q) + 3 &\geq d_{B,I \setminus q} \geq d_{B,I} - (d_{S,q} - 1) = s(I) + 3 - (d_{S,q} - 1) \\ &= d_{S,I} + 6 - (d_{S,q} - 1) = d_{S,I \setminus q} + 7 \geq s(I \setminus q) + 4. \end{aligned}$$

Thus all $q \in H$ are such that $d_{B,q} = d_{S,q}$. Let $J = I \setminus H$, then by Lemma 5.2.16,

$$\begin{aligned} s(J) + 3 &\geq d_{B,J} = d_{B,I} - \sum_{q \in H} d_{B,q} = d_{S,I} + 6 - \sum_{q \in H} d_{S,q} \\ &= d_{S,J} + 6 \geq s(J) + 3. \end{aligned} \quad \square$$

Corollary 5.4.10. *Suppose that the unicycle conditions hold for D , that $I \subseteq [c]$ is cycle critical and $u, v, w \in S(I)$ are distinct and satisfy $d_{u,I} = s(I) - 1$, $d_{v,I} = \Delta_{2,I}$ and $d_{w,I} = \Delta_{3,I}$. Then there exists $J \subseteq I$, such that uvw is a directly forced triangle with colours J .*

Proof. We show that the assumptions imply those of Lemma 5.4.9. If $d_{u,I} = s(I) - 1$, then $\Delta_{2,I} \leq 2$ by condition (U5) and I cycle critical gives $\Delta_{2,I} \geq \Delta_{3,I} \geq 2$ by condition (U3). Thus $d_{u,I} + d_{v,I} = s(I) - 1 + 2 = s(I) + 1$ and $d_{w,I} = \Delta_{3,I}$. $\square$

5.5 The unicycle conditions are sufficient

5.5.1 Sufficiency when each total degree is two

Lemma 5.5.1. *Let $D = (d_{v,q} : v \in [n], q \in [c])$ satisfy the unicycle conditions, and suppose that $d_{v,[c]} = 2, \forall v \in [n]$. Then D can be realised by a single cycle through all vertices.*

Proof. Let each demand for a colour at a vertex $v \in [n]$ correspond to a half-edge of this colour. We will try to find a realisation by pairing the half-edges of the same colour. There is an even number $t(q)$ of half-edges of each colour q by condition

5.5. The unicycle conditions are sufficient

(U1). Arbitrarily pairing half-edges gives a (not necessarily simple) realisation of D , which consists of a disjoint union of cycles. Consider a realisation G with a minimal number of cycles and suppose there are at least two cycles. If two cycles have a colour in common on edges x_1y_1, x_2y_2 say, then switching these edges to x_1x_2, y_1y_2 gives a realisation of D with fewer cycles. If a cycle C has no common colours with $G \setminus C$, then the colours on C and the remaining colours are disjoint cycle critical sets, contradicting condition (U4). Thus G is a single (simple) cycle through all vertices. $\square$

This proof suggests a $O(n^2)$ time algorithm. Cycles can be formed in $O(n^2)$ time and each of the $O(n)$ cycles can be merged with another cycle in $O(n)$ time.

5.5.2 Sufficiency in general

Theorem 5.2.3. *The demand matrix $D = (d_{v,q} : v \in [n], q \in [c])$ is the colour degree matrix of a graph with at most one cycle if and only if*

- (U1) $t(q)$ is even for each $q \in [c]$,
- (U2) $t(I) \leq 2s(I)$ for each $I \subseteq [c]$,
- (U3) there are at least 3 vertices v with $d_{v,I} \geq 2$ for each cycle critical set $I \subseteq [c]$, and
- (U4) there are no two disjoint cycle critical sets.

5.5.2.1 Proof outline

We will consider a counterexample $D = (d_{v,q} : v \in [n], q \in [c])$, which is minimal with respect to n and then with respect to c . In a series of Claims we will show more and more properties of D , and eventually obtain a contradiction.

The main steps are as follows. First we show that the full colour set $[c]$ is the unique cycle critical set. Lemma 5.4.8 then implies that there is no directly forced triangle. By Lemma 5.5.1 there is a vertex $v_1 \in [n]$ with total demand one, say $d_{v_1,1} = 1$. We fix v_1 : the rest of the proof involves looking for a suitable partner $w \in S(1)$ for v_1 , so that we can make v_1w an edge with colour 1 in a unicyclic realisation of D . We consider the demand matrix $\tilde{D} = \tilde{D}(w)$, which is D with v_1 removed and $d_{w,1}$ reduced by one. If the unicycle conditions hold in $\tilde{D}$, then a unicyclic realisation of $\tilde{D}$ exists by minimality of D . By appending v_1 to w with colour 1, this realisation will extend to a realisation of D , contradicting our choice of D .

When we check the unicycle conditions for $\tilde{D}$, we see quickly that conditions (U1) and (U4) must hold in $\tilde{D}$. Further we see that, if w or I satisfy some extra conditions then $(U2)_I$ and $(U3)_I$ hold in $\tilde{D}$. There are some sets $I \subsetneq [c]$, that we shall call dangerous, for which the unicycle conditions will not hold in $\tilde{D}$, unless w is picked such that $\tilde{d}_{w,I} > 1$. We see that $s(1) = 4$, and for each dangerous set I , we consider the set $W(I) = S(1) \cap S(I \setminus 1)$ of ‘good’ vertices. We finish the proof by showing that for any two dangerous sets I and J , we have $W(I) = W(J)$ (using the fact there is no directly forced triangle), and picking w from this set yields the desired contradiction.

5.5.2.2 Proof of Theorem 5.2.3

Proof of Theorem 5.2.3. Suppose the conditions are not sufficient and consider a counterexample $D = (d_{v,q} : v \in [n], q \in [c])$ with n minimal and then c minimal, for which conditions (U1)-(U4) hold but there is no realisation with at most one cycle.

Claim 1. $[c]$ is the unique cycle critical set.

Proof. There must exist a cycle critical set $I \subseteq [c]$, or D would be the degree sequence of a forest by Theorem 4.2.3. Suppose $I \subsetneq [c]$ is a cycle critical set. We will show that a unicyclic realisation can be found.

Set $I^c = [c] \setminus I$. By minimality with respect to c , $t(I^c) \geq 2$, $s(I^c) \geq 2$; and $s(I) \leq n - 1$, since if $s(I) = n$

$$2n = t([c]) = t(I) + t(I^c) \geq 2s(I) + s(I^c) \geq 2s(I) + 2 = 2n + 2.$$

Let $D_1 = (d_{v,q} : v \in S(I), q \in I)$ and let a be a new vertex with $d_{a,q} = \sum_{v \in S(I)} d_{v,q}$, for each $q \in I^c$. Let $D_2 = (d_{v,q} : v \in ([n] \setminus S(I)) \cup \{a\}, q \in I^c)$.

By assumption conditions (U1) - (U4) hold for D_1 , so by minimality of D there exists a unicyclic realisation G_1 of D_1 . For all $J \subseteq I^c$, let $s_2(J) = s_{D_2}(J)$ and $t_2(J) = t_{D_2}(J)$. Observe that $t(J) = t_2(J)$. If for all $J \subseteq I^c$, then $t_2(J) \leq 2s_2(J) - 2$, a forest realisation T_2 of D_2 exists by Theorem 4.2.3. A unicyclic realisation of D can then be found by replacing a by G_1 .

Let $J \subseteq I^c$. It remains to show that J is not cycle critical for D_2 . If $S(J) \cap S(I) = \emptyset$, then $t(J) = t_2(J)$ and $s(J) = s_2(J)$. By condition (U4), J is not cycle critical for D , because it is disjoint from I , and so it is not cycle critical for D_2 . If $S = S(J) \cap S(I) \neq \emptyset$, then $s_2(J) = s(J) - |S| + 1$. Suppose $t_2(J) \geq 2s_2(J)$, then since $s(I \cup J) = s(I) + s(J) - |S|$,

5.5. The unicycle conditions are sufficient

we have

$$t(I \cup J) = t(I) + t_2(J) \geq 2s(I) + 2s_2(J) = 2s(I) + 2s(J) - 2|S| + 2 = 2s(I \cup J) + 2.$$

Thus $t_2(J) \leq 2s_2(J) - 2$, as required. $\square$

Claim 2. *There is no directly forced triangle.*

Proof. By condition (U3), $n \geq 3$. There can be no directly forced triangle in $I \subsetneq [c]$, as $[c]$ is the only cycle critical set by Claim 1 and by Lemma 5.4.8 a directly forced triangle in $[c]$ can be realised. $\square$

Claim 3. *There is a leaf, that is, a vertex $v \in [n]$ with $d_{v,[c]} = 1$.*

Proof. If the minimal total degree is 2, then condition (U2) implies all total degrees are exactly 2 and D can be realised by a single cycle through all vertices by Lemma 5.5.1. $\square$

Fix a leaf v_1 . We may suppose without loss of generality that $d_{v_1,1} = 1$ and $d_{v_1,q} = 0$ for all $q \neq 1$. By condition (U1), $S^* = S(1) \setminus \{v_1\} \neq \emptyset$. We aim to find a suitable partner w for v_1 , such that we can make v_1w an edge with colour 1 in a unicyclic realisation of D .

Given $w \in S^*$, let $\tilde{D} = \tilde{D}(w)$ be obtained from D by deleting v_1 and decreasing $d_{w,1}$ by 1. Note that $\tilde{D}$ is a demand matrix on $\tilde{n} = n - 1$ vertices, so if the unicycle conditions hold for $\tilde{D}$, then a unicyclic realisation $\tilde{G}$ of $\tilde{D}$ can be found by minimality of D . This can be extended to a realisation of D by appending v_1 to w with colour 1. We start by showing that some of the unicycle conditions hold, regardless of the choice of w .

Claim 4. *Let $w \in S^*$. Then (U1) and (U4) hold in $\tilde{D}$.*

Proof. It is clear that (U1) holds. If $1 \notin I$ then $\tilde{t}(I) = t(I)$ and $\tilde{s}(I) = s(I)$, so if I is not cycle critical in D then I is not cycle critical in $\tilde{D}$. Thus any cycle critical set in $\tilde{D}$ must contain 1, hence (U4) holds. $\square$

Claim 5. *If $1 \notin I$, then $(U2)_I$ and $(U3)_I$ hold.*

Proof. If $1 \notin I$, then (as noted above) $\tilde{t}(I) = t(I)$ and $\tilde{s}(I) = s(I)$. $\square$

Chapter 5. Colour degree matrices of graphs with at most one cycle

Therefore by Claim 4 and 5 it is enough to show that, for a suitable choice of $w \neq v_1$ in $S(1)$, for all $I \subseteq [c]$ with $1 \in I$,

$$(U2)_I \text{ and } (U3)_I \text{ hold in } \tilde{D}. \quad (5.11)$$

We will first show that Eq. 5.11 holds for some particular $I \subseteq [c]$.

Claim 6. *Let $w \in S^*$ and $I \subseteq [c]$ be such that $1 \in I$, and suppose that $t(I) < 2s(I) - 2$. Then (5.11) holds.*

Proof. Since $t(I) \leq 2s(I) - 4$ by (U1), and $\tilde{s}(I) \geq s(I) - 2$,

$$\tilde{t}(I) = t(I) - 2 \leq 2s(I) - 4 - 2 \leq 2\tilde{s}(I) - 2.$$

Thus I is not cycle critical for $\tilde{D}$, and (5.11) holds. $\square$

Claim 7. *Let $w \in S^*$, let $I \subseteq [c]$ be such that $1 \in I$, and suppose that $w \in \tilde{S}(I)$. Then (5.11) holds.*

Proof. Consider first the case $I \subsetneq [c]$. Then I is not cycle critical (since $[c]$ is the unique cycle critical set), and $\tilde{s}(I) = s(I) - 1$. Hence

$$\tilde{t}(I) = t(I) - 2 \leq 2s(I) - 2 - 2 = 2\tilde{s}(I) - 2.$$

Thus I is not cycle critical for $\tilde{D}$, and (5.11) holds.

Now consider the case $I = [c]$. Note that $s([c]) = n$ and $\tilde{s}([c]) = \tilde{n}$. Then

$$\tilde{t}([c]) = t([c]) - 2 = 2n - 2 = 2\tilde{n}.$$

Also, $\Delta_{1,[c]} + \Delta_{2,[c]} + \Delta_{3,[c]} \leq n + 2$, because there is no directly forced triangle by Claim 2, and $\Delta_{3,[c]} \geq 2$ by (U3). Therefore

$$\tilde{\Delta}_{1,[c]} + \tilde{\Delta}_{2,[c]} \leq \Delta_{1,[c]} + \Delta_{2,[c]} \leq n = \tilde{n} + 1.$$

Thus $(U5)_{[c]}$ holds, and so also (5.11) holds. $\square$

In the next three claims we will show some properties of vertices in $S(1)$. We will need this at the end of the proof.

Claim 8. $d_{x,1} = 1$ for each $x \in S(1)$.

Proof. Suppose there is $w \in S(1)$ with $d_{w,1} \geq 2$. Then $w \in \tilde{S}(I)$ for each $I \subseteq [c]$ that contains 1, so the unicycle conditions hold for $\tilde{D}$ by Claims 4, 5 and 7. $\square$

5.5. The unicycle conditions are sufficient

Claim 9. $S(1) \cap S([c] \setminus 1) \neq \emptyset$.

Proof. Suppose $S(1) \cap S([c] \setminus 1) = \emptyset$, then since $t(1) = s(1) \geq 2$ by Claim 8 and since $[c] \setminus 1$ is not cycle critical,

$$2n = t([c]) = t(1) + t([c] \setminus 1) \leq s(1) + 2s([c] \setminus 1) - 2 = 2n - s(1) - 2 \leq 2n - 4,$$

which is a contradiction. Thus $S(1) \cap S([c] \setminus 1) \neq \emptyset$. $\square$

Claim 10. $s(1) \geq 4$.

Proof. Note that by Claim 8, $t(1) = s(1)$, and condition (U1) implies $s(1)$ is even. Suppose $s(1) = 2$. Then

$$\begin{aligned} t(1) &= 2, \text{ and} \\ s([c] \setminus \{1\}) &\leq s([c]) - 1 = n - 1, \end{aligned}$$

as $v_1 \notin s([c] \setminus \{1\})$; and since $[c] \setminus \{1\}$ is not cycle critical,

$$2s([c] \setminus \{1\}) > t([c] \setminus \{1\}) = t([c]) - 2 = 2n - 2 \geq 2s([c] \setminus \{1\}),$$

which is a contradiction. Thus $s(1) \geq 4$. $\square$

Definition 5.5.2. We say that the set $I \subsetneq [c]$ is **dangerous** if

$$\begin{aligned} 1 &\in I, \text{ and} \\ \Delta_{1,I} + \Delta_{2,I} &= s(I). \end{aligned}$$

We next show that if $I \subseteq [c]$ is not dangerous, then Eq. 5.11 holds and conclude that there must be a dangerous set. Recall that $S^* = S(1) \setminus v_1$.

Claim 11. Let $w \in S^*$, let $1 \in I \subsetneq [c]$, and suppose that I is not dangerous. Then (5.11) holds. So there must be a dangerous set.

Proof. By Claims 6 and 7 we may assume that $t(I) = 2s(I) - 2$ and $w \notin \tilde{S}(I)$. Thus

$$\tilde{t}(I) = t(I) - 2 = 2s(I) - 4 = 2\tilde{s}(I),$$

and so (U2)_I holds in $\tilde{D}$. We have $(S(1) \setminus \{w, v_1\}) \subseteq \tilde{S}(I)$, so $\tilde{s}(I) \geq 4 - 2 = 2$ by Claim 10. Further, $1 \in I \subsetneq [c]$ but I is not dangerous, so

$$\tilde{\Delta}_{1,I} + \tilde{\Delta}_{2,I} \leq \Delta_{1,I} + \Delta_{2,I} \leq s(I) - 1 = \tilde{s}(I) + 1.$$

Thus $(U5)_I$ holds, and so also (5.11) holds.

Suppose there is no dangerous set. By Claim 9 we can choose $w \in S(1) \cap S([c] \setminus \{1\})$, and then $(U2)_I$ and $(U3)_I$ hold in $\tilde{D}$ for $I = [c]$ by Claim 7, and for each $I \subsetneq [c]$ which is not dangerous by the argument above. Finally $(U1)$ and $(U4)$ hold by Claim 4, contradicting our choice of D . We conclude that there must be a dangerous set. $\square$

Definition 5.5.3. For each dangerous $I \subseteq [c]$, let $W(I) = S(1) \cap S(I \setminus 1)$ be the set of *good vertices* for I .

If $I \subseteq [c]$ is a dangerous set, Eq. 5.11 holds if $w \in \tilde{S}(I)$ by Claim 7, and $w \in \tilde{S}(I)$ only if $w \in W(I)$, because $d_{x,1} = 1$ for all $x \in S(1)$ by Claim 8. The next two claims will show that we can pick w such that it is in $W(I)$ for all dangerous sets $I \subseteq [c]$.

Claim 12. $s(1) = 4$; and for each dangerous set $I \subsetneq [c]$, $W(I)$ consists of the two vertices x with $d_{x,I} \geq 2$.

Proof. Let I be dangerous and let x_1 and x_2 be distinct vertices with $d_{x_1,I} + d_{x_2,I} = \Delta_{1,I} + \Delta_{2,I} = s(I)$. Then, as I is not cycle critical,

$$2s(I) - 2 \geq t(I) \geq d_{x_1,I} + d_{x_2,I} + s(I) - 2 = 2s(I) - 2.$$

So $d_{y,I} = 1$ for each $y \in S(I) \setminus \{x_1, x_2\}$, and $W(I) \subseteq \{x_1, x_2\}$. Let $I' = I \setminus 1$. Then I' is not cycle critical either and thus $d_{x_1,I'} + d_{x_2,I'} \leq s(I')$ by Lemma 5.2.11. Now

$$2s(I) - 2 = t(I) = d_{x_1,I} + d_{x_2,I} + s(I) - 2 \leq d_{x_1,1} + d_{x_2,1} + s(I') + s(I) - 2.$$

We saw that $d_{x,1} = 1$ for each $x \in S(1)$, and $W(I) \subseteq \{x_1, x_2\}$. Hence

$$\begin{aligned} 2 &\geq d_{x_1,1} + d_{x_2,1} \geq s(I) - s(I') = (s(I') + s(1) - |W(I)|) - s(I') \\ &= s(1) - |W(I)| \geq s(1) - 2 \geq 2. \end{aligned}$$

Thus $s(1) = 4$ and $W(I) = \{x_1, x_2\}$. $\square$

Claim 13. Let I and J be dangerous. Then $W(I) = W(J)$.

Proof. Let $I, J \subsetneq [c]$ be distinct dangerous sets. By Claim 12, $s(1) = 4$. Now note that $W(I), W(J) \subseteq S(1)$, $v_1 \in S(1)$, and that $v_1 \notin W(I) \cup W(J)$. Suppose that $W(I) \neq W(J)$. Then $W(I) \setminus W(J)$, $W(I) \cap W(J)$ and $W(J) \setminus W(I)$ are each singletons, say $\{x_1\}$, $\{x_2\}$ and $\{x_3\}$; and then $S(1) = \{x_1, x_2, x_3, v_1\}$. Since I is not cycle critical and x_1 and x_2 are in $W(I)$, it follows from Lemma 5.3.3 that x_1x_2 is a directly forced

5.6. Colour degree matrices of multi-graphs with at most one cycle

edge in colour q , for some $q \in I$; and $q \neq 1$, since $d_{x_1,1} + d_{x_2,1} = 2 < 4 = s(1)$. Also note that $S(1) \cap S(q) = \{x_1, x_2\}$. Similarly, x_2x_3 is a directly forced edge in some colour $r \neq 1$, $r \in J$. Further, $x_1 \notin S(J \setminus \{1\})$ (since $x_1 \notin W(J)$), thus $q \neq r$. Let $K = \{1, q, r\}$, then

$$s(K) \leq |S(q) \setminus S(1)| + |S(r) \setminus S(1)| + s(1) = s(q) - 2 + s(r) - 2 + 4 = s(q) + s(r).$$

Therefore, since $d_{x_1,q} + d_{x_2,q} = s(q)$, $d_{x_2,r} + d_{x_3,r} = s(r)$ and $d_{x_1,1} + d_{x_2,1} + d_{x_3,1} = 3$, we have

$$d_{x_1,K} + d_{x_2,K} + d_{x_3,K} \geq s(q) + s(r) + 3 \geq s(K) + 3 :$$

but now we have a directly forced triangle, contradicting Claim 2. $\square$

Let W^* be the non-empty common set $W(I)$ of good vertices for each dangerous set I . We may at last complete the proof of Theorem 5.2.3 by choosing $w \in W^*$. For now, $w \in \tilde{S}(I)$ for each dangerous set I . Hence Eq. 5.11 holds for each dangerous set I by Claim 7. Moreover, Eq. 5.11 holds for each non-dangerous $I \subsetneq [c]$ by Claim 11. Eq. 5.11 holds for $[c]$ by Claim 7, as $W^* \subseteq S(1) \cap S([c] \setminus 1)$, so $w \in \tilde{S}([c])$. Finally conditions (U1) and (U4) hold by Claim 4. We now see that the unicycle conditions hold for $\tilde{D}$ and a unicyclic realisation of $\tilde{D}$ exists by minimality of D . Appending v_1 to w in colour 1 yields a unicyclic realisation of D , contradicting our choice of D . We conclude that the unicycle conditions are sufficient. $\square$

Chapter 6 outlines algorithms to check the unicycle conditions and to find a realisation if one exists. We will end this chapter with corresponding results for multi-graphs and pseudoforests.

5.6 Colour degree matrices of multi-graphs with at most one cycle

In this section we consider unicyclic multi-graph realisations. A multi-graph is a graph which allows multi-edges and self-loops.

5.6.1 Degree sequences of connected unicyclic multi-graphs

What happens if we relax condition (II) from Lemma 5.1.2?

Corollary 5.6.1. *Let $D = (d_1, d_2, \dots, d_n)$ and $d_1 \geq d_2 \geq \dots \geq d_n \geq 1$, be a sequence of integers. D is the degree sequence of a connected unicyclic multi-graph on $[n]$ if and only if $\sum_{i \in [n]} d_i = 2n$.*

Proof. It is clearly necessary. If $d_1 + d_2 \leq n + 1$, Lemma 5.1.2 gives sufficiency. Otherwise $d_1 + d_2 = n + 2$. If $d_1 = n + 1$, D is realised by a self-loop on vertex 1 and all other vertices adjacent to vertex 1. If $d_1 \leq n$, D is realised by a multi-edge between vertex 1 and 2 and all other vertices adjacent to exactly one of vertex 1 and 2, as their degrees require. $\square$

Corollary 5.6.2. *Let $D = (d_1, d_2, \dots, d_n)$ and $d_1 \geq d_2 \geq \dots \geq d_n \geq 1$, be a sequence of integers. D is the degree sequence of a connected unicyclic multi-graph on $[n]$ without self-loops, if and only if $\sum_{i \in [n]} d_i = 2n$ and $d_1, d_2 \geq 2$.*

Proof. If $d_1 + d_2 \leq n + 1$ by Lemma 5.1.2. Otherwise $d_1 + d_2 = n + 2$ and $d_1 \leq n$, as $d_2 \geq 2$. D is realised by a multi-edge between vertex 1 and 2 and all other vertices adjacent to exactly one of vertex 1 and 2, as their degrees require. $\square$

5.6.2 Colour degree matrices of multi-graphs with at most one cycle

What happens when we relax condition (U3)?

Corollary 5.6.3. *The demand matrix $D = (d_{v,q} : v \in [n], q \in [c])$ is the colour degree matrix of a multi-graph with at most one cycle if and only if conditions (U1), (U2) and (U4) hold.*

Proof. It is clear that these conditions are necessary. Suppose that these conditions are not sufficient and that demand matrix $D = (d_{v,q} : v \in [n], q \in [c])$ is a minimal counterexample. Without loss of generality $s(q) > 0$ for all $q \in [c]$ and $d_{v,[c]} > 0$ for all $v \in [n]$. Moreover, there exists a cycle critical set $I \subseteq [c]$, because otherwise D is the colour degree matrix of a forest by Theorem 4.2.3. If $c = 1$, then D can be realised by Corollary 5.6.1, so $c \geq 2$.

If $n = \{v\}$, (U1) gives that $c = 1$, but $c \geq 2$. If $n = \{v, w\}$, then

$$4 \leq 2c \leq t([c]) \leq 2n = 4,$$

by condition (U2) and because $s(q) > 0$ for each $q \in [c]$, so $c = 2$ and $t([c]) = 4$. So either $d_{v,[c]} = 2 = d_{w,[c]}$, or $d_{v,[c]} = 1$ and $d_{w,[c]} = 3$. Let $[c] = \{q, r\}$, then

5.6. Colour degree matrices of multi-graphs with at most one cycle

$t(q) = t(r) = 2$ by (U1). Suppose $d_{v,[c]} = 2 = d_{w,[c]}$, then condition (U4) implies that $d_{v,q} = d_{w,q} = 1 = d_{v,r} = d_{w,r}$. Thus D can be realised by two parallel edges between v and w . Suppose $d_{v,[c]} = 1$ and $d_{w,[c]} = 3$, then $d_{w,t} = 2$ for some $t \in \{q, r\}$ and so D can be realised by a self-loop at w and an edge between v and w . Thus $n \geq 3$.

Now Claim 1 from the proof of Theorem 5.2.3 implies that $[c]$ is the only cycle critical set. Moreover condition (U3) implies that there are at most two vertices $v, w \in [n]$ such that $d_{v,[c]}, d_{w,[c]} \geq 2$. Thus

$$2n = \sum_{i=1}^n \Delta_{i,[c]} = \Delta_{1,[c]} + \Delta_{2,[c]} + \sum_{i=3}^n \Delta_{i,[c]} = \Delta_{1,[c]} + \Delta_{2,[c]} + n - 2.$$

Thus $\Delta_{1,[c]} + \Delta_{2,[c]} = d_{v,[c]} + d_{w,[c]} = n + 2$. Now let $B = \{v, w\}$ and $S = [c] \setminus B$, then $d_{x,[c]} = 1$ for all $x \in S$.

Claim 14. $d_{B,I} \geq d_{S,I} + 2$ for all non-empty sets $I \subseteq [c]$.

Proof. Suppose there is a non-empty set $I \subsetneq [c]$ such that $d_{B,I} \leq d_{S,I}$. Set $J = [c] \setminus I$ (note $J \neq \emptyset$). Then

$$\begin{aligned} d_{B,J} &= d_{B,[c]} - d_{B,I} \geq n + 2 - d_{S,I} = n - (s(I) - |S(I) \cap B|) + 2 \\ &\geq n - s(I) + |S(I) \cap S(J)| + 2 = s(J) + 2, \end{aligned}$$

i.e. J is cycle critical, which is a contradiction. Thus $d_{B,I} > d_{S,I}$ for all non-empty $I \subseteq [c]$, and because their sum must always be even, $d_{B,I} \geq d_{S,I} + 2$. $\square$

As

$$n + 2 = \sum_{q \in [c]} d_{B,q} \geq \sum_{q \in [c]} \{d_{S,q} + 2\} = 2c + \sum_{q \in [c]} d_{S,q} = 2c + n - 2,$$

$2 \geq c \geq 2$ and $d_{B,t} = d_{S,t} + 2$ for $t \in [c]$. Let $[c] = \{q, r\}$. Suppose $d_{v,q} = 0$, then $d_{w,q} = d_{S,q} + 2 = s(q) - 1 + 2$, and $d_{w,q} + d_{S,q} = 2s(q)$, i.e. q is cycle critical, which is a contradiction. Thus $d_{v,q}, d_{w,q}, d_{v,r}, d_{w,r} \geq 1$ and D can be realised by a multi-edge between v and w with colour r and q and all $x \in S$ adjacent to exactly one of v and w , as their degrees require. $\square$

Corollary 5.6.4. *The demand matrix $D = (d_{v,q} : v \in [n], q \in [c])$ is the colour degree matrix of a multi-graph with at most one cycle and without self-loops if and only if conditions (U1), (U2) and (U4) hold and*

(U3') each cycle critical set I contains at least 2 vertices v with $d_{v,I} \geq 2$.

Chapter 6 outlines algorithms to check these conditions and to find a realisation if one exists. We will end this chapter with corresponding results for pseudoforests.

5.7 Colour degree matrices of pseudoforests

Definition 5.7.1. A *pseudoforest* is a simple graph in which every connected component has at most one cycle. A *pseudotree* is a connected pseudoforest.

In the monochromatic case a pseudoforest with at most k cyclic components can be turned into a unicyclic graph by merging cycles. It is thus enough to consider the coloured case; what happens if we relax condition (U4)?

Observe that $(d_1, \dots, d_n)$ has a pseudoforest realisation if and only if it has a realisation where each component is a pseudotree. Clearly, for each $I \subseteq [c]$, it is necessary that $(d_{1,I}, \dots, d_{n,I})$ is the degree sequence of a pseudoforest.

Theorem 5.7.2. The demand matrix $D = (d_{v,q} : v \in [n], q \in [c])$ is the colour degree matrix of a pseudoforest if and only if for each $I \subseteq [c]$, the sequence $(d_{1,I}, \dots, d_{n,I})$ is the degree sequence of a pseudoforest. Further, the minimal number of cyclic components in such a realisation of D is the number of minimal cycle critical sets.

Let us restate Theorem 5.7.2, as we did for Theorem 5.2.2 and Theorem 4.2.1.

Theorem 5.7.3. The demand matrix $D = (d_{v,q} : v \in [n], q \in [c])$ is the colour degree matrix of a pseudoforest with at most k cyclic components if and only if

- (M1) $t(q)$ is even for each $q \in [c]$,
- (M2) $t(I) \leq 2s(I)$ for each $I \subseteq [c]$,
- (M3) $s(I) \geq 2$ and $\Delta_{1,I} + \Delta_{2,I} \leq s(I) + 1$ for each cycle-critical $I \subseteq [c]$, and
- (M4) there are at most k minimal cycle critical sets.

Proof. Condition (M1) is always necessary in graph realisations. Condition (M2) is the maximum number of edges in any unicyclic component. Condition (M3) is the maximum joint degree in a of union unicyclic components. Finally, condition (M4) is necessary because every disjoint minimal cycle critical set yields at least one cycle in any realisation.

Assume condition (M1)-(M4) hold for D and that there is no $v \in [n]$ such that

$$\sum_{q \in [c]} d_{v,q} = 0, \quad (5.12)$$

and no $q \in [c]$ such that

$$\sum_{v \in [n]} d_{v,q} = 0. \quad (5.13)$$

If $n \leq 3$ or $k \leq 1$, then there are no disjoint cycle critical sets and there exists a realisation with at most one cycle by Theorem 5.2.3. Suppose $n \geq 4$ and $k \geq 2$ and let $I_1, I_2, \dots, I_k$ be disjoint minimal cycle critical sets. Suppose $S(I_i) \cap S(I_j) \neq \emptyset$ for some $i \neq j$. Then

$$\begin{aligned} t(I_i \cup I_j) &= t(I_i) + t(I_j) = 2(s(I_i) + s(I_j)) \\ &> 2(s(I_i) + s(I_j) - |S(I_i) \cap S(I_j)|) = 2s(I_i \cup I_j). \end{aligned}$$

as I_i and I_j are disjoint and $|S(I_i) \cap S(I_j)| \geq 1$, contradicting condition (M2). Thus $S(I_i) \cap S(I_j) = \emptyset$, for $1 \leq i < j \leq k$. If $I = \bigcup_{i=1}^k I_i = [c]$, then by Eq. 5.12, $S(I) = [n]$. For $1 \leq i \leq k$, let $D(I_i) = (d_{v,q} : v \in S(I_i), q \in I_i)$. By Theorem 5.2.3 there exists a unicyclic realisation G_i of each $D(I_i)$. Putting them together gives a realisation of D with k cycles.

Otherwise let $J = [c] \setminus I$. By Eq. 5.13, $t(J) \neq \emptyset$ and thus by condition (M2), $S(I) \subsetneq [n]$. In particular this means that $S(J) \setminus S(I) \neq \emptyset$. Let a be a new vertex and for all $q \in J$, let $d_{a,q} = \sum_{v \in S(I)} d_{v,q}$. Consider

$$\tilde{D} = (d_{v,q} : v \in a \cup S(J) \setminus S(I), q \in J). \quad (5.14)$$

If $\tilde{D}$ is the colour demand matrix of a forest T , then replacing a by all G_i and connecting edges correctly, gives a realisation G of D . Moreover, if any two G_i were in the same component in G , T would have a cycle. Thus G has no multicyclic components and exactly k unicyclic components, with disjoint colour sets on each cycle.

For $K \subseteq J$, set $\tilde{s}(K) = s_{\tilde{D}}(K)$ and $\tilde{t}(K) = t_{\tilde{D}}(K)$. As $\tilde{t}(K) = t(K)$ for all $K \subseteq J$, condition (1) of Theorem 4.2.3 still holds. It is enough to check that $\tilde{t}(K) \leq 2\tilde{s}(K) - 2$ for all $K \subseteq J$.

If $S(K) \cap S(I) = \emptyset$, then $\tilde{s}(K) = s(K)$ and $t(K) \leq 2s(K) - 2$, or condition (M4) would be violated. If $S(K) \cap S(I) \neq \emptyset$, then $s(K) = \tilde{s}(K) + |S(K) \cap S(I)| - 1$. Moreover, $t(K) \leq 2s(K) - 2|S(K) \cap S(I)|$, as by condition (M2)

$$2(s(K) + s(I) - |S(K) \cap S(I)|) = 2s(K \cup I) \geq t(K \cup I) = t(K) + t(I) = t(K) + 2s(I).$$

Rearranging gives the required inequality. In this case,

$$\begin{aligned}\tilde{t}(K) &= t(K) \leq 2s(K) - 2|S(K) \cap S(I)| \\ &= 2(\tilde{s}(K) + |S(K) \cap S(I)| - 1 - |S(K) \cap S(I)|) \\ &= 2\tilde{s}(K) - 2.\end{aligned}$$

Thus $\tilde{D}$ is the colour demand matrix of a forest, which yields a pseudoforest realisation of D . $\square$

5.8 Conclusion

Chapter 6 outlines algorithms to check the conditions from Theorems 4.2.3, 5.2.3, 5.6.3, and 5.7.3, and to find a corresponding realisation if it exists.

Another extension to consider is demand sequences of directed graphs.

5.8.1 Directed degree sequences of unicyclic graphs

Definition 5.8.1. Let $D^+ = (d_1^+, d_2^+, \dots, d_n^+)$ and $D^- = (d_1^-, d_2^-, \dots, d_n^-)$ be sequences of non-negative integers. Then (D^+, D^-) is the **directed degree sequence** of a directed graph G if each $v \in [n]$ has out-degree d_v^+ and in-degree d_v^- .

Lemma 5.8.2. Let $D^+ = (d_1^+, d_2^+, \dots, d_n^+)$ and $D^- = (d_1^-, d_2^-, \dots, d_n^-)$ be sequences of non-negative integers. If (D^+, D^-) is the directed degree sequence of a directed unicyclic graph G , then

$$\sum_{v \in [n]} d_v^+ = \sum_{v \in [n]} d_v^- \leq s, \quad (5.15)$$

where $s = |\{v \in [n] : d_v^+ + d_v^- > 0\}|$.

Proof. By the handshaking lemma $\sum_{v \in [n]} d_v^+ = \sum_{v \in [n]} d_v^- = |E(G)|$, and $|E(G)| \leq s$, because G is unicyclic. $\square$

However, Eq. 5.15 is not sufficient. Let $D^+ = (2, 0, 1)$ and $D^- = (1, 2, 0)$. Then $\sum_{v \in [3]} d_i^+ = \sum_{v \in [3]} d_v^- = 3$, but (D^+, D^-) can never have a simple realisation, because vertex 1 needs to be the tail of an edge to every other vertex and vertex 3 has in-degree 0. So another necessary conditions is that for all $v \in [n]$

$$d_v^+ \leq |\{w \neq v : d_w^- > 0\}|,$$

and similarly for d_v^- . As opposed to the directed forest case, it is not obvious what necessary and sufficient conditions would be, and indeed, it is not in the scope of this thesis to find these.

6

Algorithms for realising colour demand matrices

In this chapter we will give algorithms to check the unicycle conditions and find a realisation if one exists. Recall:

Theorem 5.2.3. *The demand matrix $D = (d_{v,q} : v \in [n], q \in [c])$ is the colour degree matrix of a graph with at most one cycle if and only if*

- (U1) *$t(q)$ is even for each $q \in [c]$,*
- (U2) *$t(I) \leq 2s(I)$ for each $I \subseteq [c]$,*
- (U3) *there are at least 3 vertices v with $d_{v,I} \geq 2$ for each cycle critical set $I \subseteq [c]$, and*
- (U4) *there are no two disjoint cycle critical sets.*

We assume $t(q) > 0$ for each $q \in [c]$. Then we may also assume $c \leq n$, as D would not satisfy condition (U2) otherwise. We show that the unicycle conditions can be checked in $O(n^2)$ time, and find a violating set if they do not hold. We extend these results to forests, multi-graphs, and pseudoforests. Finally we show that the required realisations can be found in $O(n^3)$ time if they exist, by following the steps of the proofs of Theorems 4.2.3, 5.2.3, 5.6.3, and 5.7.3.

Clearly we can check condition (U1) in $O(n^2)$ time. We first give an $O(n^2)$ time algorithm to check condition (U2). We then introduce ‘weakly intersecting’ and ‘sub-intersecting’ families of sets, and give a polynomial time algorithm to find minimal sets in such families. We show that the set of all cycle critical sets is weakly

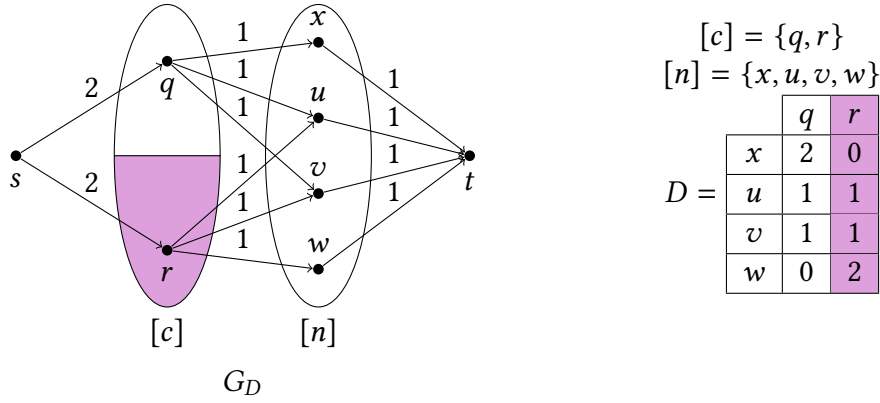


Figure 6.1: Example G_D graph.

intersecting and use this algorithm to prove that conditions (U3) and (U4) can be checked in $O(n^2)$ time. Given all minimal cycle critical sets, all directly forced triangles can be found in $O(n)$ time per minimal set. These methods generalise to the conditions from Theorems 4.2.3, 5.2.3, 5.6.3, or 5.7.3 and all directly forced edges can be found in $O(n^2)$ time. Finally, given an algorithm to check the conditions from Theorems 4.2.3, 5.2.3, 5.6.3, and 5.7.3, we show how to find a realisation by following the steps of the corresponding proof. We end this chapter by giving an overview of the complexity of all discussed methods.

6.1 Checking condition (U2)

Fix $D = (d_{v,s} : v \in [n], s \in [c])$. Without loss of generality we may assume that

$$\sum_{s \in [c]} d_{v,s} \geq 1, \forall v \in [n], \text{ and}$$

$$\sum_{v \in [n]} d_{v,s} \geq 1, \forall s \in [c],$$

since we can simply ignore any vertices with no demand and any unwanted colours.

We will show that condition (U2) is equivalent to finding a maximum flow in a related flow graph G_D .*

Definition 6.1.1. Given a demand matrix $D = (d_{v,q} : v \in [n], q \in [c])$, let $G_D = (V_D, E_D)$ be a directed bipartite flow graph with source s and sink t , and

$$V_D = s \cup [c] \cup [n] \cup t$$

*For more information on flow graphs see Section 2.5.

6.1. Checking condition (U2)

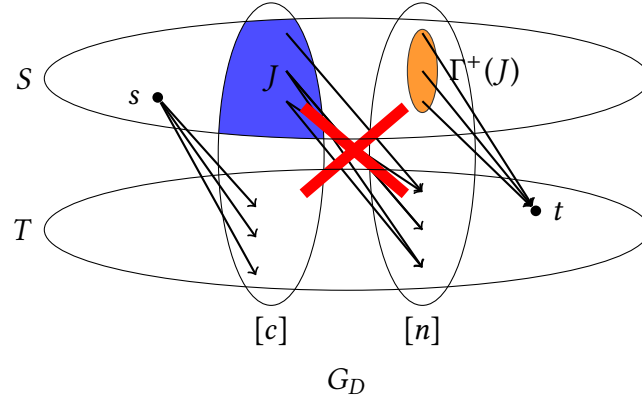


Figure 6.2: A visual guide to the proof of Lemma 6.1.2.

$$E_D = \{\overrightarrow{sq} : q \in [c]\} \cup \{\overrightarrow{qv} : q \in [c], v \in S(q)\} \cup \{\overrightarrow{vt} : v \in [n]\},$$

as illustrated in Fig. 6.1. Edge $\overrightarrow{xy} \in E_D$ has capacity

$$c(x, y) = \begin{cases} \frac{t(q)}{2} & \text{if } x = s \text{ and } y \in [c], \\ 1 & \text{otherwise.} \end{cases}$$

Lemma 6.1.2. *There is a flow of volume $\frac{t([c])}{2}$ in G_D if and only if condition (U2) holds in D .*

Proof. Suppose there is a flow f of volume $\frac{t([c])}{2}$. Let $J \subseteq [c]$. Set $S = s \cup J \cup \Gamma^+(J)$ and $T = V_D \setminus S$ as in Fig. 6.2. Then (S, T) is an $s - t$ cut, so $c(S, T) \geq v(f)$ by the max-flow/min-cut theorem (Theorem 2.5.7) and

$$\begin{aligned} c(S, T) &= c(s, [c] \setminus J) + c(J, [n] \setminus \Gamma^+(J)) + c(\Gamma^+(J), t) \\ &= \frac{t([c]) - t(J)}{2} + 0 + |\Gamma^+(J)| \\ &= \frac{t([c]) - t(J)}{2} + s(J) \\ &\geq v(f) = \frac{t([c])}{2}. \end{aligned}$$

Rearranging gives $t(J) \leq 2s(J)$ and thus condition (U2) holds.

Suppose condition (U2) holds in D and that (S, T) is a minimum $s - t$ cut such that $S \neq \{s\}$. As $(\{s\}, V_D \setminus s)$ is an $s - t$ cut in G_D ,

$$\frac{t([c])}{2} = c(\{s\}, V_D \setminus s) \geq c(S, T).$$

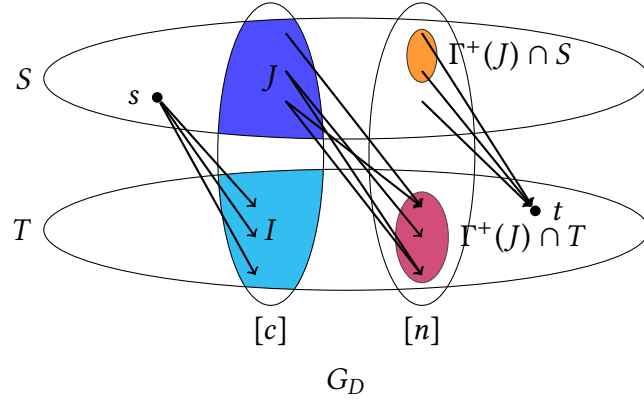


Figure 6.3: A visual guide to the proof of Lemma 6.1.2.

Set $S \cap [c] = J$ and $[c] \setminus S = I$. See Fig. 6.3 for a visualisation. Then $I \cup J = [c]$ and

$$\begin{aligned}
 c(S, T) &= c(s, I) + c(J, T \cap [n]) + c(S \cap [n], t) \\
 &= c(s, I) + c(J, T \cap \Gamma^+(J)) + c(S \cap [n], t) \\
 &\geq c(s, I) + c(J, T \cap \Gamma^+(J)) + c(S \cap \Gamma^+(J), t) \\
 &= \frac{t(I)}{2} + c(J, T \cap S(J)) + |S \cap \Gamma^+(J)| \\
 &\geq \frac{t(I)}{2} + |T \cap S(J)| + |S \cap S(J)| \\
 &= \frac{t(I)}{2} + s(J) \\
 &\geq \frac{t(I)}{2} + \frac{t(J)}{2} = \frac{t[c]}{2}.
 \end{aligned}$$

Therefore a minimum cut has capacity $\frac{t[c]}{2}$ and the maximum flow has volume $\frac{t[c]}{2}$ by max-flow/min-cut theorem (Theorem 2.5.7). $\square$

Lemma 6.1.3. *Let f be a flow in G_D and suppose there is no augmenting path starting with edge $\vec{s}q$. Let I_q be the set of vertices in $[c]$, visited on a (breadth first) search for an augmenting path in G_f , which starts with $\vec{s}q$. Then*

$$f^-(I_q) = s(I_q).$$

Proof. Let $J = [c] \setminus I_q$ and let W be the set of vertices in $[n]$, that we visit on our search; $W \subseteq \Gamma_{G_D}^+(I_q)$. See Fig. 6.4 step 0 for a visualisation of the different vertex sets in G_f (the residual flow graph left) and G_D (the original flow graph right). By showing the absence of edges between different sets in G_f , we will be able to deduce the volume of the flow between these sets in G_D and the result will follow.

6.1. Checking condition (U2)

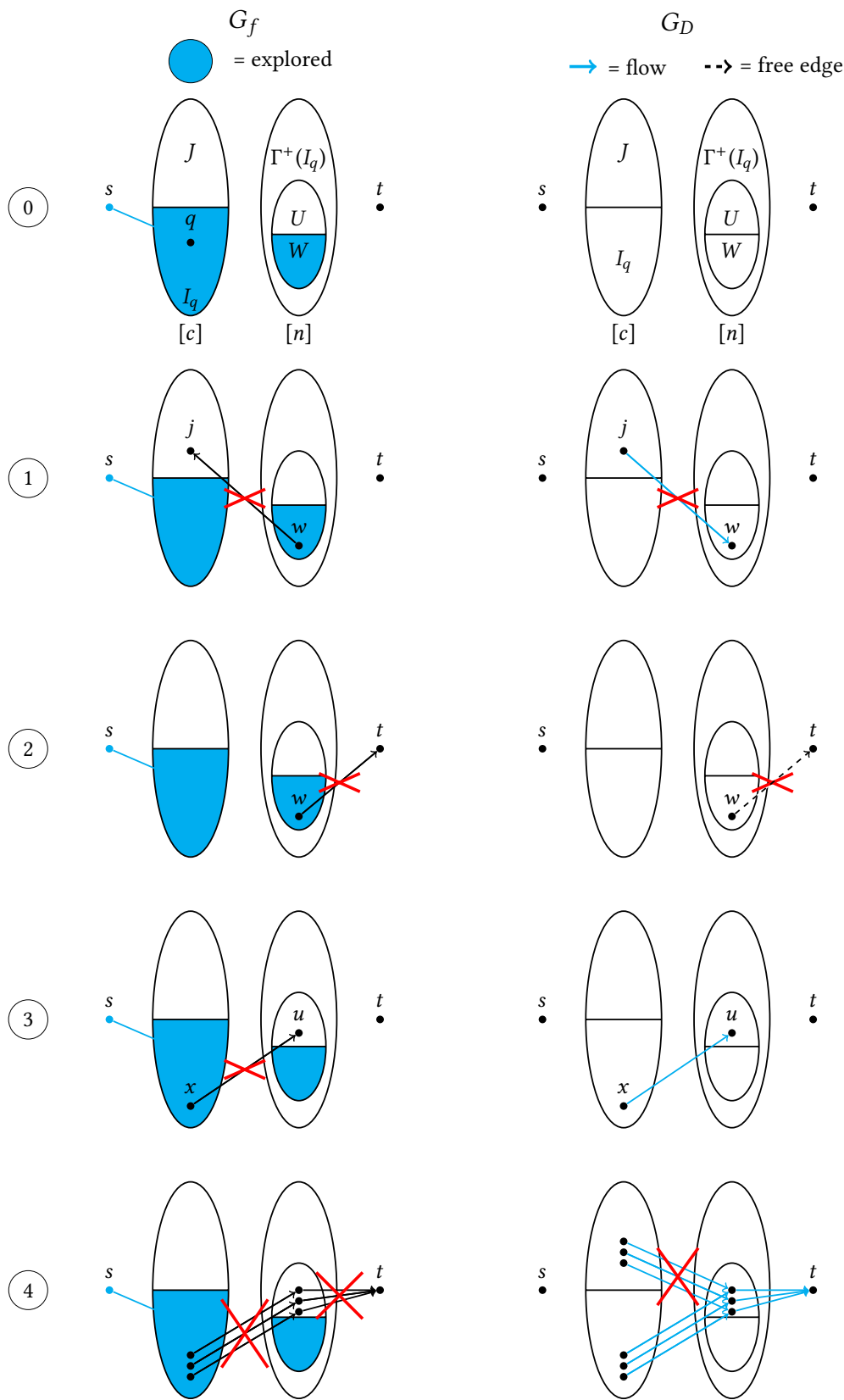


Figure 6.4: A visual guide to the proof of Lemma 6.1.3.

Suppose there exists $w \in W$ and $j \in J$ such that $f(j, w) > 0$, then $\vec{wj} \in E_f$ and would be explored during the search. This implies $j \in I_q$, contradicting the definition of J . Thus $f(j, w) = 0$ for all $j \in J$ and $w \in W$. See Fig. 6.4 step 1.

Suppose there exists $w \in W$ such that $f(w, t) < c(w, t) = 1$. At some point during the search we visit w . At this point we would have found an augmenting path from $\vec{s}q$ ending with $\vec{wt}$. This contradicts our assumptions and therefore $f(w, t) = c(w, t) = 1$ for all $w \in W$. See Fig. 6.4 step 2.

Let $U = \Gamma_{G_D}^+(I_q) \setminus W$. For all $u \in U$, there exists $x \in I_q$ such that $f(x, u) = c(x, u) = 1$. Otherwise $\vec{xu} \in E_f$ and would be explored, which would imply $u \in W$. See Fig. 6.4 step 3. Note that $\Gamma^+(u) = \{t\}$ and thus $1 = c(u, \Gamma^+(u)) \geq f^+(u)$ by definition of flow. Conservation of flow gives

$$1 \geq f^+(u) = f^-(u) = \sum_{y \in \Gamma^-(u)} f(y, u) \geq f(x, u) = 1,$$

i.e. that $f^+(u) = f(u, t) = 1$. Thus $f(u, t) = 1 = c(u, t)$ for all $u \in U$ and there is a unique $x \in [c]$ such that $f(x, u) = 1$. Moreover, we observed above that $x \in I_q$. See Fig. 6.4 step 4.

We can now conclude that $f(J, \Gamma_{G_D}^+(I_q)) = 0$. Moreover

$$\begin{aligned} f^-(\Gamma_{G_D}^+(I_q)) &= f([c], \Gamma_{G_D}^+(I_q)) \\ &= f(J, \Gamma_{G_D}^+(I_q)) + f(I_q, \Gamma_{G_D}^+(I_q)) \\ &= f(I_q, \Gamma_{G_D}^+(I_q)) \\ &= f^+(I_q) = f^-(I_q), \end{aligned}$$

and

$$\begin{aligned} f^+(\Gamma_{G_D}^+(I_q)) &= f^+(U) + f^+(W) \\ &= c(U, t) + c(W, t) \\ &= |U| + |W| \\ &= |\Gamma_{G_D}^+(I_q)| = s(I_q). \end{aligned}$$

Observing that $f^-(\Gamma_{G_D}^+(I_q)) = f^+(\Gamma_{G_D}^+(I_q))$ concludes the proof. $\square$

Corollary 6.1.4. *Let f be a flow in G_D . Suppose $f(s, q) < c(s, q)$ and that there is no augmenting path starting with edge $\vec{s}q$. Recall that I_q is the set of vertices in $[c]$, visited on a (breadth first) search for an augmenting path in G_f , which starts with $\vec{s}q$. Then I_q*

violates condition (U2).

Proof. By Lemma 6.1.3

$$\frac{t(I_q)}{2} = c(s, I_q) > f^-(I_q) = s(I_q). \quad \square$$

6.1.1 Algorithm sketch

First we check in $O(n^2)$ time that $t([c]) \leq 2n$. The Ford-Fulkerson algorithm finds a maximum integer flow in flow graphs $G = (V, E)$ with integral flow in $O(v(f)|E|)$ time, where $v(f)$ is the volume of the maximum flow.[†] We observed in the proof of Lemma 6.1.2 that the maximum flow in G_D is at most $\frac{t([c])}{2} \leq n$. Moreover

$$|E_D| = c + n + \sum_{v \in [n]} \sum_{q \in [c]} 1_{v \in S(q)} \leq c + n \sum_{v \in [n]} \sum_{q \in [c]} d_{v,q} = c + n + t([c]) \leq 4n.$$

The Ford-Fulkerson algorithm finds a maximum volume flow by repeatedly finding an augmenting path in the residual flow graph. Suppose that $\vec{s}q$ is an edge such that $f(s, q) < c(s, q) = \frac{t(q)}{2}$. Let I_q be the set of vertices in $[c]$ that we visited on a search for an augmenting path from edge $\vec{s}q$, then Corollary 6.1.4 gives that I_q is a violating set.

Thus condition (U2) can be checked in $O(n^2)$ time and a violating set can be returned in $O(|E|) = O(n)$ additional time, if the condition does not hold.

6.2 Finding minimal sets in intersecting families

In this section we will show that the minimal sets in a sub-intersecting family can be found in $O(mn)$ time, where m is the number of sets in the family and n is the number of elements in the ground set. We will use this algorithm to find minimal cycle critical sets in Section 6.3.

6.2.1 Intersecting families

Definition 6.2.1. For any set or family of sets $\mathcal{A}$, we say $A \in \mathcal{A}$ is **minimal** (with respect to $\mathcal{A}$) if for all $B \in \mathcal{A}$, $B \subseteq A$ implies $A = B$.

Let $\text{minl}(\mathcal{A})$ denote the set of all minimal sets in $\mathcal{A}$ (so there are no repetitions in $\text{minl}(\mathcal{A})$), and if $\mathcal{A}$ is empty, then so is $\text{minl}(\mathcal{A})$.

[†]See Subsection 2.5.2.

Let us call a family $\mathcal{A}$ of non-empty sets **weakly intersecting** if for all $A, B \in \mathcal{A}$ for which $A \cap B \neq \emptyset$, we have $A \cap B \in \mathcal{A}$.

Call $\mathcal{A}$ **sub-intersecting** if for all $A, B \in \mathcal{A}$ for which $A \cap B \neq \emptyset$, there is $X \in \mathcal{A}$ such that $X \subseteq A \cap B$.

Note that a weakly intersecting family is also sub-intersecting.

Lemma 6.2.2. *Let $\mathcal{A}$ be sub-intersecting. Then all sets in $\text{minl}(\mathcal{A})$ are pairwise disjoint.*

Proof. Suppose $M \cap N \neq \emptyset$ for distinct $M, N \in \text{minl}(\mathcal{A})$. Then there is $A \in \mathcal{A}$ such that $A \subseteq M \cap N$ and $A \subseteq M \cap N \subsetneq M$ since $M \not\subseteq N$. But then M is not minimal, which is a contradiction. We conclude that $M \cap N = \emptyset$ for all distinct $M, N \in \text{minl}(\mathcal{A})$. $\square$

Lemma 6.2.3. *Let $\mathcal{A}$ be a family of sets. Then for each $A \in \mathcal{A}$, there is $M \in \text{minl}(\mathcal{A})$ such that $M \subseteq A$.*

Let $B = \bigcup_{A \in \mathcal{A}} A$, the union of all sets in $\mathcal{A}$, also referred to as the **ground set** of $\mathcal{A}$.

Lemma 6.2.4. *Let $\mathcal{A}$ be a weakly intersecting family and let B be the ground set of $\mathcal{A}$. For each $i \in B$, let $C_i = \bigcap_{A \in \mathcal{A}: i \in A} A$, that is, the intersection of all sets in $\mathcal{A}$ that contain i and let $\mathcal{C} = (C_i : i \in B)$. Then $\mathcal{C} \subseteq \mathcal{A}$ is sub-intersecting and*

$$\text{minl}(\mathcal{C}) = \text{minl}(\mathcal{A}).$$

Proof. As $\mathcal{A}$ is weakly intersecting, $\mathcal{C} \subseteq \mathcal{A}$. Suppose $A \in \text{minl}(\mathcal{A})$ and $i \in A$. Then C_i is the intersection of all sets containing i , so $C_i = C_i \cap A \subseteq A$; minimality of A implies $C_i = A$ and $A \in \mathcal{C}$. So $\text{minl}(\mathcal{A}) \subseteq \mathcal{C} \subseteq \mathcal{A}$ and thus $\text{minl}(\mathcal{A}) \subseteq \text{minl}(\mathcal{C})$.

Let $C_i \in \text{minl}(\mathcal{C})$ and suppose C_i is not minimal in $\mathcal{A}$. Then by Lemma 6.2.3 there is $M \in \text{minl}(\mathcal{A})$ such that $M \subsetneq C_i$. Thus $i \notin M$, because by definition of C_i , $C_i \subseteq A$ for all $A \in \mathcal{A}$ which contain i . Let $j \neq i$ and suppose $j \in M$, then $C_j = M \in \mathcal{C}$ as above. Hence C_i is not minimal in $\mathcal{C}$, which gives the required contradiction. Thus $\text{minl}(\mathcal{C}) \subseteq \text{minl}(\mathcal{A})$ and we conclude that $\text{minl}(\mathcal{C}) = \text{minl}(\mathcal{A})$.

For each $A, B \in \mathcal{C} \subseteq \mathcal{A}$ for which $A \cap B \neq \emptyset$, $A \cap B \in \mathcal{A}$. By Lemma 6.2.3 there is $M \in \text{minl}(\mathcal{A}) = \text{minl}(\mathcal{C})$ such that $M \subseteq A \cap B$. So $\mathcal{C}$ is sub-intersecting. $\square$

6.2.2 Finding minimal sets in intersecting families

Consider the following pseudocode:

6.2. Finding minimal sets in intersecting families

Input: A family $\mathcal{A}$ of m non-empty subsets from $[n]$, represented by an m by n (0,1)-matrix.

Output: The set $\mathcal{M}$ of minimal sets in $\mathcal{A}$, i.e. $\text{minl}(\mathcal{A})$.

- List the sets in $\mathcal{A}$ in non-decreasing order of cardinality in a linked list L .
- Set $\mathcal{M}$ to be empty.
- While L is non-empty:
 - Set X to be the first member of L .
 - Add X to $\mathcal{M}$.
 - Remove from L each Y for which $X \subseteq Y$.
- Return $\mathcal{M}$.

Lemma 6.2.5. *The pseudocode above returns $\text{minl}(\mathcal{A})$; that is $\mathcal{M} = \text{minl}(\mathcal{A})$.*

Proof. Let $\mathcal{M}$ be the set returned by the algorithm. Then $\text{minl}(\mathcal{A}) \subseteq \mathcal{M}$, because all $M \in \text{minl}(\mathcal{A})$ are only removed from L when they are added to $\mathcal{M}$, as no $M \in \text{minl}(\mathcal{A})$ is a strict superset of any $A \in \mathcal{A}$. Now suppose $M \in \mathcal{M}$, but $M \notin \text{minl}(\mathcal{A})$. Then, as M is not minimal, there is $X \in \mathcal{A}$ such that $X \subsetneq M$; then $|X| < |M|$ and X must appear in L before M . If X was removed from L because it was added to $\mathcal{M}$, M would be removed from L and therefore not be in $\mathcal{M}$. If X was removed from L because there is $Y \in \mathcal{M}$ such that $Y \subseteq X$, then $Y \subsetneq M$ and M would be removed from L . So there is no such $M \in \mathcal{M}$ and thus $\mathcal{M} \subseteq \text{minl}(\mathcal{A})$. We conclude that $\mathcal{M} = \text{minl}(\mathcal{A})$ as required. $\square$

Lemma 6.2.6. *Suppose $\mathcal{A}$ is a family of m non-empty subsets from $[n]$, specified by the rows of an m by n (0,1)-matrix. Let $X \subseteq [n]$, then all sets $A \in \mathcal{A}$ that contain X can be found in $O(m|X|)$ time.*

Proof. Each $A \in \mathcal{A}$ is such that $X \subseteq A$ if and only if $A[x] = 1$ for all $x \in X$. This can be checked in $O(|X|)$ time per set $A \in \mathcal{A}$. This gives an overall time complexity of $O(m|X|)$. $\square$

Lemma 6.2.7. *If $\mathcal{A}$ is a sub-intersecting family, the above pseudocode has time complexity $O(m \log m + mn)$.*

Proof. Determining the sizes of all sets takes $O(mn)$ time. To sort a list of m numbers takes $O(m \log m)$ time. Deletion and insertion into a linked list takes $O(1)$ time. For each $X \in \mathcal{M} = \text{minl}(\mathcal{A})$ it takes $O(m|X|)$ time to remove each $A \in \mathcal{A}$ for which

$X \subseteq A$ by Lemma 6.2.6. Lemma 6.2.2 gives that all $X \in \text{minl}(\mathcal{A})$ are disjoint, so the total time the while loop takes is

$$\sum_{X \in \text{minl}(\mathcal{A})} O(|X|m) = O(nm).$$

This gives an overall time complexity of $O(m \log m + mn)$. $\square$

6.3 Checking conditions (U3) and (U4)

In this section we show how to find minimal cycle critical sets. We saw that if (U2) holds then the collection of cycle critical sets is weakly intersecting. Let I'_q be the minimal cycle critical set containing q if q is in any cycle critical set and empty otherwise. We first show that $I'_q = I_q$ for all $q \in [c]$, where I_q is as in Lemma 6.1.3, and how to find $\mathcal{J} = \{I_q : q \in [c] \text{ and } I_q \neq \emptyset\}$. Finally we show that we can find all minimal cycle critical sets, by finding $\text{minl}(\mathcal{J})$.

6.3.1 Finding cycle critical sets

Suppose we checked condition (U2) and it holds. Then we found a flow f of volume $\frac{t([c])}{2}$ in $G_D = G$. Set $c(s, q) = \frac{t(q)}{2} + 1$ for some $q \in [c]$. We look for an augmenting path in the residual flow graph G_f , starting with edge $\overrightarrow{sq}$. If we find such a path, let g be flow f augmented along this path. Then q is in no cycle critical sets, as for all $I \subseteq [c]$ with $q \in I$

$$\begin{aligned} 1 + \frac{t(I)}{2} &= g^-(I) = g^+(I) \\ &\leq g^-(\Gamma_G^+(I)) = g^+(\Gamma_G^+(I)) \\ &\leq c(\Gamma_G^+(I), t) = |\Gamma_G^+(I)| = s(I) \end{aligned}$$

and so $t(I) \leq 2s(I) - 2$. Let I_q be empty if there is an augmenting path starting with $\overrightarrow{sq}$, that is, q is in no cycle critical sets.

Now suppose we find no such path. Let I_q be the vertices in $[c]$ we visit while searching for an augmenting path starting with $\overrightarrow{sq}$.

Lemma 6.3.1. *Let I'_q and I_q be defined as above. Then $I'_q = I_q$ for all $q \in [c]$.*

Proof. Suppose $I_q \neq \emptyset$. Then $\frac{t(I_q)}{2} = f^-(I_q) = s(I_q)$ by Lemma 6.1.3, i.e. I_q is cycle critical. We need to show that I_q has no strict cycle critical subsets containing q , as

6.3. Checking conditions (U3) and (U4)

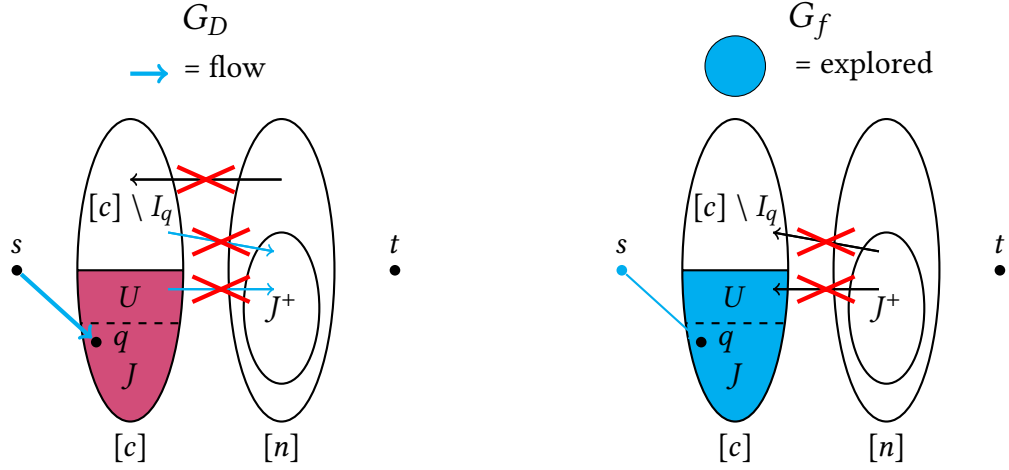


Figure 6.5: A visual guide to the proof of Lemma 6.3.1.

then $I'_q = I_q$. We will write E_A for edges in G_A and Γ_A for the neighbourhood in G_A , where $A \in \{D, f\}$. Suppose $J \subsetneq I_q$ is cycle critical and $q \in J$. Set $U = I_q \setminus J$ and $J^+ = \Gamma_D^+(J)$. As $f(s, r) = \frac{t(r)}{2}$ for all $r \in [c]$, we must have $f^-(J) = c(s, J) = \frac{t(J)}{2}$ and

$$\begin{aligned} s(J) &= \frac{t(J)}{2} = f^-(J) = f^+(J) = f(J, J^+) \\ &\leq f(J, J^+) + f(U, J^+) + f([c] \setminus I_q, J^+) = f^-(J^+) = f^+(J^+) \\ &\leq c(J^+, t) = s(J). \end{aligned}$$

Therefore $f(U, J^+) + f([c] \setminus I_q, J^+) = 0$ and because $E_D([n], [c]) = \emptyset$,

$$f(U, J^+), f([c] \setminus I_q, J^+) \geq 0.$$

This implies $f(U, J^+) = f([c] \setminus I_q, J^+) = 0$. Thus $E_f(J^+, U) = \emptyset$ and $E_f(J^+, [c] \setminus I_q) = \emptyset$. See Fig. 6.5 for an illustration. Note that $E_D([n], [c]) = \emptyset$ also implies that for all $I \subseteq [c]$, $\Gamma_f^+(I) \subseteq \Gamma_D^+(I)$.

Now we have everything to show that all vertices at distance $2i$ ($i \geq 0$) from q in the BFS tree we create during the search, are in J . We will do this by induction on i . At distance 0 we have q and $q \in J$. Now consider A_{2i} , the vertices at distance $2i$, $i \geq 1$, and suppose it is true for smaller values. The vertices $A_{2(i-1)}$, at distance $2(i-1)$ from q , are a subset of J by the inductive assumption, so $\Gamma_f^+(A_{2(i-1)}) \subseteq \Gamma_D^+(A_{2(i-1)}) \subseteq J^+$. Thus the vertices at distance $2i-1$ are in J^+ . There are only edges between J^+ and J in G_f , so from there we will visit $A_{2i} \subseteq J$. Thus $I_q \subseteq J$, which contradicts $J \subsetneq I_q$. We conclude that no strict cycle critical subsets containing q exist.

If $I_q = \emptyset$ then q is in no cycle critical set and $I'_q = \emptyset$. If q is in no cycle critical

sets and $I'_q = \emptyset$, then I_q must be empty as well, as otherwise I_q would be a cycle critical set containing q . Thus $I'_q = I_q$. $\square$

6.3.2 Finding minimal cycle critical sets

Let $\mathcal{C} = \{I \subseteq [c] : t(I) = 2s(I)\}$, i.e all cycle critical sets in D . Lemma 6.3.1 shows that I_q is the minimal cycle critical set containing q . Thus for each $q \in [c]$ we can check whether q appears in a cycle critical set and if so, what the minimal cycle critical set that q appears in is. Let $\mathcal{J} = \{I_q : q \in [c] \text{ and } I_q \neq \emptyset\}$. We will show in this section that $\mathcal{C}$ is weakly intersecting and that $\mathcal{J}$ is sub-intersecting. We use that to show that given $\mathcal{J}$, $\text{minl}(\mathcal{C})$ can be found in $O(n^2)$ time.

Lemma 6.3.2. *If condition (U2) holds, $\mathcal{C}$ is a weakly intersecting family.*

Proof. By condition (U2) and Lemma 5.2.5. $\square$

Lemma 6.3.3. *Let $q \in [c]$ be in some cycle critical set. Then $I_q = \bigcap_{I \in \mathcal{C}: q \in I} I$.*

Proof. Let $C_q = \bigcap_{I \in \mathcal{C}: q \in I} I$. Then $C_q \subseteq I_q$, because $I_q \in \mathcal{C}$ and $q \in I_q$. As $\mathcal{C}$ is weakly intersecting, C_q is cycle critical. Moreover, $q \in C_q$, so by minimality $I_q \subseteq C_q$. Thus $C_q = I_q$. $\square$

Lemma 6.3.4. *Given $\mathcal{J}$, $\text{minl}(\mathcal{C})$ can be found in $O(n^2)$ time.*

Proof. By Lemma 6.3.2, $\mathcal{C}$ is weakly intersecting, so Lemma 6.2.4 and 6.3.3 imply that $\mathcal{J}$ is sub-intersecting and that $\text{minl}(\mathcal{C}) = \text{minl}(\mathcal{J})$. By Lemma 6.2.7 $\text{minl}(\mathcal{J})$ can be found in $O(n^2)$ time, as $|\mathcal{J}| = O(n)$. $\square$

6.3.3 Checking conditions (U3) and (U4)

Condition (U4) holds if and only if $|\text{minl}(\mathcal{C})| \leq 1$ by condition (U6). An augmenting path can be found in $O(|E_f|) = O(2|E_D|) = O(n)$ time. As in Section 6.1, we repeat this for every $q \in [c]$, so $\mathcal{J}$ can be found in $O(n^2)$ time. Given $\mathcal{J}$, condition (U3) can be checked in $O(n^2)$ time by Lemma 6.3.4. Suppose $\text{minl}(\mathcal{J}) = \{I\}$, that is, that condition (U4) holds. If $\Delta_{3,I} \geq 2$, then condition (U3) holds. So all that is left to check is that there are at least three vertices $v \in S(I)$ such that $d_{v,I} \geq 2$. This can be done in $O(n^2)$ time. So conditions (U3) and (U4) can be checked in $O(n^2)$ time.

So we can check whether conditions (U1)-(U4) holds in $O(n^2)$ time, and if they do not hold, we find a violated constraint with a witness.

6.4 Checking other conditions

Suppose the unicycle conditions hold for demand matrix D .

6.4.1 Finding a directly forced triangle

Suppose the unicycle conditions hold for D . If there is a directly forced triangle in I , then I is a minimal cycle critical set. Only if $|I| \leq 3$ can there be a directly forced triangle in I . If so, we find $d_{v,I}$ for all $v \in S(I)$ and find $s(I)$ in time $O(n)$. While we calculate $d_{v,I}$, we keep track of the vertices v such that $d_{v,I} > 1$. There can only be a directly forced triangle in I , if there are exactly three such vertices. If their sum is equal to $s(I) + 3$, there is a directly forced triangle in I . Otherwise no directly forced triangle exists. So, given the minimal cycle critical sets, all directly forced triangles can be found in $O(n)$ time per minimal cycle critical set, so $O(n^2)$ time overall.

6.4.2 Checking the conditions from Theorem 4.2.3

If no $q \in [c]$ is in a cycle critical set, the conditions from Theorem 4.2.3 hold, that is, if $I_q = \emptyset$ for all $q \in [c]$. So the conditions from Theorem 4.2.3 can be checked in $O(n^2)$ time.

6.4.3 Finding tree critical sets

One can check for tree critical sets in a similar manner, by setting $c(s, q) = \frac{t(q)}{2} + 2$ for some $q \in [c]$. If an augmenting path starting with $\vec{s}q$ can be found, let g be flow f augmented along this path. Then q is in no tree critical set, as for all $I \subseteq [c]$ with $q \in I$

$$\begin{aligned} 2 + \frac{t(I)}{2} &= g^-(I) = g^+(I) \\ &\leq g^-(\Gamma_G^+(I)) = g^+(\Gamma_G^+(I)) \\ &\leq c(\Gamma_G^+(I), t) = |\Gamma_G^+(I)| = s(I) \end{aligned}$$

and so $t(I) \leq 2s(I) - 4$. Let J_q be empty if q is in no tree critical sets.

If there is no augmenting path, let J_q be the vertices visited on the search for an augmented path starting with sq . Then J_q is the minimal tree critical set containing q or empty if q is in no tree critical set, as in the proof of Lemma 6.3.1. This takes $O(n^2)$ time overall.

6.4.4 Finding a directly forced edge

Each minimal tree critical set J_q containing q contains a potential directly forced edge if $|J_q| = 1$. If $|J_q| = 1$, find $s(q)$, the largest and the second largest demand in q in $O(n)$ time. If the sum of the largest and second largest demand is equal to $s(q)$, there is a directly forced edge in q . We can check this for each J_q in $O(n)$ time, so $O(n^2)$ time overall.

6.4.5 Checking the conditions from Theorem 5.7.3

Conditions (M1)-(M3) are as (U1)-(U3), so only condition (M4) needs to be checked. Condition (M4) holds if and only if $|\minl(\mathcal{A})| \leq k$. This can be checked in $O(n^2)$ time by Lemma 6.3.4.

6.5 Finding realisations

6.5.1 Finding a forest realisation

Suppose we checked the conditions from Theorem 4.2.3. We find a realisation recursively, by following the proof of Theorem 4.2.3.

6.5.1.1 Algorithm pseudocode

Input: demand matrix $D = (d_{v,q} : v \in [n], q \in [c])$, for which the conditions of Theorem 4.2.3 are checked and hold.

Output: A forest realisation of D .

- For each $q \in [c]$, let I_q be the minimal tree critical set containing q , or empty otherwise.
- If there is non-empty $I = I_{q_0} \subsetneq [c]$:
 - Set $I^c = [c] \setminus I$, $V = \{[n] \setminus S_D(I)\}$, $D_1 = (d_{v,q} : v \in S(I), q \in I)$ and let a be a new vertex with demands $d_{a,q} = \sum_{v \in S(I)} d_{v,q}$, for each $q \in I^c$.
 - Run this algorithm on D_1 and obtain realisation T_1 .
 - Set $D_2 = (d_{v,q} : v \in V \cup \{a\}, q \in I^c)$.
 - Run this algorithm on D_2 and obtain realisation T_2 .
 - Combine realisation T_1 and T_2 and return.
- Else:
 - Pre-process all total degrees.

- Find a leaf v with demand for $t \in [c]$ say.
- If $d_{w,[c]} = 1$ for all $w \in S(t)$:
 - * Run this algorithm on $D' = (d_{v,q} : q \in [c] \setminus t, v \in [n] \setminus S(t))$ and obtain realisation T' .
 - * Return T' with $s(t)$ disjoint edges in $S(t)$.
- Else:
 - * Pick any $w \in S(t) \setminus v$ for which $d_{w,[c]} > 1$.
 - * Let $\tilde{D}(w)$ be obtained from D by removing v and reducing $d_{w,t}$ by one.
 - * Run this algorithm on $\tilde{D}(w)$ and obtain realisation $\tilde{T}$.
 - * Append v to w with colour t in $\tilde{T}$, return.

The depth of the recursive tree is at most $O(n)$ and each step takes at most $O(n^2)$ time, thus this is a $O(n^3)$ time algorithm.

6.5.2 Finding a unicyclic realisation

Suppose we checked the conditions from Theorem 5.2.3. We find a realisation recursively, by following the proof of Theorem 5.2.3 and using the forest algorithm.

6.5.2.1 Algorithm pseudocode

Input: demand matrix $D = (d_{v,q} : v \in [n], q \in [c])$, for which the conditions of Theorem 5.2.3 are checked.

Output: A realisation of D with at most one cycle.

- Find the unique minimal cycle critical set I , or $\emptyset$ if there is none.
- If $I = \emptyset$:
 - Find tree realisation T and return T .
- Else if $I \subsetneq [c]$:
 - Set $I^c = [c] \setminus I$, $V = \{[n] \setminus S_D(I)\}$, $D_1 = (d_{v,q} : v \in S(I), q \in I)$ and let a be a new vertex with demands $d_{a,q} = \sum_{v \in S(I)} d_{v,q}$, for each $q \in I^c$.
 - Run this algorithm on D_1 and obtain realisation C_1 .
 - Set $D_2 = (d_{v,q} : v \in V \cup \{a\}, q \in I^c)$.
 - Find tree realisation T_2 of D_2 .
 - Combine realisation C_1 and T_2 and return.
- Else: Pre-process all total degrees.
- If $d_{v,[c]} = 2$ for all $v \in [n]$:

- Return a cycle through all vertices.
- Else if $\Delta_{1,[c]} + \Delta_{2,[c]} + \Delta_{3,[c]} = n + 3$:
 - Return a directly forced triangle in $[c]$.
- Else:
 - Find a leaf v with demand for t say.
 - If there is $x \in S(t)$ with $d_{x,t} \geq 2$:
 - * Set $w = x$.
 - Else:
 - For each $x \in S(t) \setminus v$, let $\tilde{D}(x)$ be obtained from D by removing v and reducing $d_{x,t}$ by one.
 - If $s(t) = 4$:
 - * Set $w = x$, if x is the first vertex such that the unicycle conditions hold for $\tilde{D}(x)$.
 - Else:
 - * Pick any $w \in S(t) \setminus v$.
 - Run this algorithm on $\tilde{D}(w)$ and obtain realisation $\tilde{C}$. Append v to w with colour t in $\tilde{C}$, return.

6.5.2.2 Complexity

Lemma 6.5.1. *The algorithm finds a realisation with at most one cycle in $O(n^3)$ time.*

Proof. Suppose that $\beta, \gamma > 0$ are constants, such that a forest realisation can be found in $\leq \beta n^3$ time and the algorithm can do all steps but the recursive calls in $\leq \gamma n^2$ time. Set $\alpha = \beta + \gamma$. We will show by induction that the algorithm finishes in $\leq \alpha n^3$ time. If $n = 1, 2$, the algorithm finishes in $\leq \beta n^3 + \gamma n^2 \leq \alpha n^3$ time. Let $n \geq 3$ and suppose it is true for smaller values.

Let I be the minimal cycle critical set or $\emptyset$ if there is none. If $I = \emptyset$, the algorithm takes $\leq \beta n^3 + \gamma n^2 \leq \alpha n^3$ time. If $I = [c]$, we call the algorithm on $n - 1$ vertices. This takes time at most

$$\alpha(n-1)^3 + \gamma n^2 = \alpha n^3 - 3\alpha n(n-1) - \alpha + \gamma n^2 \leq \alpha n^3 - 2\alpha n^2 + 3\alpha n \leq \alpha n^3,$$

for $n \geq 3$. If $I \subsetneq [c]$, we split the problem in a unicyclic problem on $n_1 \geq 3$ vertices and an acyclic problem on $n_2 \geq 2$ vertices, such that $n_1 + n_2 = n + 1$. We have

$$\alpha n_1^3 + \beta n_2^3 + \gamma n^2 = \alpha(n+1-n_2)^3 + \beta n_2^3 + \gamma n^2$$

$$\leq \alpha(n-1)^3 + 8\beta + \gamma n^2,$$

because for $n-2 \geq n_2 \geq 2$ and $\alpha > \beta$, $\alpha(n+1-n_2)^3 + \beta n_2^3 + \gamma n^2$ is maximal when n_2 is minimal. Then we obtain

$$\begin{aligned} &= \alpha n^3 - 3\alpha n(n-1) - \alpha + 8\beta + \gamma n^2 \\ &\leq \alpha n^3, \end{aligned}$$

and the last inequality holds because

$$\begin{aligned} 3\alpha n(n-1) + \alpha - 8\beta - \gamma n^2 &\geq 2\alpha n^2 + \alpha - 8\beta - \gamma n^2 \quad \text{since } n-1 \geq \frac{2}{3}n \\ &= (2\beta + \gamma)n^2 + \alpha - 8\beta \\ &> 0. \end{aligned}$$

Thus a realisation with at most one cycle can be found in $O(n^3)$ time. $\square$

6.5.2.3 Global algorithm for graphs with at most one cycle

Input: demand matrix $D = (d_{v,q} : v \in [n], q \in [c])$.

Output: A realisation of D with at most one cycle, or a violating set if no such realisation exists.

- Check the unicycle conditions for D .
- If the unicycle conditions hold:
 - Return realisation.
- Else:
 - Return a violating set.

The unicycle conditions can be checked in $O(n^2)$ time. A realisation can be found in $O(n^3)$ time, so this algorithm finishes in $O(n^3)$ time.

6.5.3 Finding a unicyclic multi-graph realisation

Exactly as simple graphs, except condition (U3) is now relaxed. The proof of Theorem 5.6.3 and 5.6.4 outline conditions for when a self-loop or a multi-edge need to be created. Otherwise the algorithm for simple graphs with at most one cycle can be used.

Relaxing the condition does not change the complexity of the algorithms, so the conditions can still be checked in $O(n^2)$ time and a realisation can be found in $O(n^3)$

Task	Time	Place
Check (U2)	$O(n^2)$	Section 6.1
Find minimal sets	$O(m \log m + mn)$	Lemma 6.2.7
Find minimal cycle critical sets	$O(n^2)$	Lemma 6.3.4
Check (U3) and (U4)	$O(n^2)$	Subsection 6.3.3
Find a directly forced triangle	$O(n^2)$	Subsection 6.4.1
Check conditions Theorem 4.2.3	$O(n^2)$	Subsection 6.4.2
Find minimal tree critical sets	$O(n^2)$	Subsection 6.4.2
Find a directly forced edge	$O(n^2)$	Subsection 6.4.4
Check conditions Theorem 5.7.3	$O(n^2)$	Section 6.4.5

Table 6.1: An overview of the different algorithms in this chapter.

time.

6.5.4 Finding a pseudoforest realisation

Suppose we checked the conditions from Theorem 5.7.3. We find a realisation recursively, by following the proof of Theorem 5.7.3 and using the forest and unicyclic realisation algorithms. This gives an $O(n^3)$ time algorithm.

6.6 Conclusion

In Table 6.1 is an overview of the algorithms discussed in this chapter, the corresponding time complexity and section or lemma where this was discussed.

These are the tools we use to show that given a demand matrix D , we can find a realisation of a certain type in $O(n^3)$ time, if one exists and return a violating set otherwise. The types of realisations we can handle are

- forests,
- graphs with at most one cycle,
- pseudoforests with at most k cyclic components, and
- unicyclic multi-graphs.

This concludes the discussion of colour degree matrices. In the next part of this thesis possible models for the reverse radar problem are discussed.

Part II

Modelling the reverse radar problem

7

The coloured L -model

In this chapter the coloured L -model is introduced. This is a graphical model designed to represent an idealised version of the reverse radar problem. After a short discussion of the complexity of the coloured L -model, we focus our attentions on a specific coloured L -model: the coloured $\{K_1, K_2\}$ -model. This chapter is all original work.

7.1 The coloured L -model

We consider a graph $G = (V, E)$, where each vertex represents a geographical location. We imagine a single sensor observing radar pulses at different locations in V at a single point in time. Algorithms to map radar pulses to pulse trains and ultimately to radar models are in place [1, 2]. So the problem input consists of radar models in different locations. We call these observed radar models observations. We assume that the input to the problem is clean, i.e. only those radar models of interest are part of the input. For instance, when military vessels are the objects of interest, the radars of commercial vessels can all be filtered out beforehand. The problem of filtering out irrelevant observations will not be addressed in this thesis.

7.1.1 The coloured L -model

Let R be a finite non-empty set. We call R the set of **colours**. Each colour can be thought of as representing a radar model and fix $R = \{\text{red, blue, green, } \dots\} =$

$\{\textcolor{red}{r}, \textcolor{blue}{b}, \textcolor{green}{g}, \dots\} = \{a, b, \dots\}$. Let G be a graph. Each vertex can be considered as representing a location, or a bearing in the case when G is a cycle or a path.

Definition 7.1.1. A graph G is **R -multi-coloured** if each vertex $v \in V(G)$ is assigned a multiset $\text{col}_G(v)$ of colours from R . An **R -multi-coloured subgraph** H of an R -multi-coloured graph G is a subgraph of G , such that for each $v \in V(H)$, $\text{col}_H(v) \subseteq \text{col}_G(v)$.

There is an example in Fig. 7.1 on the left. The subscript of $\text{col}(v)$ will be omitted when there is no ambiguity. Moreover, from now on we simply refer to an R -multi-coloured graph, as an R -coloured graph. Let G be an R -coloured graph. For $A \subseteq V(G)$, $\text{col}(A) := \biguplus_{v \in A} \text{col}(v)$. Also let $\text{col}(G) = \text{col}(V(G))$.

Definition 7.1.2. A graph G is **2-coloured**, if G is R -coloured, where $R = \{\textcolor{red}{r}, \textcolor{blue}{b}\}$. For a 2-coloured graph G and $A \subseteq V(G)$, $r(A)$ ($b(A)$) is the multiplicity of red (blue) in $\text{col}(A)$. For a single vertex $v \in V(G)$ we simply write $r(v)$ and $b(v)$.

We consider 2-coloured graphs in Chapters 8 and 9. See Fig. 7.5 for an example.

Definition 7.1.3. Two R -coloured graphs G and H are **isomorphic** if there is a graph isomorphism ϕ between G and H and such that

$$\text{col}_G(v) = \text{col}_H(\phi(v)), \forall v \in V(G). \quad (7.1)$$

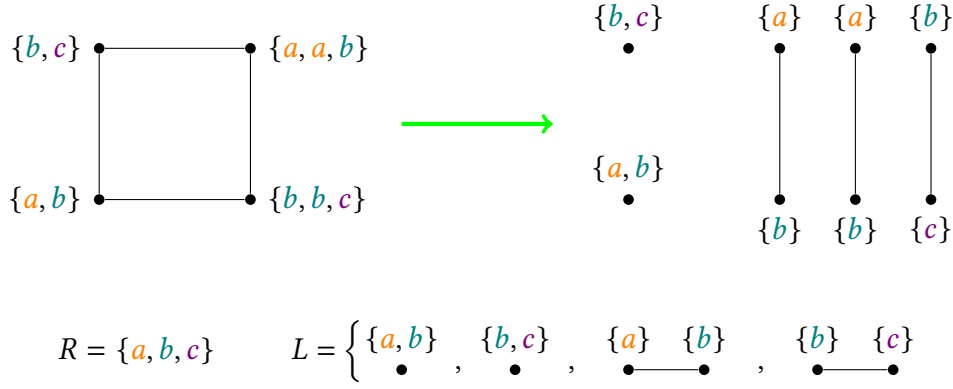
Definition 7.1.4. A **type** is an isomorphism class of R -coloured graphs. An **H -type** is the class of R -coloured graphs isomorphic to a given R -coloured graph H . A finite non-empty set of types L is a **library**. For H_i in an H -type in L we simply write $H_i \in L$.

We consider each type as representing a platform or ship. Note that types with the same colour set are generally considered to be the same type of ships, but observed in a different manner. Now fix a library L .

Definition 7.1.5. Given an R -coloured graph G , $a \in R$ and $v \in V(G)$, let $m_a(v)$ be the **multiplicity** of colour a in $\text{col}(v)$. For a set of vertices $B \subseteq V(G)$ let $m_a(B) = \sum_{v \in B} m_a(v)$.

Definition 7.1.6. An **L -decomposition** of an R -coloured graph G is a family $\mathcal{J} = \{H_1, H_2, \dots, H_k\}$ of R -coloured subgraphs of G , such that each $H_i \in L$ and for each $v \in V(G)$

$$\biguplus_{i: v \in V(H_i)} \text{col}_{H_i}(v) = \text{col}_G(v). \quad (7.2)$$

Figure 7.1: An L -decomposition of an R -coloured graph G .

Any subfamily $\mathcal{S}$ of an L -decomposition is called a **partial L -decomposition**.

$V(\mathcal{S}) = \{v : v \in V(H_i), H_i \in \mathcal{S}\}$ and for each $v \in V(\mathcal{S})$

$$\text{col}_{\mathcal{S}}(v) = \bigcup_{i: v \in V(H_i), H_i \in \mathcal{S}} \text{col}_{H_i}(v).$$

Let $\text{col}(\mathcal{J}) = \biguplus_{H_i \in \mathcal{J}} \text{col}(H_i)$. An L -decomposition with **defect** k is a partial L -decomposition $\mathcal{J}$ for which $|\text{col}(G)| - |\text{col}(\mathcal{J})| \leq k$. For $q \in R$, an L -decomposition with **q -defect** k is a partial L -decomposition $\mathcal{J}$ for which $m_q(\text{col}(G) \setminus \text{col}(\mathcal{J})) \leq k$. An L -decomposition $\mathcal{J}$ with **minimum defect** is such that for all partial L -decompositions $\mathcal{J}$, $|\text{col}(\mathcal{J})| \geq |\text{col}(\mathcal{J})|$. We also refer to this as a **maximum** L -decomposition. Similarly, an L -decomposition $\mathcal{J}$ with **minimum q -defect** is such that for all partial L -decompositions $\mathcal{J}$, $m_q(\text{col}(\mathcal{J})) \geq m_q(\text{col}(\mathcal{J}))$.

There is an example in Fig. 7.1. Note that an L -decomposition $\mathcal{J}$ allows for repetition, as $\mathcal{J}$ is a multi-set. Moreover, this model allows for subgraphs (ships) to overlap. This is a reasonable assumption, as each vertex might represent a larger geographical area. For example, if each vertex represents a bearing, platforms observed on the same vertex are not necessarily near each other, because they could be at a different distance from the sensor. From the observations we want to infer which ships are at which locations or bearings.

Without loss of generality we assume for all $H \in L$, there are no leaves with empty colour sets. In particular that means that if $|\text{col}(H)| = 1$, H is a K_1 -type and if $|\text{col}(H)| = 2$, then H is a path.

Problem 3 (L-decomposition problem (LDC)). Given a set of colours R , a set of small R -coloured graphs L and an R -coloured graph G , is there an L -decomposition of G ?

Some other things we might want to consider is if there is an L -decomposition, can we efficiently find one? Can we sample from the decompositions?

7.2 The complexity of finding L -decompositions

In this section we first show that deciding whether there is an L -decomposition in an R -coloured graph G is NP-complete, even when all ships H in L are such that $|\text{col}(H)| = 3$ and $|V(G)| = 1$ or $|R| = 1$. Next we consider what happens when the ships have size < 3 .

7.2.1 NP-complete cases

Problem 4 (Exact cover by 3-set (X3C)). *Given a finite set X with $|X| = 3q$ and a collection C of 3-element subsets of X , does C contain an exact cover for X , that is a subcollection $C' \subseteq C$ such that every element of X occurs in exactly one member of C .*

X3C is NP-complete [48]. We will show that LDC is NP-complete.

Lemma 7.2.1. *LDC is NP-complete, even when L contains only K_1 -types H such that $|\text{col}(H)| = 3$ and $G \simeq K_1$.*

Proof. It is easy to see that LDC is in NP. By reduction from X3C we will show that LDC is NP-complete. Let X be a finite set such that $|X| = 3q$ for some positive integer q and let C be a collection of 3-element subsets from X . Set $V = \{v\}$, where v is a new element, and set $R = X$. Now let L consist of a K_1 -type H with $\text{col}(H) = e$ for each $e \in C$ and let $\text{col}(v) = R$. Then $G = (V, \emptyset)$ is an R -coloured K_1 -type. We will show that G has an L -decomposition if and only if X has an exact cover.

Let $C' \subseteq C$ be an exact cover of X . Let $\mathcal{J}$ be a set which consists of a K_1 -type H with $\text{col}(H) = e$ for each $e \in C'$. Then $\mathcal{J} \subseteq L$ and by definition of an exact cover

$$\bigsqcup_{H \in \mathcal{J}} = \text{col}_H(v) = \bigcup_{H \in \mathcal{J}} \text{col}_H(v) = X = R = \text{col}_G(v).$$

So $\mathcal{J}$ is an L -decomposition of G .

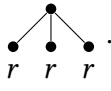
Now let $\mathcal{J}$ be an L -decomposition of G . Then

$$\bigsqcup_{H \in \mathcal{J}} \text{col}_H(v) = \text{col}_G(v) = R = X,$$

7.2. The complexity of finding L -decompositions

by definition, so if we can show that $\mathcal{J} \subseteq L$, then $C' = \{e : H \in \mathcal{J} \text{ and } \text{col}(H) = e\}$ is an exact cover of X . Suppose there is an H -type in L with $\text{col}(H) = \{x, y, z\}$, such that there are at least two H -types in $\mathcal{J}$. Then $\{x, x, y, y, z, z\} \subseteq \biguplus_{H \in \mathcal{J}} \text{col}_H(v)$, but $\{x, x, y, y, z, z\} \not\subseteq X = \text{col}_G(v)$, so $\biguplus_{H \in \mathcal{J}} \text{col}_H(v) \neq \text{col}_G(v)$, which is a contradiction. So every H -type in L appears at most once in $\mathcal{J}$ and so $\mathcal{J} \subseteq L$ as required. $\square$

Lemma 7.2.2. *LDC is NP-complete, even when $R = \{r\}$ and $L = \left\{ \begin{array}{c} \bullet \\ / \quad \backslash \\ \bullet \quad \bullet \\ | \quad | \\ r \quad r \end{array} \right\}$.*

Proof. It is easy to see that LDC is in NP. We will show by reduction from X3C that LDC is NP-complete. Let (X, C) be an instance of X3C and let V_C be a set of vertices, one for each set in C , that is, $V_C = \{v_e : e \in C\}$. Set $V = X \cup V_C$ and $E = \{v_e w : v_e \in V_C, w \in e\}$. Now let $R = \{r\}$, where r is a new element and, set $\text{col}(v) = \{r\}$ if and only $v \in X$ and let L consist of a single graph . Then $G = (V, E)$ is R -coloured

Let $\mathcal{J}$ be an L -decomposition of G and set $C' = \{e : \exists H_i \in \mathcal{J} \text{ such that } v_e \in V(H_i)\}$. We will show that C' is an exact cover of X . Let H_v be a copy of H in G , where v is such that $\text{col}(v) = \emptyset$. Then $v \in V_C$, as for each $v \in X$, $\Gamma(v) \subseteq V_C$ and for all $v_e \in V_C$, $\text{col}(v_e) = \emptyset$. So if $v \in V(H_i) \cap X$ for some $H_i \in \mathcal{J}$, $\text{col}_{H_i}(v) \neq \emptyset$, i.e. $\text{col}_{H_i}(v) = \{r\}$. Because $\mathcal{J}$ is an L -decomposition we have for all $v \in X$,

$$\biguplus_{i:v \in V(H_i)} \text{col}_{H_i}(v) = \biguplus_{i:v \in V(H_i)} \{r\} = \text{col}_G(v) = \{r\}.$$

so each $v \in X$ appears in exactly one $V(H_i)$. It is left to show that $C' \subseteq C$. Each $v_e \in V_C$ is adjacent only to $x \in X$ such that $x \in e$. Suppose that $v_e \in V(H_i) \cap V(H_j)$ for distinct $H_i, H_j \in \mathcal{J}$. Then

$$\{r, r, r\} = \text{col}(H_j) \subseteq \text{col}_G(\Gamma(v_e)) \setminus \text{col}(H_i) = \{r, r, r\} \setminus \{r, r, r\} = \emptyset,$$

which is a contradiction. So each v_e appears in at most one $H_i \in \mathcal{J}$ and therefore $C' \subseteq C$.

Now suppose C' is an exact cover of X . For $e \in C$ let H_e be a copy of H in G , with v_e as the vertex with an empty colour set. Let $\mathcal{J} = \{H_e : e \in C'\}$. Then each H_e is a subgraph of G . As C' is an exact cover, each $v \in X$ appears exactly once in some $V(H_i)$ for $H_i \in \mathcal{J}$. So for each $v \in X$ we have

$$\biguplus_{i:v \in V(H_i)} \text{col}_{H_i}(v) = \text{col}_G(v) = \{r\}.$$

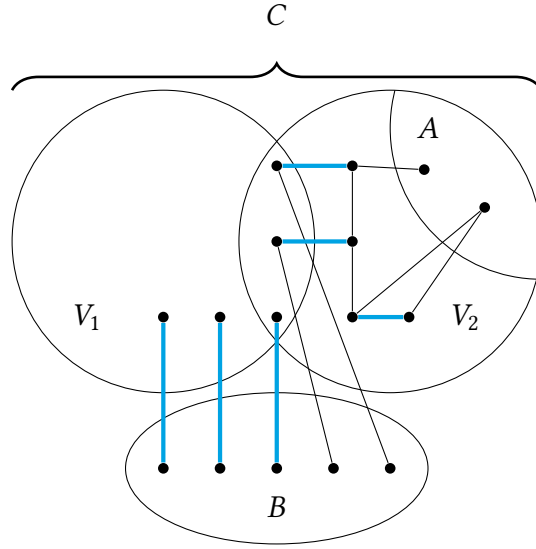


Figure 7.2: The vertex set of F .

Thus $\mathcal{J}$ is an L -decomposition of G . □

So regardless of the structure of G or the number of colours in R , LDC remains NP-complete. It is for this reason that we focus our efforts on ships with at most 2 colours for the remainder of the thesis.

7.2.2 L is such that $|\text{col}(H)| \leq 2$

If L contains only K_1 -types H such that $|\text{col}(H)| = 1$, then the problem is trivial, so it is enough to consider the case when L contains K_1 -types H such that $|\text{col}(H)| = 1$, and other types as well.

In the previous section we saw that if L contains ships with three colours, finding an L -decomposition is NP-hard, so what if L contains only ships with at most two colours?

Lemma 7.2.3. *Let G be an R -coloured graph and suppose L contains only ships with at most two colours. Set $L_i = \{H \in L : |\text{col}(H)| = i\}$ for $i = 1, 2$. If there is an L -decomposition in G with minimum defect k , then there is a pair $(\mathcal{J}_1, \mathcal{J}_2)$ such that $\mathcal{J}_1$ is a partial L_1 -decomposition, $\mathcal{J}_2$ is a maximum L_2 -decomposition and $\mathcal{J}_1 \cup \mathcal{J}_2$ is an L -decomposition with defect k .*

Proof. We will show the contra-positive. Suppose $(\mathcal{J}_1, \mathcal{J}_2)$ is a pair, such that $\mathcal{J}_1$ is a partial L_1 -decomposition, $\mathcal{J}_2$ is a maximum L_2 -decomposition, $\mathcal{J}_1 \cup \mathcal{J}_2$ is a partial L -decomposition and $\mathcal{J}_1 \cup \mathcal{J}_2$ has minimum defect over all such pairs.

7.2. The complexity of finding L -decompositions

Let G' be G with all colours covered by $\mathcal{J}_1 \cup \mathcal{J}_2$ removed. Let

$$C = \{(a, v) : v \in V(G), a \in \text{col}_G(v)\}, \text{ and}$$

$$A = \{(a, v) : v \in V(G), a \in \text{col}_{G'}(v)\},$$

so C is the multi-set of pairs consisting of a colour a at vertex v in G and $A \subseteq C$ is the multi-set of pairs that are uncovered by $\mathcal{J}_1 \cup \mathcal{J}_2$. Set

$$V_1 = \{(a, v) \in C : \exists H \in L \text{ s.t. } \text{col}(H) = \{a\}\}, \text{ and}$$

$$V_2 = \{(a, v) \in C : \exists H \in L \text{ s.t. } \{a\} \subsetneq \text{col}(H)\},$$

that is, V_i is the multi-set of colours in $\text{col}(G)$ that can be covered by a ship with i colours for $i = 1, 2$.

Let B be a copy of V_1 and let E_1 be a perfect matching between B and V_1 . Let E_2 be the set of pairs of pairs in C that can be covered by a ship with two colours, i.e.

$$E_2 = \left\{ \{(a, v), (b, w)\} : (a, v), (b, w) \in C \text{ and } \exists H \in L \text{ s.t. } \right. \\ \left. H \subseteq G, \text{col}_H(v) = \{a\}, \text{col}_H(w) = \{b\} \text{ and } \text{col}(H) = \{a, b\} \right\}.$$

Let $F = (V, E)$ be a graph where

$$V = C \cup B, \text{ as in Fig. 7.2, and}$$

$$E = E_1 \cup E_2.$$

It is easy to see that any L -decomposition corresponds to a matching in F . Let M be the matching corresponding to $\mathcal{J}_1 \cup \mathcal{J}_2$, then all unmatched pairs in F are either in A or in B . For the remainder of the proof we refer to pairs in V as elements.

We note that $A \cap V_1 = \emptyset$, or $\mathcal{J}_1$ could be extended to cover more pairs in C , contradicting maximality of $(\mathcal{J}_1, \mathcal{J}_2)$. So $A \subseteq (V_2 \setminus V_1)$, as $V_1 \cup V_2 = C$. By maximality of $\mathcal{J}_2$, there are no augmenting paths starting and ending in A . Suppose there is an augmenting path from A to B and let M' be M augmented along this path. Moreover, let $\mathcal{J}'_1$ be a partial L_1 -decomposition corresponding to $M' \cap E_1$ and similarly let $\mathcal{J}'_2$ be a partial L_2 -decomposition corresponding to $M' \cap E_2$. Then $\mathcal{J}'_1 \cup \mathcal{J}'_2$ is a partial L -decomposition, $|\text{col}(\mathcal{J}'_2)| = |\text{col}(\mathcal{J}_2)|$ (so $\mathcal{J}'_2$ is a maximum L_2 -decomposition) and $|\text{col}(\mathcal{J}'_1 \cup \mathcal{J}'_2)| > |\text{col}(\mathcal{J}_1 \cup \mathcal{J}_2)|$, contradicting our choice of $(\mathcal{J}_1, \mathcal{J}_2)$. Therefore there is no augmenting path from A to B . So if we show that an L -decomposition $\mathcal{J}$ with corresponding matching M has minimum defect if and only there are no augmenting

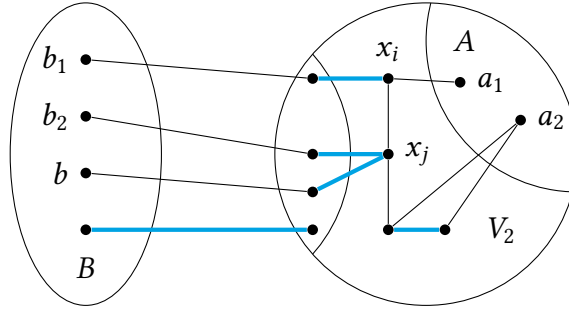


Figure 7.3: Augmenting paths in F .

paths with respect to M in F , starting and ending in A , or going between A and B , then we can conclude the proof.

Let $\mathcal{J}$ be a maximum L -decomposition with corresponding matching M and let A be the set of uncovered pairs. It is clear that if we can augment M along an augmenting path from A to A or from A to B , that we can reduce the defect, so an L -decomposition with minimum defect implies that there are no such augmenting paths in F .

Now suppose there are no augmenting paths from A to A and from A to B in F . If there are no augmenting paths from B to B , then M is maximum and so also $\mathcal{J}$. Suppose there is an augmenting path $P = b_1x_1x_2 \dots x_rb_2$ from B to B and let M' be the matching we obtain after augmenting M along P . This does not decrease the defect of $\mathcal{J}$, because this does not cover more vertices from C . Suppose there is now an augmenting path $P' = a_1y_1 \dots y_la_2$ from A to A , as displayed in Fig. 7.3. There must be an edge x_ix_j from P on P' , or P' is an augmenting path with respect to M . Without loss of generality let it be the first edge on P' they have in common. Suppose $x_ix_j \notin M'$ and that x_{i-1} and y precede x_i in P and P' respectively. Then, as P and P' are alternating, $x_{i-1}x_i, y_kx_i \in M'$, which contradicts M' being a matching. Thus $x_ix_j \in M' \setminus M$ and $P' = a_1y_1 \dots y_kx_ix_jy_r \dots y_la_2$ say. Then $a_1y_1 \dots y_kx_ix_{i-1} \dots x_1b_1$ is an augmenting path with respect to M starting in A and ending in B , which is a contradiction.

Suppose there is an augmenting path $P^* = a_1z_1 \dots z_tb$ with respect to M' from A to B , as displayed in Fig. 7.3. Without loss of generality $P^* = a_1z_1 \dots z_sx_ix_jz_r \dots z_tb$ and $x_ix_j \in M' \setminus M$. But then $a_1z_1 \dots z_sx_ix_{i-1} \dots x_1b_1$ is an augmenting path with respect to M starting in A and ending in B , which is a contradiction. So M covers a maximum number of vertices from C and thus $\mathcal{J}$ has minimum defect. This concludes the proof. $\square$

7.2. The complexity of finding L -decompositions

So there is always a maximum L_2 -decomposition which is part of a maximum L -decomposition. Next we show how to find such an L_2 -decomposition.

Definition 7.2.4. Given a graph $G = (V, E)$ with weight function $w : V \rightarrow \mathbb{R}^+, *$, a **maximum vertex-weight matching** M in G is a matching that maximizes the weight $W(M)$ of the matching, where

$$W(M) = \sum_{v \in V(M)} w(v).$$

Theorem 7.2.5 ([55]). Given a graph $G = (V, E)$ and weight function $w : V \rightarrow \mathbb{R}^+, *$, a maximum vertex-weight matching M in G is also a maximum cardinality matching.

Lemma 7.2.6. Let $F_2 = (V_2, E_2)$, where V_2 and E_s are as in the proof of Lemma 7.2.3 and let $w : V_2 \rightarrow \mathbb{R}^+$ be defined as follows:

$$w(v) = \begin{cases} 2|V_2| & \text{if } v \in V_2 \setminus V_1, \\ 1 & \text{otherwise.} \end{cases}$$

Then a maximum vertex-weight matching in F_2 corresponds to a maximum L_2 -decomposition that is part of a maximum L -decomposition.

Proof. Let M_2 be a maximum vertex-weight matching in F_2 , then M_2 is also a maximum matching by Theorem 7.2.5. As M_2 has maximum vertex-weight, M_2 covers a maximum number of vertices from $V_2 \setminus V_1$. Let $\mathcal{J}_2$ be the L_2 -decomposition corresponding to M_2 , then $\mathcal{J}_2$ is maximum. Let $\mathcal{J}_1$ be an L_1 -decomposition that covers all vertices in $(V_1 \setminus V_2) \cup ((V_1 \cap V_2) \setminus V(M))$ and let $\mathcal{J} = \mathcal{J}_1 \cup \mathcal{J}_2$. Let $\mathcal{J}'$ be an L -decomposition that covers more colours from $V_1 \cup V_2$ than $\mathcal{J}$, then $\mathcal{J}'$ covers more colours from $V_2 \setminus V_1$, as $\mathcal{J}$ covers all colours in V_1 . This contradicts vertex-weight maximality of M_2 , thus no such L -decomposition exists and so $\mathcal{J}$ is a maximum L -decomposition. $\square$

Spencer and Mayr [106] give $O(\sqrt{nm} \log n)$ algorithms to find a maximum vertex-weighted matching in a graph $G = (V, E)$, where $|V| = n$ and $|E| = m$. Halappanavar [55] gives $O(n \log n + nm)$ algorithms. Both methods make use of augmenting path methods which start with a maximum cardinality matching, so it is enough to consider libraries L_2 that consist only of ships with two colours and find a maximum L_2 -decomposition. The matching corresponding to this decomposition can be used as a starting point to find a maximum L -decomposition. We give algorithms to find L_2 -decompositions in Chapters 8, 9 and 10.

* $0 \notin \mathbb{R}^+$.

7.3 The coloured $\{K_1, K_2\}$ -model

A further simplification of the reverse radar problem is considering a single sensor intercepting pulses from all directions. Each pulse comes in on a certain bearing; the direction on the compass at which this pulse is intercepted. The precision of this bearing depends on the equipment, but we assume that radar types on different bearings are the input to the problem. We assume that there are observations on each bearing, but that the bearing might be wrongly measured by at most one; i.e. an observation either really is at its measured bearing, or at one of the two bearings directly adjacent to the measured bearing.

These simplifications remove the time dependency of the problem, make it essentially one-dimensional and dependent on one source only. In reality these simplifications may appear drastic, but the model still captures the essence of the problem, namely associating combinations of radar types with platforms.

We model these bearings by a cycle C_n , where each vertex on the cycle represents one of the n measured (groups of) bearings of the sensor, placing them in the order they would have on a compass. The coloured $\{K_1, K_2\}$ -model is a toy model trying to capture exactly this simplification, as will be explained in the following sections and chapters. The coloured $\{K_1, K_2\}$ -model allows us to link the problem to matchings and to explore some of its properties, as matchings are well-understood combinatorial objects. Though it is a simplification of reality, it throws light on the more complicated, actual problem at hand, in terms of both methods and models. Moreover, it might be possible to extend the model to be more realistic, while retaining its main properties. One such extension can be found in Chapter 11.

7.3.1 The coloured $\{K_1, K_2\}$ -model

In the coloured $\{K_1, K_2\}$ -model, we consider a specific library. Recall that K_1 is the complete graph on 1 vertex, i.e. one vertex and K_2 the complete graph on 2 vertices, i.e. an edge.

Definition 7.3.1. *In the **coloured $\{K_1, K_2\}$ -model** with a (finite non-empty) set R of colours, the library consists of a subset of all H -types such that H is either a K_1 -type with two colours from R (not necessarily distinct) or a K_2 -type, where each vertex has one colour from R . If $r = |R|$, the library consists of at most $2\binom{r}{2} + 2r = r(r-1) + 2r = r(r+1)$ types.*

7.4. The coloured $\{K_1, K_2\}$ -model and matchings

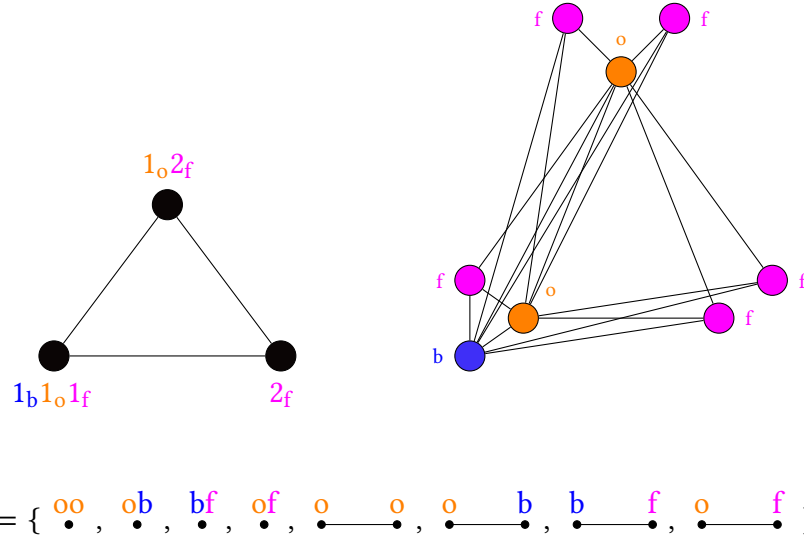


Figure 7.4: R -coloured graph G , $R = \{o, b, f\}$, and the fully separated graph G_s .

Definition 7.3.2. Let a $\{K_1, K_2\}$ -*decomposition* be an L -decomposition, where $L \subseteq \{ \overset{ab}{\bullet}, \overset{a}{\bullet} \text{---} \overset{b}{\bullet} : (a, b) \in R \times R \}$.

This corresponds to the case where each boat carries two radars. We think of a K_1 -type and a K_2 -type with the same colour set as the same type of ship, but observed in a different way.

Definition 7.3.3. An *optimal* $\{K_1, K_2\}$ -*decomposition* is a $\{K_1, K_2\}$ -decomposition, for which the number of K_2 -types used is minimum.

Since every vertex can be thought of as representing a bearing, including a K_2 -type would have a higher uncertainty level than including a K_1 -type. This is why we say the decomposition is optimal when the number of K_2 -types is minimum. Generally if $H \in L$ is a K_2 -type, then there is a K_1 -type $H' \in L$ such that $\text{col}(H) = \text{col}(H')$. The converse is not necessarily true, because some radars might be more precisely measured, and therefore never be observed at different bearings.

7.4 The coloured $\{K_1, K_2\}$ -model and matchings

The R -coloured graph can be transformed in such a way that the problem can be reduced to finding matchings[†].

[†]There is more information about matchings in Section 2.4.

Definition 7.4.1. Given R -coloured graph G and library L , let the **fully separated graph** G_s be an unweighted graph defined as follows; For each $v \in V(G)$, let $S(v)$ be a set of $|\text{col}(v)|$ elements, say $S(v) = \{v_a : a \in \text{col}(v)\}$. Let all $S(v)$ be pairwise disjoint. Let $G_s = (V_s, E_s)$ be given by

$$V_s = \bigcup_{v \in V(G)} S(v)$$

$$E_s = E_1 \cup E_2,$$

where

$$E_1 = \{v_a v_b : \begin{array}{c} ab \\ \bullet \end{array} \in L, v \in V(G)\}$$

$$E_2 = \{v_a w_b : \begin{array}{c} a \quad b \\ \bullet \text{---} \bullet \end{array} \in L, v \in V(G), w \in \Gamma(v)\}.$$

An example of a fully separated graph can be seen in Fig. 7.4.

Definition 7.4.2. Given R -coloured graph G and library L , let the **partially separated graph** G'_s be a vertex-weighted graph defined as follows; For each $v \in V(G)$, let $S'(v)$ be a set of weighted elements:

$$S'(v) = S'_1(v) \cup S'_2(v) \cup S'_3(v)$$

$$= \{v_{a_0} : m_a(v) \geq 1 \text{ and } \begin{array}{c} aa \\ \bullet \end{array} \notin L\} \cup \{v_{a_0} : m_a(v) = 1 \text{ and } \begin{array}{c} aa \\ \bullet \end{array} \in L\}$$

$$\cup \{v_{a_1}, v_{a_2} : m_a(v) \geq 2 \text{ and } \begin{array}{c} aa \\ \bullet \end{array} \in L\}.$$

Note that $S'_i(v) \cap S'_j(v) = \emptyset$ for distinct $i, j \in [3]$. Each $v_{a_0} \in S'_1(v) \cup S'_2(v)$ has weight $w(v_{a_0}) = m_a(v)$. Each pair $v_{a_1}, v_{a_2} \in S'_3(v)$ has weights $w(v_{a_1}) = \lceil \frac{m_a(v)}{2} \rceil$ and $w(v_{a_2}) = \lfloor \frac{m_a(v)}{2} \rfloor$. Let all $S'(v)$ be pairwise disjoint and let $G'_s = (V'_s, E'_s)$ be given by

$$V'_s = \bigcup_{v \in V(G)} S'(v)$$

$$E'_s = E'_1 \cup E'_2,$$

where

$$E'_1 = \{v_{a_i} v_{b_j} : \begin{array}{c} ab \\ \bullet \end{array} \in L, v \in V(G), a_i \neq b_j, i, j \in \{0, 1, 2\}\}$$

$$E'_2 = \{v_{a_i} w_{b_j} : \begin{array}{c} a \quad b \\ \bullet \text{---} \bullet \end{array} \in L, v \in V(G), w \in \Gamma(v), i, j \in \{0, 1, 2\}\}$$

See Fig. 7.5 for an example. By construction, for every $v \in V(G)$ and $a \in \text{col}(v)$,

7.4. The coloured $\{K_1, K_2\}$ -model and matchings

there is either exactly one $v_{a_0} \in S'_1(v) \cup S'_2(v)$, or exactly one pair $v_{a_1}, v_{a_2} \in S'_3(v)$. For $v \in V(G)$ and $a \in R$,

1. there is no $v_{a_i} \in S'(v)$ for $i \in \{0, 1, 2\}$, if $a \notin \text{col}(v)$,
2. there are $v_{a_1}, v_{a_2} \in S'(v)$, only if there are at least two occurrences of a in $\text{col}(v)$ and there is a K_1 -type in L with two times a , and
3. there is $v_{a_0} \in S'(v)$ otherwise.

If $a \notin \text{col}(v)$, we can disregard it. Equivalently we could create a vertex with weight zero. The pairs in $S'_3(v)$ are to accommodate K_1 -types with twice the same colours. Naturally this is only necessary if $m_a(v) \geq 2$ and there is a K_1 -type in L with twice colour a . Otherwise creating one instance is sufficient. Note that it would be equivalent to create two vertices here, but this would increase the number of vertices and edges in the graph.

The edges in E'_1 represent all possible K_1 -types, but to avoid self-loops we specify that $a_i \neq b_j$, that is $a \neq b$ and/or $i \neq j$. The edges in E'_2 represent all possible K_2 -types.

The difference with the fully separated graph is that instead of creating a vertex for every colour in the multi-set $\text{col}(v)$, we only create a vertex for every colour in the ground set of $\text{col}(v)$. If there are many occurrences of the same colour at any one vertex, this decreases the number of vertices and edges significantly.

Note that if you add the constraint that every boat carries two different radars, G_s and G'_s are $|R|$ -partite graphs, where each vertex class corresponds to a colour in R , because there are no edges between vertices of the same colour.

Definition 7.4.3. Given G and G_s (G'_s), and $A \subseteq V(G_s)$ ($A' \subseteq V(G'_s)$), let $H(A) \subseteq V(G)$ ($H'(A') \subseteq V(G)$) be the **heritage** of A (A'), given by

$$H(A) = \{v : A \text{ contains some vertex from } S(v)\}$$

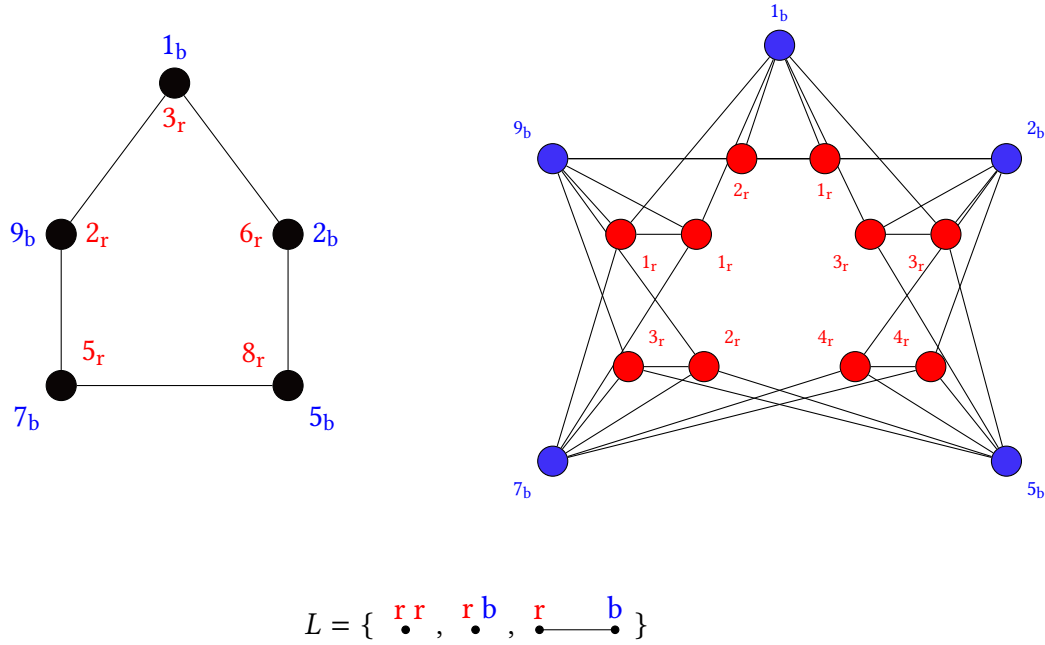
$$(H'(A') = \{v : A' \text{ contains some vertex from } S'(v)\}).$$

If $A = \{x\}$ we simply write $H(x)$.

Note that $\{H(A) : A \subseteq V(G_s)\} = \{H'(A') : A' \subseteq V(G'_s)\} = \mathcal{P}(V(G))$.

Lemma 7.4.4. Let R be a finite set of colours and $L \subseteq \{ \overset{ab}{\bullet}, \overset{a}{\bullet} \text{---} \overset{b}{\bullet} : (a, b) \in R \times R \}$. Then an R -coloured graph G has an L -decomposition if and only if G_s has a perfect matching M .

Proof. Given an L -decomposition $\mathcal{J} = \{H_i\}_i$, each H_i is either characterised by two colours and one vertex, or by an ordered pair of colours and two adjacent vertices.


 Figure 7.5: A 2-coloured graph G and the partially separated graph G'_s .

Note that if H_i and H_j are characterised by the same colours and a vertex or by the same ordered pair of colours and pair of vertices, they are indistinguishable, so it suffices to count them:

$$h(a, b, v) = |\{H_i \in \mathcal{J} : V(H_i) = \{v\} \text{ and } \text{col}_{H_i}(v) = \{a, b\}\}|$$

$$h(a, b, v, w) = |\{H_i \in \mathcal{J} : V(H_i) = \{v, w\}, \text{col}_{H_i}(v) = \{a\} \text{ and } \text{col}_{H_i}(w) = \{b\}\}|$$

Moreover G_s consists of a union of complete subgraphs $B(a, a, v)$ and complete bipartite subgraphs $B(a, b, v)$ and $B(a, b, v, w)$, defined by one colour and a vertex in G , by two colours and a vertex in G or an ordered pair of colours and two adjacent vertices in G for each $\begin{smallmatrix} a & b \\ \bullet & \bullet \end{smallmatrix} \in L$, and $\begin{smallmatrix} a & b \\ \bullet & \bullet \end{smallmatrix} \in L$ respectively. That is, if $\begin{smallmatrix} a & b \\ \bullet & \bullet \end{smallmatrix} \in L$, The set of all instances a and all instances b in $\text{col}(v)$ translates to a complete subgraph $B(a, a, v)$ in G_s if $a = b$. If $a \neq b$ it translates into a complete bipartite subgraph $B(a, b, v)$ in G_s with $m_a(v)$ vertices at one side and $m_v(b)$ vertices on the other side. Similarly for $B(a, b, v, w)$. Now for each $B(a, b, v)$, take a matching of size $h(a, b, v)$ and delete these vertices from the graph. Moreover, for each $B(a, b, v, w)$, take a matching of size $h(a, b, v, w)$ and delete these vertices from the graph. At the end the graph is empty, because for each $v \in V$

$$|S(v)| = |\text{col}(v)| = \sum_{i: V(H_i) = \{v\}} 1 + \sum_{i: v \in V(H_i)} 1$$

(count 1 for each H_i for which $v \in V(H_i)$ and 1 extra each time $V(H_i) = \{v\}$). All partial matchings combined form a perfect matching.

Conversely, given a perfect matching M on G_s , let $\phi : M \rightarrow \mathcal{J}$ be the following map on $e = xy \in M$:

$$\phi(xy) = \begin{cases} \begin{array}{c} \text{col}(x)\text{col}(y) \\ \bullet \\ v \end{array} & \text{if } H(\{x, y\}) = \{v\} \\ \begin{array}{cc} \text{col}(x) & \text{col}(y) \\ \bullet & \bullet \\ v & w \end{array} & \text{if } H(x) = \{v\} \text{ and } H(y) = \{w\}, w \neq v, \end{cases}$$

where v, w are vertices in the original graph G . This is a proper decomposition, because only legal graphs are taken in such a way that exactly all colours in the colour set are used. $\square$

Lemma 7.4.5. *Let R be a finite set of colours and $L \subseteq \{ \begin{array}{c} ab \\ \bullet \end{array}, \begin{array}{cc} a & b \\ \bullet & \bullet \end{array} : (a, b) \in P \times P \}$. Then an R -coloured graph G has an L -decomposition $\mathcal{J}$ if and only if G'_s has a perfect weighted matching[‡] $f : E'_s \rightarrow \mathbb{N}_0$.*

Proof. We note that we can turn an L -decomposition $\mathcal{J}$ in G into a perfect weighted matching f on E'_s and vice versa, by setting

$$\begin{aligned} h(a, b, v) &= f(v_a v_b) \quad \forall \begin{array}{c} ab \\ \bullet \end{array} \in L \text{ and } v \in V \\ h(a, b, v, w) &= f(v_a w_b) \quad \forall \begin{array}{cc} a & b \\ \bullet & \bullet \end{array} \in L \text{ and } v, w \in V, \end{aligned}$$

where $h(a, b, v)$ and $h(a, b, v, w)$ are as defined in the proof of Lemma 7.4.4.

Note that by construction $v_a v_b \in E'_s$ for all $\begin{array}{c} ab \\ \bullet \end{array} \in L$ and $v \in V$ for which $h(a, b, v) > 0$. Similarly for $h(a, b, v, w)$. $\square$

7.5 Outline

We discuss $\{K_1, K_2\}$ -decompositions in 2-coloured in trees and cycles in Chapter 8 and in more general graphs in Chapter 9. Algorithms for $\{K_1, K_2\}$ -decompositions with arbitrarily many colours are discussed in Chapter 10. The likelihood and two-time models are presented in Chapter 11. This part of the thesis is all original work, unless otherwise stated.

[‡]See Definition 2.4.1.

Chapter 7. The coloured L -model

Table 7.1 shows an overview of the complexity of finding a $\{K_1, K_2\}$ -decomposition in different types of 2-coloured and general coloured graphs. The section where the algorithm is explained is given in parentheses.

Find a $\{K_1, K_2\}$ - decomposition in $G = (V, E)$	2 colours			>2 colours		
	Time complexity per node		Overall space complexity	Time complexity per node		Overall space complexity
	Perfect	Defect $\leq k$	Perfect Defect $\leq k$	Perfect	Defect $\leq k$	Perfect Defect $\leq k$
G is a tree	$O(n)$ overall (8.2)	$O(n)$ (8.2)	$O(n)$ (8.2)	$O(4^{2d})^a$	$O(20^{2d}k)^b$	$O(2^{2d})^a$ $O(20^{2d}k)^b$
G is a cycle	$O(dn)$ overall (8.3)	$O(dn)$ (8.3)	$O(n)$ (8.3)	$O(4^{3d})^c$	$O(20^{3d}k)^d$	$O(2^{3d})^c$ $O(20^{3d}k)^d$
G_s has pathwidth $\leq p-1$	$O(4^p m)$ (9.6)	$O(4^p m)$ (9.6)	$O(m)$ (9.6)	$O(4^p)$ (9.6)	$O(20^p k)$ (10.4.2)	$O(2^{2p})$ (10.3.3) $O(20^{2p}k)$ (10.4.2)
G_s has treewidth $\leq p-1$	$O(8^p m)$ (9.6)	$O(8^p m)$ (9.6)	$O(m)$ (9.6)	$O(4^p)$ (10.3.3)	$O(20^p k)$ (10.4.2)	$O(4^p)$ (10.3.3) $O(20^{2p}k)$ (10.4.2)
G'_s has pathwidth $\leq p-1$	$O(4^p m \log d)$ (9.6)	$O(4^p m \log d)$ (9.6)	$O(m)$ (9.6)	$O(4^{dp})^e$	$O(20^{dp} k)^f$	$O(4^{dp})^e$ $O(20^{dp} k)^f$
G'_s has treewidth $\leq p-1$	$O(8^p m \log d)$ (9.6)	$O(8^p m \log d)$ (9.6)	$O(m)$ (9.6)	$O(4^{dp})^e$	$O(20^{dp} k)^f$	$O(4^{dp})^e$ $O(20^{dp} k)^f$

Table 7.1: Overview of the complexity of finding a $\{K_1, K_2\}$ -decomposition in different R -coloured graphs $G = (V, E)$, where G_s is the fully separated graph, G'_s the partially separated graph and $d = \max_{v \in V} |\text{col}(v)|$, $n = |V|$, and $m = |E|$.

^a G_s has treewidth at most $2d-1$, then as in Section 10.3.3.

^b G_s has treewidth at most $2d-1$, then as in Section 10.4.2.

^c G_s has treewidth at most $3d-1$, then as in Section 10.3.3.

^d G_s has treewidth at most $3d-1$, then as in Section 10.4.2.

^e G_s has treewidth at most $dp-1$, then as in Section 10.3.3.

^f G_s has treewidth at most $dp-1$, then as in Section 10.4.2.

8

Decompositions in 2-coloured trees and cycles

In this chapter it will be shown that $\{K_1, K_2\}$ -decompositions in 2-coloured trees and cycles can be found in linear time. We fix $R = \{\text{red}, \text{blue}\} = \{\textcolor{red}{r}, \textcolor{blue}{b}\}$ and library $L = \{ \begin{smallmatrix} \textcolor{red}{r} \textcolor{blue}{b} \\ \bullet \end{smallmatrix}, \begin{smallmatrix} \textcolor{red}{r} & \textcolor{blue}{b} \\ \bullet & \text{---} \bullet \end{smallmatrix} \}$.

8.1 Definitions

Definition 8.1.1. Given a 2-coloured graph G and a $\{K_1, K_2\}$ -decomposition $\mathcal{J}$, we will say $G_i \in \mathcal{J}$ is **along an edge** $uv \in E(G)$ if G_i is a K_2 -type and $V(G_i) = \{u, v\}$. A pair of graphs $G_1, G_2 \in \mathcal{J}$ along edge $uv \in E(G)$ is **opposite**, if $\text{col}_{G_1}(v) = \{\textcolor{red}{r}\} = \text{col}_{G_2}(u)$ and $\text{col}_{G_1}(u) = \{\textcolor{blue}{b}\} = \text{col}_{G_2}(v)$, or vice versa.

Lemma 8.1.2. Given a 2-coloured graph G with a $\{K_1, K_2\}$ -decomposition $\mathcal{J}$, then G has a $\{K_1, K_2\}$ -decomposition $\mathcal{J}$, such that for each edge $uv \in E(G)$ there are no opposite pairs of graphs along uv . Furthermore, every optimal $\{K_1, K_2\}$ -decomposition has this property.

Proof. Suppose that $\mathcal{J}$ is $\{K_1, K_2\}$ -decomposition of G with a minimal total number of opposite pairs. Without loss of generality, suppose there is an opposite pair $G_1, G_2 \in \mathcal{J}$ along $uv \in E(G)$. Then G_1 and G_2 can be replaced by a K_1 -type at u and a K_1 -type at v , contradicting minimality.

Because an opposite pair can be replaced by two K_1 -types, an optimal $\{K_1, K_2\}$ -decomposition cannot have opposite pairs, or else the number of K_2 -types could be reduced. $\square$

8.2 Decomposition in trees

In this section a decomposition algorithm for a 2-coloured tree is presented. The algorithm returns an optimal decomposition with minimal defect. Note that this algorithm only works when there are exactly two colours in total and each platform carries exactly two distinct radars.

First an algorithm sketch is given, after which details of an implementation are discussed. Finally, it is shown that the algorithm is correct.

8.2.1 Algorithm sketch

Given a 2-coloured tree T , the algorithm returns an optimal $\{K_1, K_2\}$ -decomposition with minimal defect k . If T is empty, it returns $\emptyset$. If $T = \{v\}$ it returns $\min(r(v), b(v))$ K_1 -types at v and defect $|r(v) - b(v)|$. Otherwise the algorithm takes $\min(r(v), b(v))$ K_1 -types at leaf v with neighbour w and if $r(v) \geq b(v)$ ($b(v) > r(v)$), $\min(|r(v) - b(v)|, b(w))$ ($\min(|r(v) - b(v)|, r(w))$) K_2 -types along vw . It calls itself with v removed and the used colours removed from $\text{col}(w)$, to obtain a partial decomposition $\mathcal{J}'$ with defect k' . The algorithm returns this decomposition with the previously formed K_1 -types and K_2 -types added and with defect k' plus the number of unused colours at v .

8.2.2 Details of implementation

Assume the colour set $\text{col}(v)$ of a vertex v is stored as the numbers $r(v)$ and $b(v)$, such that $|\text{col}(v)| = r(v) + b(v)$.

Input: 2-coloured tree T with $\text{col}(T)$ and list of leaves $L(T)$.

Output: An optimal $\{K_1, K_2\}$ -decomposition with minimal defect k .

- If $T = \emptyset$: return $\emptyset$ and defect 0.
- If $V(T) = \{v\}$:
 - Return $\min(r(v), b(v))$ K_1 -types at v and defect $|r(v) - b(v)|$.
- Take a leaf $v \in L(T)$ with neighbour w .

- If w has only one child: $L(T') = w \cup L(T) \setminus v$.
- Else: $L(T') = L(T) \setminus v$.
- If $r(v) \geq b(v)$:
 - Set $b'(w) = b(w) - r(v) + b(v)$.
 - Call this algorithm on $T' = T \setminus v$ with $b(w) = \max(b'(w), 0)$ in $\text{col}(T')$, and $L(T')$ to obtain decomposition $\mathcal{J}$ with defect k' .
 - Return decomposition with $b(v)$ K_1 -types at v , $\min(r(v) - b(v), b(w))$ K_2 -types along vw with red at v and blue at w and defect $k' + \max(-b'(w), 0)$.
- Else:
 - Set $r'(w) = r(w) - b(v) + r(v)$.
 - Call this algorithm on $T' = T \setminus v$ with $r(w) = \max(r'(w), 0)$ in $\text{col}(T')$, and $L(T')$ to obtain decomposition $\mathcal{J}$ with defect k' .
 - Return decomposition with $r(v)$ K_1 -types at v , $\min(b(v) - r(v), r(w))$ K_2 -types along vw with blue at v and red at w and defect $k' + \max(-r'(w), 0)$.

Note that an initial list of leaves can easily be constructed in $O(n)$ time by a depth first search. The depth of the recursive tree is at most n , so this gives an $O(n)$ time complexity, assuming that arithmetic operations such as subtractions and comparisons with 0, take unit time. The space complexity is $O(n)$.

8.2.3 Correctness

The following technical lemmas are needed to show that the algorithm always find an optimal decomposition with minimal defect.

Definition 8.2.1. Let G be a 2-coloured graph, $v \in V(G)$, and library $L = \{ \overset{rb}{\bullet}, \overset{r}{\bullet} \text{---} \overset{b}{\bullet} \}$. Fix $a, b \in \mathbb{N}_0$. For $i \in \mathbb{N}_0$ and $v \in V(G)$ suppose $(a+i)r + (b+i)b \subseteq \text{col}(v)$; $G(i, v)$ is G with colour set $\text{col}(G(i, v))$:

$$\begin{aligned} \text{col}_{G(i,v)}(w) &= \text{col}_G(w) \quad \text{for } w \neq v, \text{ and} \\ \text{col}_{G(i,v)}(v) &= \text{col}_G(v) - (a+i)r - (b+i)b. \end{aligned}$$

Lemma 8.2.2. Let v be a leaf in G . If $G(i, v)$ has a $\{K_1, K_2\}$ -decomposition with defect at most k , then $G(0, v)$ has $\{K_1, K_2\}$ -decomposition with defect at most k .

Proof. Suppose $G(i, v)$ has a decomposition $\mathcal{J}$ with defect at most k , then this extends to a decomposition in $G(0, v)$ by adding i K_1 -types at v . This does not affect the defect of the decomposition. $\square$

Lemma 8.2.3. *Let G be a 2-coloured graph G and v a leaf adjacent to w . Set $x = |r(v) - b(v)|$. Suppose $r(v) \geq b(v)$. G has a $\{K_1, K_2\}$ -decomposition with defect at most k if and only if $x \leq b(w) + k$ and $G' = G \setminus v$ has a $\{K_1, K_2\}$ -decomposition with defect at most $k' = k - \max(x - b(w), 0)$, where $\text{col}(G')$ is obtained from $\text{col}(G)$ by setting*

- $b_{G'}(w) = \max(b_G(w) - x, 0)$,
- $r_{G'}(w) = r_G(w)$, and
- $\text{col}_{G'}(u) = \text{col}_G(u)$ for all $u \in V(G) \setminus \{v, w\}$.

Proof. If G' has a decomposition with defect at most k' , this extends to G .

Now suppose that G has a $\{K_1, K_2\}$ -decomposition with defect at most k , then by Lemma 8.1.2 there is a $\{K_1, K_2\}$ -decomposition $\mathcal{C}$ with defect at most k without opposite K_2 -types along vw . By Lemma 8.2.2 and because $r(v) \geq b(v)$, $\mathcal{C}$ has $b(v)$ K_1 -types at v and $\min(b(w), x)$ K_2 -types along vw with $\textcolor{red}{r}$ at v and $\textcolor{blue}{b}$ at w . Thus $x \leq b(w) + k$ and removing all $H_i \in \mathcal{C}$ such that $v \in V(H_i)$ constitutes a $\{K_1, K_2\}$ -decomposition of G' with defect at most k' . $\square$

By symmetry red and blue can be swapped:

Lemma 8.2.4. *Let G be a 2-coloured graph G and v a leaf adjacent to w . Set $x = |r(v) - b(v)|$. Suppose $b(v) \geq r(v)$. G has a $\{K_1, K_2\}$ -decomposition with defect at most k if and only if $x \leq r(w) + k$ and $G' = G \setminus v$ has a $\{K_1, K_2\}$ -decomposition with defect at most $k' = k - \max(x - r(w), 0)$, where $\text{col}(G')$ is obtained from $\text{col}(G)$ by setting*

- $r_{G'}(w) = \max(r_G(w) - x, 0)$,
- $b_{G'}(w) = b_G(w)$, and
- $\text{col}_{G'}(u) = \text{col}_G(u)$ for all $u \in V(G) \setminus \{v, w\}$.

Now we have everything we need to show that the algorithm is correct.

Lemma 8.2.5. *The above described algorithm returns an optimal $\{K_1, K_2\}$ -decomposition with minimal defect in $O(n)$ time.*

Proof. By induction on $n = |T|$. If $T = \emptyset$ or $T = \{v\}$ the statement holds. Now let $n > 1$ and suppose the statement is true for smaller values. Let v be a leaf and w its neighbour. Without loss of generality $r(v) \geq b(v)$. By Lemma 8.2.2, we may take $b(v)$ K_1 -types at v , leaving an excess of $x = r(v) - b(v)$ $\textcolor{red}{r}$ in $\text{col}(v)$. The algorithm finds an optimal decomposition $\mathcal{J}$ with defect k' in T' (as defined in algorithm) and

returns $\mathcal{J}$ with $b(v)$ K_1 -types at v and $\min(x, b(w))$ K_2 -types along vw with red at v and blue at w with defect

$$k = k' + \max(-b'(w), 0) = k' + \max(r(v) - b(v) - b(w), 0) = k' + \max(x - b(w), 0).$$

By Lemma 8.2.3 there is a $\{K_1, K_2\}$ -decomposition with defect at most k if and only if $x \leq b(w) + K$ and T' has a $\{K_1, K_2\}$ -decomposition with defect at most $k' = k - \max(x - b(w), 0)$. So the algorithm finds a decomposition with defect at most k if and only if such a decomposition exists. By Lemma 8.1.2 this decomposition is optimal, as $\mathcal{J}$ is optimal in T' and every other decomposition would have opposite K_2 -types along vw .

Finally, the algorithm takes $O(n)$ time, because a leaf is removed in constant time on each recursive call. $\square$

8.3 Decomposition in cycles

In this section a decomposition algorithm for a 2-coloured cycle C is considered. First we will discuss how to break cycles, after which an algorithm sketch is given. Next we provide the details of an implementation and we end this section by showing that this algorithm is correct.

8.3.1 Breaking cycles

If a cycle can be broken (i.e. one or more edges or vertices removed), it can be treated as a (collection of) tree(s). If there is a vertex with an empty colour set, this vertex can be removed. If there are two adjacent vertices u and v that both only have red or both only have blue, the edge uv can be removed and the remaining tree can be considered. If there is no such vertex or edge, the problem is more complicated, because a cycle does not have leaves.

One might think that a cycle can be broken by removing the edge between two vertices that have an excess of the same colour. Consider the 2-coloured C_6 in Fig. 8.1. Obviously the edge v_1v_6 has the largest excess of red. However, taking away that edge, leaves no $\textcolor{blue}{b}$ to match the $\textcolor{red}{r}$ of v_2 . If the edge is left in on the other hand, a $\{K_1, K_2\}$ -decomposition can easily be found.

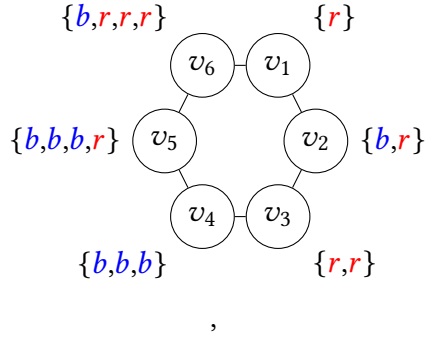


Figure 8.1: Counterexample: the edge between v_1 and v_6 cannot be removed.

8.3.2 Algorithm Sketch

Let C be a 2-coloured cycle. Fix an edge $uv \in E(C)$. Let a type 1 K_2 -type be a K_2 -type along uv with blue at u and red at v and a type 2 K_2 -type be a K_2 -type with red at u and blue at v . The algorithm checks for all possible numbers of type 1 and all possible numbers of type 2 K_2 -types, by calling the tree algorithm on $C \setminus uv$. It returns the decomposition with minimal defect.

8.3.3 Details of implementation

Input: 2-coloured cycle C with $\text{col}(C)$.

Output: An optimal $\{K_1, K_2\}$ -decomposition with minimal defect k .

- Pick an edge $uv \in E(C)$.
- Let
 - $r(vu) = \min(r(v), b(u))$,
 - $b(vu) = \min(b(v), r(u))$,
 - $D = \text{decomposition} = \emptyset$,
 - $k = \text{min defect} = n = \text{number of vertices}$,
 - $s = \text{number of } K_2\text{-types} = 0$, and
 - $t = \text{type of } K_2\text{-types} = 1$.
- For $i \in [0, r(vu)]$:
 - Run tree algorithm on $C' = C \setminus uv$ with $r'(v) = r(v) - i$ and $b'(u) = b(u) - i$ in $\text{col}(C)$ and $L(C') = \{u, v\}$.
 - If $\text{defect} < k$: let $D = \text{decomposition}$ and $k = \text{defect}$ and $s = i$.
 - If $\text{defect} = k$ and $i < s$: $D = \text{decomposition}$ and $s = i$.
 - $i = i + 1$.

- For $j \in [0, b(vu)]$:
 - Run tree algorithm on $C' = C \setminus uv$ with $b'(v) = b(v) - j$ and $r'(u) = r(u) - j$ in $\text{col}(C)$ and $L(C') = \{u, v\}$.
 - If $\text{defect} < k$: let $D = \text{decomposition}$, $k = \text{defect}$, $s = j$ and $t = 2$.
 - If $\text{defect} = k$ and $j < s$: $D = \text{decomposition}$, $s = j$ and $t = 2$.
 - $j = j + 1$.
- Return decomposition D with s K_2 -types of types t along vu and defect k .

8.3.4 Correctness

Lemma 8.3.1. *The above described algorithm returns a $\{K_1, K_2\}$ -decomposition with minimal defect (and a minimal number of K_2 -types along uv) in $O(dn)$ time, where $d = \max_{v \in V} |\text{col}(v)|$ and space complexity $O(n)$.*

Proof. By Lemma 8.1.2 and 8.2.5, every time the tree algorithm is called, it returns a decomposition with minimal defect. The current best decomposition is only updated if the total defect decreases, or the total stays the same, but the number of K_2 -types along uv decreases. The algorithm calls the tree algorithm $O(d)$ times, giving an overall time complexity of $O(dn)$. Only 2 decompositions are stored at any one time, giving a space complexity of $O(n)$. $\square$

8.4 Conclusion

Table 9.1 displays the different decomposition algorithms in this chapter and the next chapter. In Chapter 9 $\{K_1, K_2\}$ -decomposition algorithms for more general 2-coloured graphs are discussed. Chapter 10 discusses results for the general coloured $\{K_1, K_2\}$ -model. Chapter 11 introduces two new models: the likelihood model and the two-time model.

9

The red-blue $\{K_1, K_2\}$ - model

In this chapter we restrict the $\{K_1, K_2\}$ -model to exactly two colours in total (red and blue by default) and stipulate that each type must have exactly two different colours: we fix $R = \{\text{red}, \text{blue}\} = \{\mathbf{r}, \mathbf{b}\}$ and library $L = \{ \overset{\mathbf{rb}}{\bullet}, \overset{\mathbf{r}}{\bullet} \text{---} \overset{\mathbf{b}}{\bullet} \}$. That is, we only consider 2-coloured graphs. Fig. 9.1 shows an example of a 2-coloured graph. All work in this chapter is original, unless otherwise stated.

We start by showing a weighted version of Hall's condition (Theorem 2.4.5). As a $\{K_1, K_2\}$ -decomposition is equivalent to a matching in the partially separated graph G'_s , the weighted Hall's condition implies a simple closure condition for the existence of a $\{K_1, K_2\}$ -decomposition. In Section 9.2 we give defect closure conditions, analogue to the defect version of Hall's theorem (Corollary 2.4.6). Next we show that the partially separated graph G'_s of a 2-coloured cycle C_n has bounded path- and treewidth. Checking for the existence of a $\{K_1, K_2\}$ -decomposition in a 2-coloured graph is thus equivalent to checking weighted Hall's condition in a vertex-weighted bipartite graph with bounded pathwidth.

In Section 9.4 an algorithm to check weighted Hall's condition in weighted bipartite graphs with bounded pathwidth is given. Similarly in Section 9.5 we describe an algorithm for weighted bipartite graphs with bounded treewidth. Finally in Section 9.6, we sketch a method for finding a weighted matching, if an algorithm which checks for existence is in place.

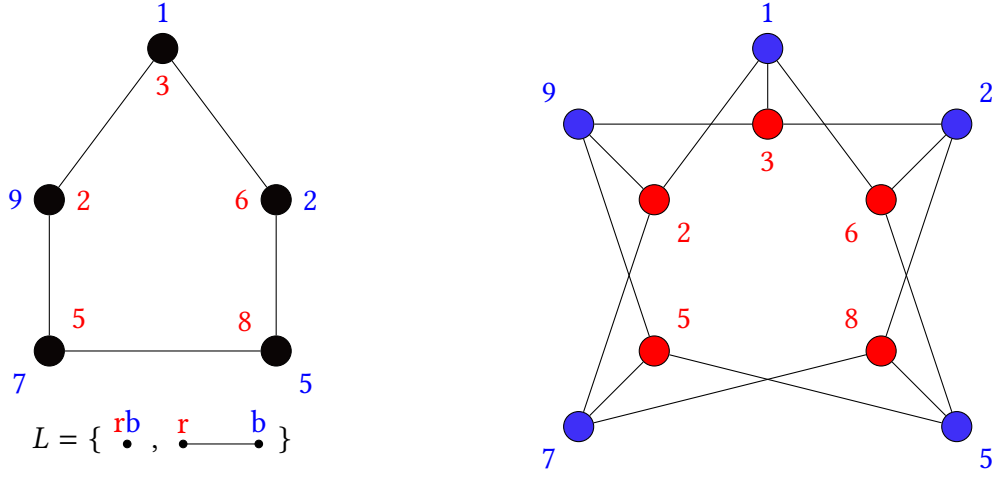


Figure 9.1: 2-coloured graph G and the partially separated graph G'_s .

9.1 Closure conditions

For a 2-coloured graph G , what are necessary conditions for G to contain a $\{K_1, K_2\}$ -decomposition? By Lemma 7.4.4 we have that it is both necessary and sufficient to find a perfect matching in G_s and G_s of a 2-coloured graph is bipartite. A necessary and sufficient condition for the existence of a perfect matching in bipartite graphs $G = (V_1 \cup V_2, E)$ is Hall's condition (Theorem 2.4.5), or equivalently for each set $X \subseteq V_1$ the defect $\Delta_X = |X| - |\Gamma(X)|$ is at most zero and $|V_1| = |V_2|$. *

To check for the existence of a matching, we could check the defect of each subset $X \subseteq V_s$. However, due to the nature of G_s , the more observations there are, the more subsets there are to check. This could get time-consuming quite quickly, as the number of subsets of a set S is $2^{|S|}$. We show that it is enough to only check certain subsets.

9.1.1 Weighted Hall's condition

Let $G = (V, E)$ be a vertex-weighted graph, that is there is a weight $w(v) \in \mathbb{N}_0$ associated with each vertex $v \in V$, then for $A \subseteq V$, $w(A) = \sum_{v \in A} w(v)$.

Lemma 9.1.1 (Weighted Hall). *Let $G = (V, E) = (X \cup Y, E)$ be a vertex-weighted bipartite graph. G has a perfect weighted matching if and only if $w(X) = w(Y)$ and for all $A \subseteq X$,*

$$w(A) \leq w(\Gamma(A)). \quad (9.1)$$

*See Section 2.4 for more details about matchings.

Proof. Without loss of generality there are no $v \in V$ with $w(v) = 0$. For each $v \in V$, let $W(v) = \{v_i : 1 \leq i \leq w(v)\}$ be a set of new elements and let all $W(v)$ be pairwise disjoint. Let $G_t = (V_t, E_t) = (X_t \cup Y_t, E_t)$, where

$$V_t = \bigcup_{v \in V} W(v)$$

$$E_t = \{v_i w_j : w \in \Gamma_G(v)\}.$$

For $A_t \subseteq V_t$, let $H(A_t) = \{v \in V : W(v) \cap A_t \neq \emptyset\}$. It is easy to verify that there is a perfect matching in G_t if and only if there is a perfect weighted matching in G . By Hall's theorem G_t has a perfect matching if and only if $|X_t| = |Y_t|$ and $\forall A_t \subseteq X_t$,

$$|A_t| \leq |\Gamma_{G_t}(A_t)|.$$

We note that $|\Gamma_{G_t}(A_t)| = w(H(\Gamma_{G_t}(A_t)))$ and $|A_t| \leq w(H(A_t))$, with equality only if $A_t \cap W(v) = W(v)$ for all $v \in H(A_t)$. In particular $|X_t| = w(H(X_t)) = w(X)$ and $|Y_t| = w(H(Y_t)) = w(Y)$. Moreover, every $A \subseteq X$ corresponds to $H(A_t)$ for a range of $A_t \subseteq X_t$ with $A_t \cap W(v) \neq \emptyset$ for all $v \in A$ and $A_t \cap W(v) = \emptyset$ for all $v \notin A$. Thus Hall's condition holding in G_t is equivalent to Eq. 9.1 holding in G . $\square$

We will refer to Eq. 9.1 as *weighted Hall's condition*.

9.1.2 Closure conditions

Corollary 9.1.2. *Given a partially separated graph $G'_s = (V, E) = (X \cup Y, E)$ of a 2-coloured graph with library $L = \{ \overset{rb}{\bullet}, \overset{r}{\bullet} \text{---} \overset{b}{\bullet} \}$, a perfect weighted matching exists if and only if $w(X) = w(Y)$ and for all subsets $A \subseteq X$,*

$$w(A) \leq w(\Gamma(A)). \quad (9.2)$$

Equivalently, one could state there is weighted matching in G'_s if and only if $w(X) = w(Y)$ and for all $A \subseteq X$, the defect

$$\Delta(A) = w(A) - w(\Gamma(A)) \leq 0. \quad (9.3)$$

This weighted version of Hall's marriage theorem translates into the following closure conditions for a general 2-coloured graph G .

Theorem 9.1.3 (Closure Conditions). *Let $G = (V, E)$ be a 2-coloured graph with library $L = \{ \overset{rb}{\bullet}, \overset{r}{\bullet} \text{---} \overset{b}{\bullet} \}$. Let $\sigma(A) = A \cup \Gamma(A)$ be the closed neighbourhood. G has*

an L -decomposition if and only if $r(V) = b(V)$ and for all $A \subseteq V$,

$$r(A) \leq b(\sigma(A)). \quad (9.4)$$

An equivalent statement would to be replace Eq. 9.4 with $b(A) \leq r(\sigma(A))$.

9.2 Defect closure conditions

Analogous to the defect version of Hall's theorem (see Theorem 2.4.6), we may form a defect version of weighted Hall.

Corollary 9.2.1 (Defect version of weighted Hall). *Let $G = (V, E) = (X \cup Y, E)$ be a vertex-weighted bipartite graph and let $0 \leq d \leq w(X)$ be an integer. G has a weighted matching leaving at most weight d from vertices in X unmatched if and only if for all $S \subseteq X$,*

$$w(S) \leq w(\Gamma(S)) + d. \quad (9.5)$$

Proof. Add a dummy vertex v to Y with $w(v) = d$ and adjacent to all vertices in X . Then by Lemma 9.1.1. $\square$

We will refer to Eq. 9.5 as *defect weighted Hall's condition*.

Corollary 9.2.2 (Defect closure conditions). *Let $G = (V, E)$ be a 2-coloured graph and $0 \leq d \leq r(\sigma(V))$ be an integer. G has a $\{K_1, K_2\}$ -decomposition with red defect at most d if and only if for all $A \subseteq V$,*

$$r(A) \leq b(\sigma(A)) + d. \quad (9.6)$$

Note that we can replace blue for red: $0 \leq d \leq r(\sigma(V))$ becomes $0 \leq d \leq b(\sigma(V))$ and $r(A) \leq b(\sigma(A)) + d$ becomes $b(A) \leq r(\sigma(A)) + d$.

9.3 The partially separated cycle has pathwidth 4

Lemma 9.3.1. *The 2-coloured partially separated cycle has pathwidth 4.[†]*

Proof. A path decomposition of width 4 can be seen in Fig. 9.2. It is easy to verify that this is best possible, because the graph contains K_5 as a minor (contract $e_1 a_2, a_1 b_2, \dots$). K_5 is a forbidden minor for graphs of treewidth at most 3 [97]. $\square$

[†]For more information on pathwidth and treewidth, see Section 2.6.

9.4. Checking weighted Hall's condition in graphs with bounded pathwidth

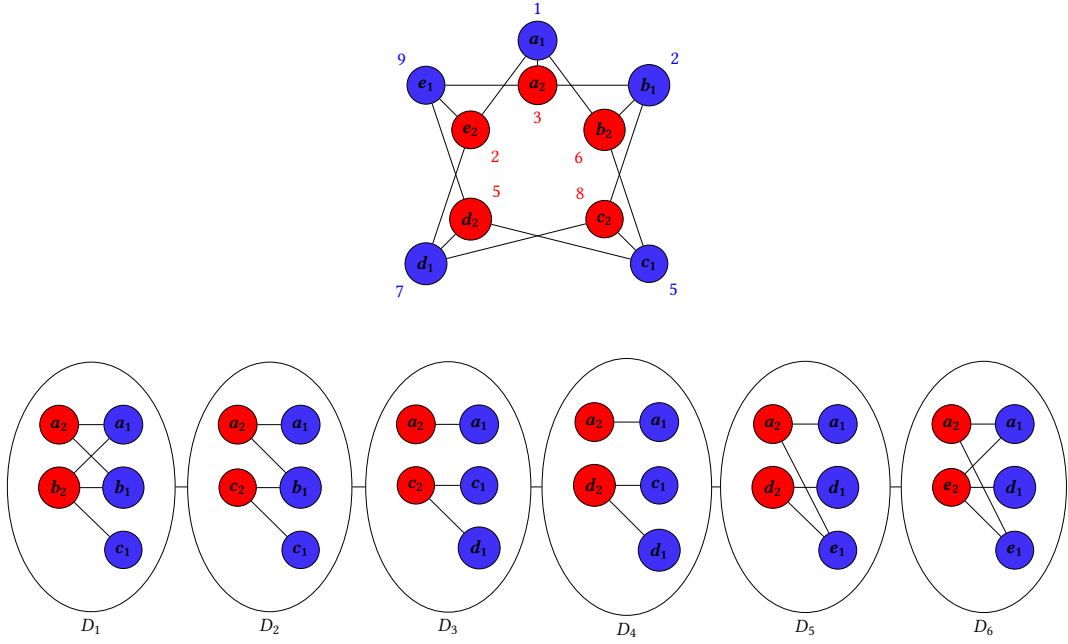
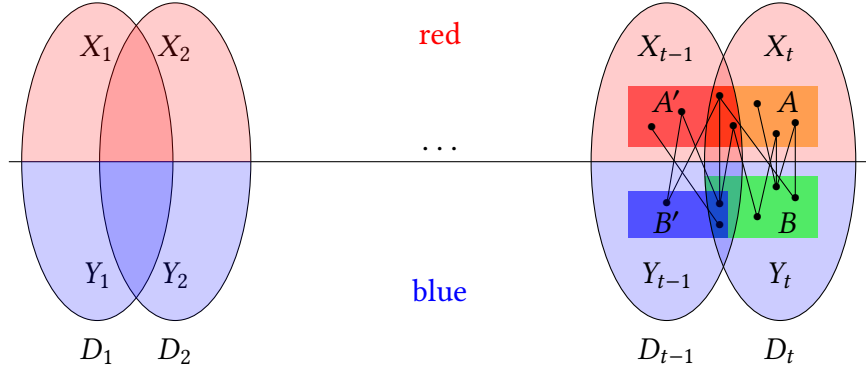


Figure 9.2: A partially separated cycle and its path-decomposition.

In Chapter 8 linear time algorithms are discussed for finding a $\{K_1, K_2\}$ -decomposition in 2-coloured trees and cycles. Because the partially separated cycle has pathwidth 4 by Lemma 9.3.1, we can consider weighted matchings in graphs with bounded pathwidth, instead of $\{K_1, K_2\}$ -decompositions. The pathwidth of a partially separated cycle does not depend on the number of observations and therefore such algorithms are appropriate for decompositions with an unbounded number of observations per vertex.

9.4 Checking weighted Hall's condition in graphs with bounded pathwidth

In this section an algorithm is presented that checks weighted Hall's condition in vertex-weighted bipartite graphs with bounded pathwidth. The algorithm searches for a set with the worst defect. The worst defect is found by using the path decomposition of the graph. In the case of a $\{K_1, K_2\}$ -decomposition, we work with the path decomposition of the partially separated cycle G'_S .


 Figure 9.3: Example pairs $(A', B') \subseteq D_{t-1}$ and $(A, B) \subseteq D_t$.

9.4.1 The strategy

Consider a vertex-weighted bipartite graph $G = (X \cup Y, E)$ with bounded pathwidth p . Each *node* and its *bag* in the path decomposition is considered at some time t , starting at one end and going through the nodes by following the path. We call the last bag we have seen at time t the *boundary* $D_t = X_t \cup Y_t$ and $D_{\leq t} = \bigcup_{i \leq t} D_i$. Let $X_t = D_t \cap X$ and $Y_t = Y \cap D_t$. When we have considered a collection of vertices $A \subseteq V(G)$, it means we have considered the induced subgraph $G[A] \subseteq G$, that is to say, we have also seen all edges between vertices of A .

At each step t we implicitly look for all possible sets $S \subseteq X \cap D_{\leq t}$ and their associated defect at time t : $\Delta_{\leq t}(S) = w(S) - w(\Gamma_{\leq t}(S))$, where $\Gamma_{\leq t}(S) = \Gamma(S) \cap D_{\leq t}$. When $t = t_f$ we have considered all sets $S \subseteq X$ and their defects $\Delta(S) = w(S) - w(\Gamma(S))$.

9.4.2 Determining intermediate defects

We identify a class of subsets by the way they intersect the boundary and then calculate the maximum intermediate defect of all sets in this class.

Definition 9.4.1. For each $t = 1, 2, \dots$, $A \subseteq X_t$, and $B \subseteq Y_t$, let

$$\mathcal{S}_t(A, B) = \{S \subseteq X \cap D_{\leq t} : S \cap D_t = A, \Gamma(S) \cap D_t = B\}.$$

Let $\Delta_t(A, B)$ be the maximum defect $\Delta_{\leq t}(S)$ of any $S \in \mathcal{S}_t(A, B)$ or $-\infty$ if there is none:

$$\Delta_t(A, B) = \begin{cases} \max_{S \in \mathcal{S}_t(A, B)} \Delta_{\leq t}(S) & \text{if } \mathcal{S}_t(A, B) \neq \emptyset \\ -\infty & \text{if } \mathcal{S}_t(A, B) = \emptyset \end{cases}$$

9.4. Checking weighted Hall's condition in graphs with bounded pathwidth

For $t = 1$ and $A \subseteq X_1, B \subseteq Y_1$ this simplifies to

$$\Delta_1(A, B) = \begin{cases} w(A) - w(B) & \text{if } B = \Gamma(A) \cap Y_1 \\ -\infty & \text{otherwise.} \end{cases} \quad (9.7)$$

Lemma 9.4.2. *Let f be the final node in a path decomposition of vertex-weighted bipartite graph $G = (V, E) = (X \cup Y, E)$ and let $0 \leq d \leq w(X)$ be an integer. G has a perfect weighted matching with defect at most d if and only if for all $A \subseteq X_f$ and $B \subseteq Y_f$,*

$$\Delta_f(A, B) \leq d. \quad (9.8)$$

Proof. We will show that Eq. 9.8 is equivalent to defect weighted Hall's condition (Eq. 9.5). For $S \subseteq X$,

$$\Delta_{\leq f}(S) = w(S) - w(\Gamma_{\leq f}(S)) = w(S) - w(\Gamma(S) \cap D_{\leq f}) = w(S) - w(\Gamma(S)).$$

because $D_{\leq f} = V$. Thus $\Delta_{\leq f}(S) \leq d$ is equivalent to $w(S) \leq w(\Gamma(S)) + d$.

Without loss of generality we can disregard $A \subseteq X_f, B \subseteq Y_f$, for which $\mathcal{S}_f(A, B) = \emptyset$. For all $A \subseteq X_f, B \subseteq Y_f$, for which $\mathcal{S}_f(A, B) \neq \emptyset$ we have

$$\Delta_f(A, B) = \max_{S \in \mathcal{S}_f(A, B)} \Delta_{\leq f}(S) \leq d,$$

that is, for all $S \in \mathcal{S}_f(A, B)$,

$$w(S) \leq w(\Gamma(S)) + d.$$

Moreover, each pair $(S, \Gamma(S))$ appears in $\mathcal{S}_f(A, B)$ for $A = S \cap X_f, B = \Gamma(S) \cap Y_f$. Note that it is possible for A and B to be empty. Thus by Lemma 9.1.1, G has a perfect weighted matching with defect at most d if and only if for all $A \subseteq X_f$ and $B \subseteq Y_f$, $\Delta_f(A, B) \leq d$. $\square$

9.4.3 Towards a recursive definition of $\Delta_t(A, B)$

We give some definitions that will allow us to express $\Delta_t(A, B)$ in terms of defects of pairs in D_{t-1} .

Definition 9.4.3. *For $A \subseteq X_t, B \subseteq Y_t$, let $\mathcal{C}_t(A, B) = \emptyset$ unless $\Gamma(A) \cap Y_t \subseteq B$ and if $\Gamma(A) \cap Y_t \subseteq B$, let $\mathcal{C}_t(A, B)$ be the set of pairs (A', B') with*

$$A' \subseteq X_{t-1}, A' \cap D_t = A \cap D_{t-1}, \quad (9.9a)$$

$$B' \subseteq Y_{t-1}, B' \cap D_t \subseteq B \cap D_{t-1}, \quad (9.9b)$$

$$B \setminus \Gamma(A) \subseteq B', \text{ and} \quad (9.9c)$$

$$\Delta_{t-1}(A', B') > -\infty. \quad (9.9d)$$

See Fig. 9.3 for example pairs $(A', B') \subseteq D_{t-1}$ and $(A, B) \subseteq D_t$. Note that combining 9.9b and 9.9c with $\Gamma(A) \cap Y_t \subseteq B$ gives $B = (\Gamma(A) \cup B') \cap Y_t$.

We will show that for $t > 1$, $\Delta_t(A, B)$ can be calculated by only considering pairs in $\mathcal{C}_t(A, B)$. That is, if we have calculated $\Delta_{t-1}(A', B')$ for all (A', B') in D_{t-1} , we can calculate all $\Delta_t(A, B)$ by only using those and we do not need to consider any vertices in $D_{\leq t-1} \setminus D_{t-1}$.

The following lemma is a technical lemma which states the implications of the existence of certain sets and pairs. We will use this to show that $\Delta_t(A, B)$ can be expressed recursively in terms of $\Delta_{t-1}(A', B')$.

Lemma 9.4.4. *Let $t \geq 2$, $A \subseteq X_t$ and $B \subseteq Y_t$.*

- (a) *If $S \in \mathcal{S}_t(A, B)$, then $S' \in \mathcal{S}_{t-1}(A', B')$ and $(A', B') \in \mathcal{C}_t(A, B)$, where $S' = S \cap X \cap D_{\leq t-1}$, $A' = S' \cap X_{t-1}$ and $B' = \Gamma(S') \cap Y_{t-1}$.*
- (b) *If $(A', B') \in \mathcal{C}_t(A, B)$, then there exists $S' \in \mathcal{S}_{t-1}(A', B')$ and $S' \cup A \in \mathcal{S}_t(A, B)$.*

Proof. (a) Suppose $S \in \mathcal{S}_t(A, B)$. Set $S' = S \cap X \cap D_{\leq t-1}$, $A' = S' \cap X_{t-1}$ and $B' = \Gamma(S') \cap Y_{t-1}$. Then condition 9.9a holds:

$$\begin{aligned} A' \cap D_t &= S' \cap D_{t-1} \cap D_t \\ &= S \cap X \cap D_{\leq t-1} \cap D_{t-1} \cap D_t = A \cap D_{t-1} \end{aligned} \quad (9.10)$$

Additionally, condition 9.9b holds:

$$\begin{aligned} B' \cap D_t &= \Gamma(S') \cap D_{t-1} \cap D_t \\ &\subseteq \Gamma(S) \cap D_{t-1} \cap D_t \\ &= B \cap D_{t-1}. \end{aligned} \quad (9.11)$$

Moreover, condition 9.9c holds:

$$\begin{aligned} B &= \Gamma(S) \cap D_t \\ &= (\Gamma(A) \cup \Gamma(S')) \cap D_t \\ &\stackrel{*}{=} (\Gamma(A) \cup (\Gamma(S') \cap D_{t-1})) \cap D_t \\ &= (\Gamma(A) \cup B') \cap D_t. \end{aligned} \quad (9.12)$$

9.4. Checking weighted Hall's condition in graphs with bounded pathwidth

Suppose $y \in \Gamma(S') \cap D_t$, but $y \notin D_{t-1}$. Let $x \in S'$ be a neighbour of y . By property 2 of a path decomposition $\{x, y\} \subseteq D_i$ for some i and as $y \notin D_{t-1}$ and $y \in D_t$ property 3 gives $\{x, y\} \not\subseteq D_i$ for $1 \leq i \leq t-1$. Thus $\{x, y\} \subseteq D_i$ for $i \geq t$. Because $x \in D_{\leq t-1}$, property 3 implies $x \in D_t$. Then by condition 9.9a, $x \in A$ and $y \in \Gamma(A)$. Thus $\star$ holds.

Finally condition 9.9d holds by existence of S' , so $(A', B') \in \mathcal{C}_t(A, B)$, and $S' \in \mathcal{S}_{t-1}(A', B')$ by construction.

(b) Suppose $(A', B') \in \mathcal{C}_t(A, B)$. Then $\Delta_{t-1}(A', B') > -\infty$ and $\mathcal{S}_{t-1}(A', B') \neq \emptyset$. Take $S' \in \mathcal{S}_{t-1}(A', B')$. Set $S = S' \cup A$. Clearly $S \cap X_t = A$. It is left to show that $\Gamma(S) \cap Y_t = B$.

$$\begin{aligned} \Gamma(S) \cap Y_t &= \Gamma(S' \cup A) \cap Y_t \\ &= (\Gamma(S') \cup \Gamma(A)) \cap Y_t \\ &= (B' \cup \Gamma(A)) \cap Y_t = B, \end{aligned} \tag{9.13}$$

as above. Thus $S \in \mathcal{S}_t(A, B)$. □

Corollary 9.4.5. $\mathcal{C}_t(A, B) = \emptyset$ if and only if $\mathcal{S}_t(A, B) = \emptyset$.

9.4.4 A recursive definition of $\Delta_t(A, B)$

Lemma 9.4.6. For $A \subseteq X_t, B \subseteq Y_t, t \geq 2$

$$\Delta_t(A, B) = \max\{\Delta_{t-1}(A', B') + w(A \setminus A') - w(B \setminus B') : (A', B') \in \mathcal{C}_t(A, B)\}, \tag{9.14}$$

or $\Delta_t(A, B) = -\infty$ if $\mathcal{C}_t(A, B) = \emptyset$.

Proof. If $\Delta_t(A, B) = -\infty$, then by Corollary 9.4.5, $\mathcal{C}_t(A, B) = \emptyset$. Otherwise, let $R = \max\{\Delta_{t-1}(A', B') + w(A \setminus A') - w(B \setminus B') : (A', B') \in \mathcal{C}_t(A, B)\}$. By definition $\Delta_t(A, B) \geq \Delta_{\leq t}(S)$ for all $S \in \mathcal{S}_t(A, B)$. Suppose $\Delta_t(A, B) > -\infty$ and let $S \in \mathcal{S}_t(A, B)$. Set $S' = S \cap X \cap D_{\leq t-1}$, $A' = S' \cap X_{t-1}$, and $B' = \Gamma(S') \cap Y_{t-1}$. Then by Lemma 9.4.4, $(A', B') \in \mathcal{C}_t(A, B)$. We obtain

$$\begin{aligned} \Delta_{\leq t}(S) &= \Delta_{\leq t-1}(S') + w(A \setminus A') - w(B \setminus B') \\ &\leq \Delta_{t-1}(A', B') + w(A \setminus A') - w(B \setminus B') \\ &\leq R. \end{aligned} \tag{9.15}$$

Chapter 9. The red-blue $\{K_1, K_2\}$ - model

Since this holds for all $S \in \mathcal{S}_t(A, B)$, it holds in particular for $S \in \mathcal{S}_t(A, B)$ with $\Delta_{\leq t}(S) = \Delta_t(A, B)$. Thus $\Delta_t(A, B) \leq R$.

Let (A', B') be a pair in $\mathcal{C}_t(A, B)$. Then $\Delta_{t-1}(A', B') > -\infty$ and $\mathcal{S}_{t-1}(A', B') \neq \emptyset$. Take $S' \in \mathcal{S}_{t-1}(A', B')$. Set $S = S' \cup A$, then $S \in \mathcal{S}_t(A, B)$ by Lemma 9.4.4 and

$$\begin{aligned} \Delta_{\leq t}(S) &= w(S) - w(\Gamma(S) \cap D_{\leq t}) \\ &= w(S') + w(A \setminus A') - w(\Gamma(S') \cap D_{\leq t-1}) - w(B \setminus B') \\ &\leq \Delta_{t-1}(A', B') + w(A \setminus A') - w(B \setminus B') \\ &\leq R \end{aligned} \tag{9.16}$$

Since this holds for all $(A', B') \in \mathcal{C}_t(A, B)$ and $S' \in \mathcal{S}_{t-1}(A', B')$, it holds in particular for (A', B') in $\mathcal{C}_t(A, B)$ maximising $\Delta_{t-1}(A', B') + w(A \setminus A') - w(B \setminus B')$ and $S' \in \mathcal{S}_{t-1}(A', B')$ such that $\Delta_{\leq t-1}(S') = \Delta_{t-1}(A', B')$; $\Delta_t(A, B) \geq \Delta_{\leq t}(S) = R$.

Combining these arguments gives $R \geq \Delta_t(A, B) \geq R$; thus $\Delta_t(A, B) = R$, as required. $\square$

9.4.5 Algorithm

Let $G = (X \cup Y, E)$ be a vertex-weighted bipartite graph with path decomposition of width $\leq p - 1$. An algorithm would start at one end of the decomposition and work its way up to the final node f , only storing the information of two nodes at any time. By Lemma 9.4.2, a matching exists, that leaves at most k vertices from X unmatched if and only if for each $A \subseteq X \cap D_f$ and $B \subseteq D_f \cap Y$ $\Delta_f(A, B) \leq k$.

The time to calculate each $\Delta_t(A, B)$ using Eq. 9.14 is $O(|\mathcal{C}_t(A, B)|) = O(2^p)$, because we need to check all possible pairs $(A', B') \in \mathcal{C}_t(A, B)$ to find the maximum value. This is needs to be done for every pair (A, B) in D_t . So the time complexity is $O(4^p)$ per node. If the number of nodes in the decomposition is linear with respect to n , this yields a linear time algorithm. For each calculation of $\Delta_t(A, B)$, it is enough to have all $\Delta_{t-1}(A', B')$ in memory. This gives an overall space complexity $O(2^p)$.

9.5 Checking weighted Hall's condition in graphs with bounded treewidth

In this section an algorithm is presented that checks weighted Hall's condition in weighted bipartite graphs with bounded treewidth. As above, the algorithm

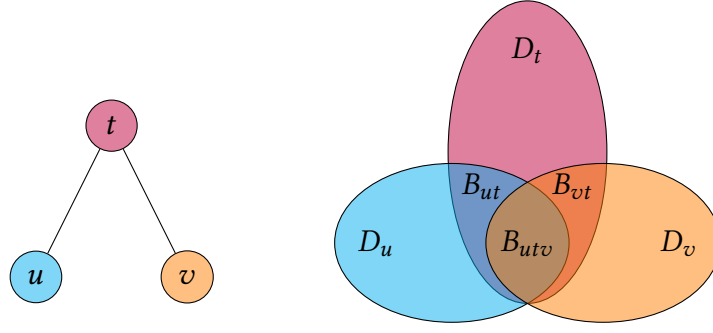


Figure 9.4: Parent node v with children u and w .

searches for a set with the worst defect. The worst defect is found by using the tree decomposition of the partially separated graph G'_s .

9.5.1 The strategy

Given a vertex-weighted bipartite graph $G = (X \cup Y, E)$ with bounded treewidth, we assume that a binary tree decomposition T is given. For each graph with constant treewidth, a binary tree decomposition can be found in linear time [13].

We find an expression for the defects at node t , given the defects of children nodes u and v . Leaves can be initialised as the first node in the path decomposition. One starts the calculation at the leaves and then works up to the root, using the expression for the defect at a node in terms of the defects of its children.

Let t be a node in the decomposition tree T with children u and v . Let the set of vertices in these nodes be denoted by D_t, D_u and D_v and their intersections as B_{ut}, B_{vt} and B_{utv} as in Fig. 9.4. Note that $B_{uv} = B_{utv}$ by the definition of a tree decomposition. For node $t \in V(T)$, let $V_{\leq t}$ be the union of all D_v in the tree rooted at t . Let $G_t = G[V_{\leq t}]$.

For each node $t \in V(T)$ we implicitly look for all possible sets $S \subseteq X \cap V_{\leq t}$ and their associated defect at node t : $\Delta_t(S) = w(S) - w(\Gamma_t(S))$, where $\Gamma_t(S) = \Gamma(S) \cap V_{\leq t}$. When $t = r$, the root, we have considered all sets $S \subseteq X$ and their defects $\Delta(S) = w(S) - w(\Gamma(S))$.

9.5.2 Redefining $\Delta_t(A, B)$

We generalise the definitions of $\Delta_t(A, B)$ and $S_t(A, B)$ given above.

Definition 9.5.1. For each node $t \in V(T)$ and $A \subseteq X_t, B \subseteq Y_t$, let

$$\mathcal{S}_t(A, B) = \{S \subseteq X \cap V_{\leq t} : S \cap D_t = A, \Gamma(S) \cap D_t = B\};$$

and let $\Delta_t(A, B)$ be the maximum defect $\Delta_t(S)$ of any $S \in \mathcal{S}_t(A, B)$ or $-\infty$ if there is none:

$$\Delta_t(A, B) = \begin{cases} \max_{S \in \mathcal{S}_t(A, B)} \Delta_t(S) & \text{if } \mathcal{S}_t(A, B) \neq \emptyset \\ -\infty & \text{if } \mathcal{S}_t(A, B) = \emptyset \end{cases}.$$

For $t \in V(T)$ a leaf, $A \subseteq X_t, B \subseteq Y_t$, this simplifies to

$$\Delta_t(A, B) = \begin{cases} w(A) - w(B) & \text{if } B = \Gamma(A) \cap Y_t \\ -\infty & \text{otherwise} \end{cases}. \quad (9.17)$$

Lemma 9.5.2. Let f be the root node in a tree decomposition of vertex-weighted bipartite graph $G = (V, E) = (X \cup Y, E)$ and let $0 \leq d \leq w(X)$ be an integer. G has a perfect weighted matching with defect at most d if and only if for all $A \subseteq X_f, B \subseteq Y_f$,

$$\Delta_f(A, B) \leq d. \quad (9.18)$$

Proof. As Lemma 9.4.2. □

9.5.3 Towards a recursive definition of $\Delta_t(A, B)$

As before, we will show that $\Delta_t(A, B)$ can be expressed in terms of $\Delta_u(A_u, B_u)$ and $\Delta_v(A_v, B_v)$ for $((A_u, B_u), (A_v, B_v)) \in \mathcal{C}_t(A, B)$, by generalising the definition of $\mathcal{C}_t(A, B)$.

Definition 9.5.3. Let $\mathcal{C}_t(A, B) = \emptyset$, unless $\Gamma(A) \cap Y_t \subseteq B$ and if $\Gamma(A) \cap Y_t \subseteq B$, $\mathcal{C}_t(A, B)$ is the set of pairs of pairs $((A_u, B_u), (A_v, B_v))$ such that

$$A_u \subseteq X_u, A_u \cap D_t = A \cap D_u \quad (9.19a)$$

$$B_u \subseteq Y_u, B_u \cap D_t \subseteq B \cap D_u \quad (9.19b)$$

$$\Delta_u(A_u, B_u) > -\infty \quad (9.19c)$$

$$A_v \subseteq X_v, A_v \cap D_t = A \cap D_v \quad (9.19d)$$

$$B_v \subseteq Y_v, B_v \cap D_t \subseteq B \cap D_v \quad (9.19e)$$

9.5. Checking weighted Hall's condition in graphs with bounded treewidth

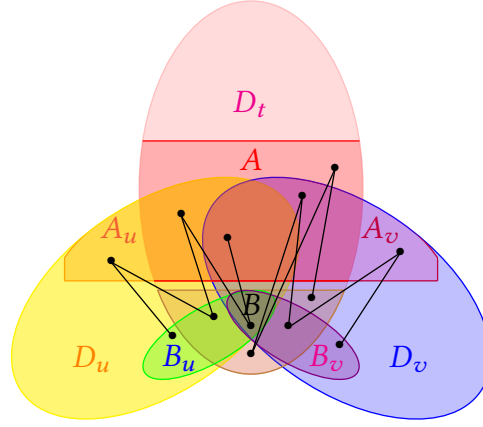


Figure 9.5: Example pair of pairs $((A_u, B_u), (A_v, B_v)) \in \mathcal{C}_t(A, B)$.

$$\Delta_v(A_v, B_v) > -\infty \quad (9.19f)$$

$$(\Gamma(A) \cup B_u \cup B_v) \cap D_t = B. \quad (9.19g)$$

See Fig. 9.5 for example pair of pairs $((A_u, B_u), (A_v, B_v)) \in \mathcal{C}_t(A, B)$.

Lemma 9.5.4. For $t \in V(T)$ not a leaf, $A \subseteq X_t, B \subseteq Y_t, A_u \subseteq X_u, B_u \subseteq Y_u, A_v \subseteq X_v, B_v \subseteq Y_v$.

(a) If $S \in \mathcal{S}_t(A, B)$, then

$$\begin{aligned} S \cap V_{\leq u} &\in \mathcal{S}_u(A_u, B_u), \\ S \cap V_{\leq v} &\in \mathcal{S}_v(A_v, B_v), \text{ and} \\ ((A_u, B_u), (A_v, B_v)) &\in \mathcal{C}_t(A, B), \end{aligned}$$

where $A_u = S_u \cap X_u, A_v = S_v \cap X_v, B_u = \Gamma(S_u) \cap Y_u$ and $B_v = \Gamma(S_v) \cap Y_v$.

(b) If

$$\begin{aligned} S_u &\in \mathcal{S}_u(A_u, B_u), \\ S_v &\in \mathcal{S}_v(A_v, B_v), \text{ and} \\ ((A_u, B_u), (A_v, B_v)) &\in \mathcal{C}_t(A, B), \end{aligned}$$

then $S_u \cup S_v \cup A \in \mathcal{S}_t(A, B)$.

Proof. (a) Suppose $S \in \mathcal{S}_t(A, B)$ and set $S_u = S \cap V_{\leq u}, S_v = S \cap V_{\leq v}, A_u = S_u \cap X_u, A_v = S_v \cap X_v, B_u = \Gamma(S_u) \cap Y_u$ and $B_v = \Gamma(S_v) \cap Y_v$. Then $S_u \in \mathcal{S}_u(A_u, B_u)$ and $S_v \in \mathcal{S}_v(A_v, B_v)$ by construction. We note that the conditions for u and v are symmetric, so it suffices to check Eq.9.19a-9.19c. These are exactly the same as in

the proof of Lemma 9.4.4. It is left to show that condition 9.19g holds:

$$\begin{aligned}
 B &= \Gamma(S) \cap D_t \\
 &= \Gamma(S_u \cup S_v \cup A) \cap D_t \\
 &= (\Gamma(S_u) \cup \Gamma(S_v) \cup \Gamma(A)) \cap D_t \\
 &\stackrel{\star}{=} ((\Gamma(S_u) \cap D_u) \cup (\Gamma(S_v) \cap D_v) \cup \Gamma(A)) \cap D_t \\
 &= (B_u \cup B_v \cup \Gamma(A)) \cap D_t.
 \end{aligned} \tag{9.20}$$

Equality $\star$ holds as in the proof of Lemma 9.4.4.

(b) Suppose $((A_u, B_u), (A_v, B_v)) \in \mathcal{C}_t(A, B)$, then there are $S_u \in \mathcal{S}_u(A_u, B_u)$ and $S_v \in \mathcal{S}_v(A_v, B_v)$. Set $S = S_u \cap S_v \cap A$, then

$$\begin{aligned}
 \Gamma(S) \cap Y_t &= (\Gamma(S_u) \cup \Gamma(S_v) \cup \Gamma(A)) \cap Y_t \\
 &= (B_u \cup B_v \cup \Gamma(A)) \cap Y_t = B,
 \end{aligned} \tag{9.21}$$

as before and $S \cap D_t = A$ by construction. Thus $S \in \mathcal{S}_t(A, B)$. $\square$

Corollary 9.5.5. $\mathcal{C}_t(A, B) = \emptyset$ if and only if $\mathcal{S}_t(A, B) = \emptyset$.

9.5.4 A recursive definition of $\Delta_t(A, B)$

For $((A_u, B_u), (A_v, B_v)) \in \mathcal{C}_t(A, B)$, set

$$\begin{aligned}
 \Delta_t(A_u, B_u, A_v, B_v) &= \Delta_u(A_u, B_u) + \Delta_v(A_v, B_v) + w(A \setminus (A_u \cup A_v)) \\
 &\quad - w(B \setminus (B_u \cap B_v)) - w(A_u \cap A_v) + w(B_u \cap B_v).
 \end{aligned}$$

Lemma 9.5.6. For $t \in V(T)$ not a leaf, $A \subseteq X_t, B \subseteq Y_t$

$$\Delta_t(A, B) = \max\{\Delta_t(A_u, B_u, A_v, B_v) : ((A_u, B_u), (A_v, B_v)) \in \mathcal{C}_t(A, B)\}, \tag{9.22}$$

or $\Delta_t(A, B) = -\infty$ if $\mathcal{C}_t(A, B) = \emptyset$.

Proof. If $\Delta_t(A, B) = -\infty$, by Corollary 9.5.5. Otherwise, let $R = \max\{\Delta_t(A_u, B_u, A_v, B_v) : ((A_u, B_u), (A_v, B_v)) \in \mathcal{C}_t(A, B)\}$. By definition $\Delta_t(A, B) \geq \Delta_t(S)$ for all $S \in \mathcal{S}_t(A, B)$. Now suppose $\Delta_t(A, B) > -\infty$ and take $S \in \mathcal{S}_t(A, B)$. Set $S_u = S \cap V_{\leq u}$ and $S_v = S \cap V_{\leq v}$, $A_u = S_u \cap X_u$ and $A_v = S_v \cap X_v$, and $B_u = \Gamma(S_u) \cap Y_u$ and $B_v = \Gamma(S_v) \cap Y_v$. Then $((A_u, B_u), (A_v, B_v)) \in \mathcal{C}_t(A, B)$ by Lemma 9.5.4 and we can express $\Delta_t(S)$ as

$$\Delta_t(S) = \Delta_u(S_u) + \Delta_v(S_v) + w(A \setminus (A_u \cup A_v))$$

9.5. Checking weighted Hall's condition in graphs with bounded treewidth

$$\begin{aligned}
& -w(B \setminus (B_u \cup B_v)) - w(A_u \cap A_v) + w(B_u \cap B_v) \\
& \leq \Delta_u(A_u, B_u) + \Delta_v(A_v, B_v) + w(A \setminus (A_u \cup A_v)) \\
& \quad - w(B \setminus (B_u \cup B_v)) - w(A_u \cap A_v) + w(B_u \cap B_v) \\
& = \Delta_t(A_u, B_u, A_v, B_v) \leq R.
\end{aligned} \tag{9.23}$$

Since this is true for any $S \in \mathcal{S}_t(A, B)$, it is true in particular for $S \in \mathcal{S}_t(A, B)$ with $\Delta_t(S) = \Delta_t(A, B)$. Thus $\Delta_t(A, B) \leq R$.

Let $((A_u, B_u), (A_v, B_v)) \in \mathcal{C}_t(A, B)$ and take $S_u \in \mathcal{S}_u(A_u, B_u)$ and $S_v \in \mathcal{S}_v(A_v, B_v)$. Set $S = S_u \cup S_v \cup A$, then $S \in \mathcal{S}_t(A, B)$ by Lemma 9.5.4:

$$\begin{aligned}
\Delta_t(S) &= w(S) - w(\Gamma(S) \cap V_{\leq t}) \\
&= w(S_u) + w(S_v) - w(\Gamma(S_u \cap V_{\leq u})) - w(\Gamma(S_v \cap V_{\leq v})) + w(A \setminus (A_u \cup A_v)) \\
&\quad - w(B \setminus (B_u \cup B_v)) - w(A_u \cap A_v) + w(B_u \cap B_v) \\
&= \Delta_u(S_u) + \Delta_v(S_v) + w(A \setminus (A_u \cup A_v)) - w(B \setminus (B_u \cup B_v)) \\
&\quad - w(A_u \cap A_v) + w(B_u \cap B_v) \\
&\leq \Delta_u(A_u, B_u) + \Delta_v(A_v, B_v) + w(A \setminus (A_u \cup A_v)) - w(B \setminus (B_u \cup B_v)) \\
&\quad - w(A_u \cap A_v) + w(B_u \cap B_v) \\
&= \Delta_t(A_u, B_u, A_v, B_v) \leq R.
\end{aligned} \tag{9.24}$$

This is true for any $((A_u, B_u), (A_v, B_v)) \in \mathcal{C}_t(A, B)$ and any $S_u \in \mathcal{S}_u(A_u, B_u)$ and $S_v \in \mathcal{S}_v(A_v, B_v)$. It holds in particular for $((A_u, B_u), (A_v, B_v))$ in $\mathcal{C}_t(A, B)$ maximising $\Delta_t(A_u, B_u, A_v, B_v)$ (i.e. $\Delta_t(A_u, B_u, A_v, B_v) = R$) and $S_u \in \mathcal{S}_u(A_u, B_u)$ and $S_v \in \mathcal{S}_v(A_v, B_v)$ such that $\Delta_u(S_u) = \Delta_u(A_u, B_u)$ and $\Delta_v(S_v) = \Delta_v(A_v, B_v)$. This gives equality in both cases, and thus $\Delta_t(S) = R$, which gives $\Delta_t(A, B) \geq \Delta_t(S) = R$.

Combining these arguments gives $R \geq \Delta_t(A, B) \geq R$: thus $\Delta_t(A, B) = R$. $\square$

9.5.5 Algorithmic Representation

Let $G = (X \cup Y, E)$ be a bipartite graph with binary tree decomposition of width $p - 1$ with root r . By Lemma 9.5.2 there is a matching that leaves at most k vertices from X unmatched if and only if for each $A \subseteq D_f \cap X$ and $B \subseteq D_f \cap Y$, $\Delta_f(A, B) \leq k$. These defects can be calculated by starting at the leaves of the tree decomposition and recursively calculating all intermediate defects, using Eq. 9.22. We move up the tree until we reach the root node r . This gives a time complexity of $O(8^p)$ per node, as for each node with two children, all triples of pairs need to be considered. If the

number of nodes in the decomposition is linear with respect to n , this yields a linear time algorithm. There is only a constant number of nodes in memory at any time, so the space complexity is $O(2^p)$ overall.

9.6 Weighted matchings in bipartite graphs with bounded treewidth

Given an algorithm that checks for the existence of a perfect weighted matching in a weighted bipartite graph G with bounded pathwidth or treewidth with time complexity $O(a)$ per node and space complexity $O(b)$ overall, a matching can be found by checking the range of possible values along each edge uv in $E(G)$. Let m be the number of edges in G and let d be the maximum weight at any vertex.

9.6.1 d is small

For small d we check the condition in $G \setminus uv$ with $w'(u) = w(u) - x$ and $w'(v) = w(v) - x$, where $x \in [0, \min(w(v), w(u))]$. Once a possible weight for an edge is found, the edge is removed from the graph and weights from the adjacent vertices are reduced accordingly. The time complexity of this method is $O(dma)$ per node. Note that this worst time complexity is not tight, as removing edges from the graph can only reduce the width of a decomposition. The space complexity is $O(b + m)$ overall. Note that if $b = O(m)$, this gives an overall space complexity of $O(m)$.

9.6.2 d is large

If d is very large, a gadget can be built for each edge to check if $\alpha \leq x \leq \beta$. We find the value of x with a binary search. We will show that this gadget does not affect the bipartiteness of G and that it increase the pathwidth by at most one and does not increase the treewidth. The time complexity is $O(ma \log d)$ per node and the space complexity stays the same as for small d .

Definition 9.6.1. Let $G = (X \cup Y, E)$ be a vertex-weighted bipartite graph and $u \in X$, $v \in Y$, $w(u) = A$, $w(v) = B$, and $uv \in E$. Let $0 \leq \alpha < \beta \leq \min(A, B)$ be integers and let u_1, u_2, u_3, u_4 be four new vertices with $w(u_1) = w(u_4) = \beta$ and $w(u_2) = w(u_3) = \beta - \alpha$. Let R be the path $R = uu_1u_2u_3u_4v$. Then $G(u, v, \alpha, \beta) = (V', E')$ is a weighted bipartite graph, where $V' = (X \cup u_2 \cup u_4) \cup (Y \cup u_1 \cup u_3)$ and $E' = P \cup E \setminus uv$.

9.6. Weighted matchings in bipartite graphs with bounded treewidth

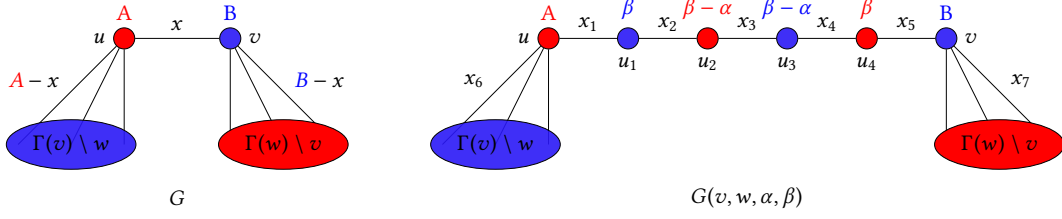


Figure 9.6: Vertex weighted graph G and $G(u, v, \alpha, \beta)$.

An example is depicted in Fig. 9.6.

Lemma 9.6.2. $G(u, v, \alpha, \beta)$ has a weighted matching if and only if G has a weighted matching with weight x along uv , and $\alpha \leq x \leq \beta$.

Proof. In $G(u, v, \alpha, \beta)$, let x_1 be the weight along uu_1 , x_5 be the weight along u_4v and for $2 \leq i \leq 4$, let x_i be the weight along edge $u_{i-1}u_i$, as depicted in Fig. 9.6. It is enough to show that $x_1 = x_5$ and $\alpha \leq x_1 \leq \beta$. Set $x_1 = x$, then

$$\begin{aligned} x_2 &= \beta - x_1 = \beta - x & x_5 &= \beta - x_4 = \beta - (\beta - x) = x \\ x_3 &= \beta - \alpha - x_2 = \beta - \alpha - (\beta - x) = x - \alpha & x_6 &= A - x_1 = A - x \\ x_4 &= \beta - \alpha - x_3 = \beta - \alpha - (x - \alpha) = \beta - x & x_7 &= B - x_5 = B - x. \end{aligned}$$

Moreover, $x_3 \leq \beta - \alpha$ (as one end has supply at most $\beta - \alpha$), so $x \leq \beta$ and $x_4 \leq \beta - \alpha$ (as one end has supply at most $\beta - \alpha$), so $x \geq \alpha$ as required. $\square$

Lemma 9.6.3. Let G have pathwidth p . Then $G(u, v, \alpha, \beta)$ has pathwidth at most $p + 1$.

Proof. Let $(X_1, X_2, \dots, X_m)$ be a path decomposition of G with width p . Let X_i be the node with the lowest index in which u and v appear together. Without loss of generality assume that $\min\{j : u \in X_j\} \leq \min\{j : v \in X_j\}$. Let $A = X_{i-1} \setminus u$ and $B = X_i \setminus \{u, v\}$. We have $|A \cap B| \leq p - 2$, because $|B| \leq p - 2$. We add nodes (Y_1, Y_2, Y_3, Y_4) between X_{i-1} and X_i , where $Y_1 = A \cap B \cup \{u, u_1, u_2\}$, $Y_2 = A \cap B \cup \{u, u_2, u_3\}$, $Y_3 = A \cap B \cup \{u, u_3, u_4\}$, and $Y_4 = A \cap B \cup \{u, u_4, v\}$. This increases the pathwidth by at most 1 $\square$

Lemma 9.6.4. $G(u, v, \alpha, \beta)$ has the same treewidth as G .

Proof. If G is a tree, then $G(u, v, \alpha, \beta)$ is a tree and the treewidth stays the same. Suppose G is not a tree and let $(\{X_i : i \in I\}, T)$ be a tree decomposition of G and let X_i be a bag that u and v appear in together. We attach the path $abcd$ to $i \in T$, where $X_a = \{u, v, u_1\}$, $X_b = \{u_1, u_2, v\}$, $X_c = \{u_2, u_3, v\}$, $X_d = \{u_3, u_4, v\}$. As G is not a tree, this tree decomposition has the same width as T . $\square$

2-coloured graph G	Decomposition with defect $\leq l$	
	Time complexity	Space complexity
G is a tree	$O(n)^a$	$O(n)^a$
G is a cycle	$O(dn)^a$	$O(n)^a$
G'_s has pathwidth $\leq p - 1$	$O(4^{p+1}m \log d)$ per node	$O(m)$ overall
G'_s has treewidth $\leq p - 1$	$O(8^p m \log d)$ per node	$O(m)$ overall

^aIn Chapter 8.

Table 9.1: Complexities for finding a $\{K_1, K_2\}$ -decomposition in different 2-coloured graphs G , where $d = \max_{v \in V(G)} |\text{col}(v)|$.

9.7 Conclusion

Table 9.1 displays the different decomposition algorithms in this chapter and the previous chapter. Chapter 10 discusses results for the general coloured $\{K_1, K_2\}$ -model. Chapter 11 introduces two new models: the likelihood model and the two-time model.

10

The general coloured $\{K_1, K_2\}$ - model

In this chapter will consider general R -coloured graphs and fix library

$$L \subseteq \{ \overset{ab}{\bullet}, \overset{a}{\bullet} \text{---} \overset{b}{\bullet} : (a, b) \in R \times R \}.$$

First we will state the connections between minimum cost matchings and decompositions. Next we show that the fully separated cycle has bounded pathwidth (and therefore also bounded treewidth). We give a matching algorithm for graphs with bounded treewidth. Finally we describe a defect matching algorithm for graphs with bounded treewidth.

10.1 Decompositions and minimum cost matchings

The closure conditions from Theorem 9.1.3 do not generalise, but finding a decomposition is still equivalent to finding a perfect matching in the fully separated graph.

To find an equivalent of an optimal decomposition in perfect matchings, we can give each edge a cost and find a minimum cost perfect matching. As K_1 -types have preference, “internal edges”, i.e. the edges from E_1 get lower cost than the “external edges” from E_2 . Blossom V is an algorithm that can find a minimum cost perfect

matching, with a worst time complexity of $O(n^3m)$, but a faster running time in practice [75]. This is a similar idea to the likelihood model outlined in Chapter 12.

Lemma 10.1.1. *Let G be an R -coloured graph. Then an optimal $\{K_1, K_2\}$ -decomposition in G corresponds to a minimum cost perfect matching in (the fully separated graph) G_s , if edges from E_1 have cost 0 and edges from E_2 have cost 1.*

Proof. For $v, w \in V(G)$, an edge $v_a v_b$ has cost one and an edge $v_a w_b$ has cost 0. A minimum cost perfect matching is a matching with a maximum number of edges with cost 0, i.e. a maximum number of K_1 -types. Thus it corresponds to an optimal $\{K_1, K_2\}$ -decomposition. $\square$

10.2 C_s has bounded pathwidth

Lemma 10.2.1. *Let C be an R -coloured cycle and let C_s be the fully separated graph of C , with library $L \subseteq \{ \overset{ab}{\bullet}, \overset{a}{\bullet} \text{---} \overset{b}{\bullet} : (a, b) \in R \times R \}$, then C_s has pathwidth $\leq 3d - 1$, where d is the maximal number of observations at any vertex in C .*

Proof. A cycle has a path decomposition with bags of ≤ 3 vertices [39]. See Fig. 2.3 for an example. We can replace each of those vertices v by $S(v)$, and $|S(v)| \leq d$. Thus C_s has pathwidth $\leq 3d - 1$. $\square$

Now a decomposition can be found by finding a matching in C_s with an algorithm for matchings in graphs with bounded pathwidth. This algorithm is appropriate for a scenario where the number of observations at each node is bounded by a reasonable constant, because the algorithm takes the fully separated graph as input.

10.3 Matching in graphs with bounded treewidth

We will give an algorithm for bounded treewidth. This includes an algorithm for bounded pathwidth, as a path decomposition is also a tree decomposition (but the reverse is not usually true). By Lemma 10.2.1, C_s has bounded pathwidth and therefore this algorithm would be appropriate for C_s .

We will show that a certain notion over ‘coverability’ of a node can be expressed recursively in terms of the ‘coverability’ of its children. We use this recursive expression to find a linear time algorithm for matchings in graphs with bounded treewidth.

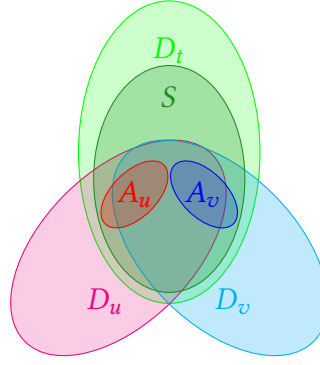


Figure 10.1: Node t , with children u and v and the subsets S , A_u and A_v .

10.3.1 Definitions

Fix graph G with treewidth $p - 1$ and corresponding binary tree decomposition $(\{D_i : i \in I\}, T)$. For node $t \in V(T)$, let V_t be the union of D_v for all nodes v in the tree rooted at t . Let $G_t = G[V_t]$ and given matching M , let $V(M)$ be the set of vertices incident with an edge in M , i.e. the set of vertices covered by M .

Definition 10.3.1. For node $t \in V(T)$ and $S \subseteq D_t$, let S be *t -coverable* if there exists a matching M in G_t , such that $V(M) = S \cup (V_t \setminus D_t)$. Let $I_t(S)$ be one if S is t -coverable and zero otherwise.

Lemma 10.3.2. Let G be a graph with a tree decomposition. Let t is a leaf node which is not the root and let $S \subseteq D_t$. The following two are equivalent:

1. $I_t(S) = 1$, and
2. there exists a matching M with $V(M) = S$.

Lemma 10.3.3. Let G be a graph with a tree decomposition. Let r be the root of the tree decomposition. Then G has a perfect matching if and only if $I_r(D_r) = 1$.

10.3.2 A recursive definition of $I_t(S)$

Let t be a node with children u and v . In this section we show that the coverability of a set S in D_t can be expressed in terms of the coverability of some F_1 and F_2 in D_u and D_v respectively.

Lemma 10.3.4. Given node $t \in T$ with children u and v , $S \subseteq D_t$, $I_t(S) = 1$ if and only if for some disjoint $A_u \subseteq D_u \cap S$ and $A_v \subseteq D_v \cap S$,

1. $I_u(A_u \cup (D_u \setminus D_t)) = 1$,

Chapter 10. The general coloured $\{K_1, K_2\}$ - model

2. $I_v(A_v \cup (D_v \setminus D_t)) = 1$, and
3. there is a matching M in G with $V(M) = S \setminus (A_u \cup A_v)$.

Proof. An example of nodes t , u and v and relevant subsets S , A_u and A_v can be seen in Fig. 10.1. First we note that $I_u(A_u \cup (D_u \setminus D_t)) = 1$ means that there exists matching M_u with $V(M_u) = A_u \cup (D_u \setminus D_t) \cup (V_u \setminus D_u) = A_u \cup (V_u \setminus D_t)$, as $V_u \cap D_t \subseteq D_u$ by the definition of a path decomposition. Similarly $I_v(A_v \cup (D_v \setminus D_t)) = 1$ means there exists matching M_v with $V(M_v) = A_v \cup (V_v \setminus D_t)$.

Now suppose t has two children u and v and all conditions hold. Then we have

$$M_u \text{ with } V(M_u) = A_u \cup (V_u \setminus D_t), \quad (10.1)$$

$$M_v \text{ with } V(M_v) = A_v \cup (V_v \setminus D_t), \text{ and} \quad (10.2)$$

$$M_S \text{ with } V(M_S) = S \setminus (A_u \cup A_v). \quad (10.3)$$

Set $M = M_u \cup M_v \cup M_S$. Then $V(M) = S \cup (V_t \setminus D_t)$, because M_u , M_v and M_S are all pairwise disjoint and their union covers $S \cup (V_t \setminus D_t)$.

Now suppose $I_t(S) = 1$, that is, there exists M with $V(M) = S \cup (V_t \setminus D_t)$. Set

$$M_S = M \cap G[D_t],$$

$$M_u = M \cap G_u \setminus M_S, \text{ and}$$

$$M_v = M \cap G_v \setminus (M_S \cup M_u) = M \cap G_v \setminus M_S,$$

because for $uv \in M$, if $u, v \in V_u$ and $u, v \in V_v$, then $u, v \in D_t$ by the definition of a tree decomposition and then $uv \in M_S$. So basically we split M into M_S , which is all the edges completely contained in D_t , and M_u and M_v , which are all edges contained completely in G_u and G_v respectively. Note that by the definition of a tree decomposition, every edge in G_t is completely contained in at least one bag D_x , for some x in the tree rooted at t . Thus $V(M_S) \cup V(M_u) \cup V(M_v) = S \cup (V_t \setminus D_t) = S \cup ((V_u \cup V_v) \setminus D_t)$. Set $A_u = V(M_u) \cap S$ and $A_v = V(M_v) \cap S$, then $A_u \subseteq D_u$, $A_v \subseteq D_v$ and

$$V(M_u) = A_u \cup V_u \setminus D_t$$

$$V(M_v) = A_v \cup V_v \setminus D_t,$$

as required. □

10.3.3 Algorithm sketch

Let G be a graph with treewidth at most $p - 1$ and corresponding binary tree decomposition $(\{D_i : i \in I\}, T)$ with root r . By Lemma 10.3.3, G has a perfect matching if and only if $I_r(D_r) = 1$ as described in Lemma 10.3.2. The algorithm first determines $I_t(S)$ for each leaf t and $S \subseteq D_t$ and then works its way up to the root, using the recursive conditions from Lemma 10.3.4 to calculate $I_t(S)$ for each non-leaf node.

Lemma 10.3.5. *The algorithm outlined above has time complexity $O(4^p)$ per node and space complexity $O(2^p)$ overall.*

Proof. Given node t with children u and v , and $S \subseteq D_t$, with $|S| = s$, there are at most 3^s choices for A_u and A_v (a vertex in S is either in A_u , A_v or in neither). Thus the number of choices of S , A_u , and A_v is at most

$$\sum_{s=0}^p \binom{p}{s} 3^s = 4^p,$$

by the binomial theorem. It takes $p^{2.5}$ per node to find a matching, thus the time complexity is $O(4^p)$. The space complexity is $O(2^p)$ overall, as for each node t only $I_t(S)$ for all $S \subseteq D_t$ need to be stored and only a constant number of nodes are ever considered at the same time. $\square$

If one wishes to find a matching explicitly, the time complexity stays unchanged and the space complexity is $O(4^p)$ per node.

10.4 Defect matching

The previous approach can be modified so that it finds a matching with a minimal number of uncovered vertices.

Definition 10.4.1. *For disjoint $A, B \subseteq D_t$ and $k \in [|B|, |V_t|]$, $J_t(A, B)$ is a set of natural numbers such that $k \in J_t(A, B)$ if there exists matching M in G_t such that $V(M) = A \cup (V_t \setminus (D_t \cup U))$, where $U \subseteq V_t$, $|U| = k$ and $U \cap D_t = B$. Let $K_t(A, B) = \min\{k : k \in J_t(A, B)\}$.*

Lemma 10.4.2. *Let G be a graph with a tree decomposition T . For leaf node l and disjoint $A, B \in D_l$, $k \in J_l(A, B)$ if and only if there exists a matching M in $G[D_l]$, with $V(M) = A$ and $|B| = k$. Moreover, for every node $t \in T$, $|V_t| \in J_t(\emptyset, D_t)$,*

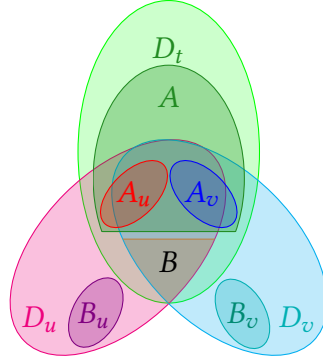


Figure 10.2: Node t , with children u and v and the subsets A and B , A_u and B_u , and A_v and B_v .

and the number of covered vertices is always even. That is, if $i \in J_t(A, B)$, then $|A \cup B \cup (V_t \setminus D_t)| - i$ is even.

Lemma 10.4.3. *Let G be a graph with a tree decomposition. Let r be the root of the tree decomposition. G has a matching leaving at most k vertices uncovered if and only if*

$$\min_{B \subseteq D_r} K_r(D_r \setminus B, B) \leq k.$$

10.4.1 A recursive definition of $J_t(A, B)$

In this section we show how the ‘defect coverability’ of a node t can be expressed in terms of the ‘defect coverability’ of its children.

Lemma 10.4.4. *For node t with children u and v , $k \in J_t(A, B)$ if and only if there exist disjoint $A_u \subseteq D_u \cap A$, $A_v \subseteq D_v \cap A$, $B_u \subseteq D_u \setminus D_t$, $B_v \subseteq D_v \setminus D_t$ and $a, b \in \mathbb{N}_0$ such that*

- (a) $a \in J_u(A_u \cup (D_u \setminus (D_t \cup B_u)), B_u)$,
- (b) $b \in J_v(A_v \cup (D_v \setminus (D_t \cup B_v)), B_v)$,
- (c) *there is a matching $M \subseteq G_t : V(M) = A \setminus (A_u \cup A_v)$, and*
- (d) $|B| = k - a - b$.

Proof. Suppose that (a)-(d) are true. See Fig. 10.2 for a visualisation of a triple of pairs. There are matchings $M_u \subseteq G_u$, $M_v \subseteq G_v$, $M_A \subseteq G_t$ and sets of unmatched vertices $U_u \subseteq V_u$, $U_v \subseteq V_v$ such that $|U_u| = a$, and $|U_v| = b$. We first show the following:

$$A_u \cup (D_u \setminus (D_t \cup B_u)) \cup (V_u \setminus (D_u \cup U_u)) = A_u \cup (V_u \setminus (D_t \cup U_u)) \quad (10.4)$$

Let $X = A_u \cup (D_u \setminus (D_t \cup B_u)) \cup (V_u \setminus (D_u \cup U_u))$ and $Y = A_u \cup (V_u \setminus (D_t \cup U_u))$.

Suppose $x \in X$, if

- (i) $x \in A_u \implies x \in Y$, as $A_u \subseteq Y$,
- (ii) $x \in D_u \setminus (D_t \cup B_u) = D_u \setminus (D_t \cup U_u)$, as $U_u \cap D_u = B_u \implies x \in V_u \setminus (D_t \cup U_u)$ as $D_u \subseteq V_u \implies x \in Y$,
- (iii) $x \in V_u \setminus (D_u \cup U_u) \implies x \in V_u \setminus (D_t \cup U_u)$, as $D_t \cap V_u \subseteq D_u$ by properties of a tree decomposition $\implies x \in Y$.

Conversely, suppose $x \in Y$, if

- (i) $x \in A_u \implies x \in X$, as $A_u \subseteq X$,
- (ii) $x \in V_u \setminus (D_t \cup U_u) = ((V_u \setminus D_u) \cup D_u) \setminus (D_t \cup U_u) \implies x \in (V_u \setminus D_u) \setminus (D_t \cup U_u) = V_u \setminus (D_u \cup U_u)$ (as $D_t \cap V_u \subseteq D_u$) or $x \in D_u \setminus (D_t \cup U_u) \implies x \in X$.

Thus $x \in X$ if and only if $x \in Y$ and thus $X = Y$ as required.

We now have by Eq. 10.4 that

$$\begin{aligned} V(M_u) &= A_u \cup (D_u \setminus (D_t \cup B_u)) \cup (V_u \setminus (D_u \cup U_u)) \\ &= A_u \cup (V_u \setminus (D_t \cup U_u)), \\ V(M_v) &= A_v \cup (V_v \setminus (D_t \cup U_v)), \text{ as above, and} \\ V(M_A) &= A \setminus (A_u \cup A_v). \end{aligned}$$

Set $M = M_u \cup M_v \cup M_A$, then

$$\begin{aligned} V(M) &= (A \setminus (A_u \cup A_v)) \cup A_u \cup (V_u \setminus (D_t \cup U_u)) \cup A_v \cup (V_v \setminus (D_t \cup U_v)) \\ &= A \cup (V_t \setminus (D_t \cup U_u \cup U_v)). \end{aligned}$$

Set $U_t = U_u \cup U_v \cup B$, then

$$|U_u \cup U_v \cup B| = |U_u| + |U_v| + |B| = a + b + (k - a - b) = k.$$

Thus $k \in J_t(A, B)$.

Suppose $k \in J_t(A, B)$, that is there exists a matching M in G_t such that $V(M) = A \cup (V_t \setminus (D_t \cup U_t))$, $U_t \subseteq V_t$, $|U_t| = k$ and $U_t \cap D_t = B$. Set

$$\begin{aligned} M_A &= M \cap G[D_t], \\ M_u &= M \cap G_u \setminus M_A, \text{ and} \\ M_v &= M \cap G_v \setminus (M_A \cup M_u) = M \cap G_v \setminus M_A, \end{aligned}$$

Chapter 10. The general coloured $\{K_1, K_2\}$ - model

as $D_u \cap D_v \subseteq D_t$ by properties of a tree decomposition, and set

$$\begin{aligned} U_u &= U_t \cap (V_u \setminus D_t), & B_u &= U_u \cap D_u, \text{ and} \\ U_v &= U_t \cap (V_v \setminus D_t), & B_v &= U_v \cap D_v. \end{aligned}$$

Set $A_u = V(M_u) \cap A$, and $A_v = V(M_v) \cap A$. Then $A_u \subseteq D_u \cap A$, $A_v \subseteq D_v \cap A$ and

$$\begin{aligned} V(M_u) &= A_u \cup (V_u \setminus (D_t \cup U_u)), \\ V(M_v) &= A_v \cup (V_v \setminus (D_t \cup U_v)). \end{aligned}$$

Therefore (a) and (b) hold, as

$$\begin{aligned} |U_u| &\in J_u(A_u \cup (D_u \setminus (D_t \cup B_u)), B_u), \text{ and} \\ |U_v| &\in J_v(A_v \cup (D_v \setminus (D_t \cup B_v)), B_v). \end{aligned}$$

Also $V(M_A), A_u, A_v$ are pairwise disjoint by construction, so as $A = V(M_A) \cup A_u \cup A_v$, we conclude $V(M_A) = A \setminus (A_u \cup A_v)$ and that (c) holds. Finally B, B_u, B_v are disjoint by construction, therefore (because $U_t = B \cup U_u \cup U_v$) we have $|B| = k - |U_u| - |U_v|$ and so (d) holds. $\square$

10.4.2 Algorithm sketch

Let G be a graph with treewidth at most $p - 1$ and corresponding binary tree decomposition $(\{D_i : i \in I\}, T)$ with root r . Let n be the number of vertices and $k \in \{0, 1, \dots, n\}$ be a constant. By Lemma 10.4.3, G has a matching leaving at most k vertices unmatched if and only if

$$\min_{B \subseteq D_r} K_r(D_r \setminus B, B) \leq k.$$

An algorithm first determines $J_t(A, B)$ for each leaf t and disjoint $A, B \subseteq D_t$ and then calculates $J_t(A, B)$ for each non-leaf node by using the recursive formulation from Lemma 10.4.4.

Lemma 10.4.5. *The above described algorithm has time complexity $O(20^p k)$ and space complexity $O(20^p k)$ overall.*

Proof. Given node t with children u and v , $J_t(A, B)$ is recursively calculated for all disjoint $A, B \subseteq D_t$. For this we need to consider all pairs of pairs $((A_u, B_u), (A_v, B_v))$ satisfying Eq. (a)-(d). If $|A| = x$ and $|B| = y$, then there are at most $\binom{p}{x}$ choices for A

	Time	Space	Section
Defect= 0	$O(4^p)$	$O(4^p)$	10.3
Defect $\leq k$	$O(20^p k)$	$O(20^p k)$	10.4

Table 10.1: Complexities for finding a $\{K_1, K_2\}$ -decomposition in an R -coloured graph G , when G_s has treewidth at most $p - 1$.

and $\binom{p-x}{y}$ choices for B . Vertices in A can be in exactly one of three categories: in A_u , in A_v or in neither, which gives a 3^x factor. There are at most 2^p choices to make up B_u and the same is true for B_v . Moreover, for $a \in J_u(A_u \cup (D_u \setminus (D_t \cup B_u)), B_u)$, and $b \in J_v(A_v \cup (D_v \setminus (D_t \cup B_v)), B_v)$, $0 \leq a, b \leq k$. Finally, a matching in D_t can be found in $p^{2.5}$ time. Thus an algorithm calculating each $J_t(A, B)$ as defined in Lemma 10.4.4 has at most the following complexity per node:

$$\begin{aligned}
& 2kp^{2.5}2^p2^p \sum_{x=0}^p \sum_{y=0}^{p-x} \binom{p}{x} \binom{p-x}{y} 3^x \\
&= 2kp^{2.5}4^p \sum_{x=0}^p \binom{p}{x} 3^x \sum_{y=0}^{p-x} \binom{p-x}{y} 1^{p-x-y} 1^y \\
&= 2kp^{2.5}4^p \sum_{x=0}^p \binom{p}{x} 3^x 2^{p-x} \\
&= 2kp^{2.5}4^p 5^p = O(20^p k),
\end{aligned}$$

by applying the binomial theorem twice. The space complexity is $O(20^p k)$ overall. $\square$

If a matching needs to be found explicitly, the space complexity goes up to $O(20^p k)$ per node, because the sets now have to be remembered.

10.5 Conclusion

Table 10.1 shows an overview of the complexity of the $\{K_1, K_2\}$ -decomposition algorithms presented in this chapter. This concludes the discussion of the coloured L -model. Chapter 11 introduces two new models: the likelihood model and the two-time model.

11

The likelihood and two-time models

This chapter focuses on the likelihood and two-time models. These are two generalisations of the coloured L -model, refining the concept of uncertainty and introducing time. We first introduce the likelihood model and then extend this to the two-time model and describe relevant problems in the context of these models. We show that these problems are NP-hard to solve in general and which conditions make them tractable. Finally, an overview of the tractable and intractable cases for both models is given.

11.1 A note on multi-set notation

In this chapter we make extensive use of multi-sets.* As we will consider bijections and matchings between multi-sets, it is important that we are able to distinguish between all elements. We will think of an element as an ordered pair (a, i) , where a is the element and $1 \leq i \leq m(a)$ is the index of this element. However, for the sake of simplicity of notation, we will omit the index whenever possible. For example for the set $A = \{(a, 1), (a, 2), (b, 1), (b, 2), (b, 3), (c, 1)\}$, we will write ‘each $x \in A$ ’, to mean ‘each ordered pair (x, i) in A for which $x \in \{a, b, c\}$ and $1 \leq i \leq m(x)$ ’. We will encounter this for example in Definitions 11.3.1 and 11.6.1.

*See Section 2.2.

11.2 The likelihood model

In this section we describe the likelihood model in relation to the likelihood problem. Suppose that we have a finite non-empty set V , where each $v \in V$ represents a geographical location. Let R be a finite non-empty set of *radars*, which we will represent as colours and let a *ship* $\mathbf{s}$ be a vector of non-negative integers $(s_r : r \in R)$. One can equivalently think of a ship $\mathbf{s}$ as a multi-set of colours from R :

$$\mathbf{s} = \{(r, 1), \dots, (r, s_r) : r \in R\}.$$

This is more convenient in some cases and we will use both expressions interchangeably. Let L be a finite non-empty set of ships, which we will refer to as a *library*. Let M be an $L \times R$ matrix of non-negative integers

$$M = (l(\mathbf{s}, r) : \mathbf{s} \in L, r \in R),$$

where $l(\mathbf{s}, r) = s_r$, so the rows of M are the ship vectors. This is an alternative way of representing the library L .

For $v \in V, \mathbf{s} \in L$, let $Z_{v,\mathbf{s}}$ be a non-negative random variable, representing the number of ships of type $\mathbf{s}$ at v . At some point in time, we imagine that there is a random distribution of ships from L on the locations in V , where the number of ships $\mathbf{s}$ at v is equal to $Z_{v,\mathbf{s}}$. We assume that $Z_{v,\mathbf{s}}$ has a geometric distribution with parameter $p_{v,\mathbf{s}}, 0 \leq p_{v,\mathbf{s}} < 1$, that is

$$\mathbb{P}(Z_{v,\mathbf{s}} = i) = p_{v,\mathbf{s}}^i (1 - p_{v,\mathbf{s}}) \quad \text{for } i = 0, 1, 2, \dots$$

A geometric random variable $Z_{v,\mathbf{s}}$ with parameter $p = p_{v,\mathbf{s}}$ has mean $\frac{p}{1-p}$ and variance $\frac{p}{(1-p)^2}$. We assume that the random variables $Z_{v,\mathbf{s}}$ are independent. Note that we are also assuming that the location v is ‘large’, so that it can accommodate more than one ship.

Definition 11.2.1. Let Z be a $V \times L$ matrix called the *ship distribution*:

$$Z = (p_{v,\mathbf{s}} : v \in V, \mathbf{s} \in L),$$

where each entry is in $[0, 1)$.

Definition 11.2.2. Let an R -*multi-colouring* $P : V \rightarrow \mathbb{N}^R$ be a $V \times R$ matrix of non-negative integers

$$P = (\rho(v, r) : v \in V, r \in R).$$

Similarly, let an ***L*-multi-colouring** $\Lambda : V \rightarrow \mathbb{N}^L$ be a $V \times L$ matrix of non-negative integers

$$\Lambda = (\lambda(v, s) : v \in V, s \in L).$$

As every colouring in this chapter is a multi-colouring, we will refer to multi-colourings as colourings from now on. See Example 5 for an *R*-colouring and an *L*-colouring.

At some fixed point in time we imagine $\rho(v, r)$ to be the number of radars r observed at location v and $\lambda(v, s)$ to be the number of ships of type s residing at location v . That is, for each location $v \in V$, P specifies the number of appearances of each colour (or radar) that are observed at this location and Λ specifies the number of ships of each type of ship $s \in L$. We will often speak of a ship s (at location v) in Λ , by which we mean one of the ships s (at location v), that make up Λ .

Definition 11.2.3. We will write capital upsilon $\Upsilon : V \rightarrow \mathbb{N}^R$ for a **random *R*-colouring** and for a fixed *R*-colouring $P : V \rightarrow \mathbb{N}^R$ we will write $\Upsilon = P$ for the event that random *R*-colouring Υ is equal to P . Similarly, we will write capital theta $\Theta : V \rightarrow \mathbb{N}^L$ for a **random *L*-colouring** and for a fixed *L*-colouring $\Lambda : V \rightarrow \mathbb{N}^L$ we will write $\Theta = \Lambda$ for the event that random *L*-colouring Θ is equal to Λ .

Note that given *L*-colouring $\Lambda : V \rightarrow \mathbb{N}^L$,

$$\mathbb{P}(\Theta = \Lambda) = \prod_{v \in V} \prod_{s \in L} \mathbb{P}(Z_{v,s} = \lambda(v, s)). \quad (11.1)$$

Definition 11.2.4. Given an *L*-colouring Λ , **the direct *R*-colouring** is $P_\Lambda = (\rho(v, r) : v \in V, r \in R)$, where

$$\rho(v, r) = \sum_{s \in L} \lambda(v, s) s_r = \sum_{s \in L} \lambda(v, s) l(s, r),$$

i.e. $P_\Lambda = \Lambda M$.

Ships are not observed as ships, but each radar $r \in s$ that the ship carries is observed separately. We shall assume that each radar $r \in s$ for s at location v is observed independently at random at a location $X_{v,r}$, according to a set distribution. This will be referred to as *dispersion of radars*.

Definition 11.2.5. Let X be a $V \times V \times R$ matrix called the **dispersion distribution**:

$$X = (\mathbb{P}(X_{v,r} = w) : v \in V, w \in V, r \in R).$$

So an L -colouring Λ is observed as a random R -colouring P . That is, if we consider all radar instances (including copies of the same radar r) in the direct R -colouring P_Λ , each is independently observed at a random location, according to some given distribution, dependent on its location and radar type. For example, if V is the vertex set of some graph G , the *closed neighbourhood distribution* is defined for $v, x \in V$ and $r \in R$ as follows:

$$\mathbb{P}(X_{v,r} = x) = \begin{cases} \frac{1}{2} & \text{if } x = v, \\ \frac{1}{2d(v)} & \text{if } x \in \Gamma(v), \\ 0 & \text{otherwise.} \end{cases}$$

Recall that $d(v)$ is the degree of a vertex.

Definition 11.2.6. Suppose V is the vertex set of a graph G . We say that X is a *G -dispersion distribution* if for each $v, w \in V$ and $r \in R$,

$$\mathbb{P}(X_{v,r} = w) > 0 \iff w \in \{v\} \cup \Gamma(v).$$

Note that $\mathbb{P}(X_{v,r} = w) > 0 \iff \mathbb{P}(X_{w,r} = v) > 0$ and that $\mathbb{P}(X_{v,r} = w) > 0$ implies that $\mathbb{P}(X_{v,q} = w) > 0$ for each $q \in R$, as we do not differentiate between different colours. The closed neighbourhood distribution is an example of a G -dispersion distribution.

Problem 5 (Likelihood problem (LHP)). Given a set of locations V and a set of colours R , an R -colouring $P : V \rightarrow \mathbb{N}^R$, a dispersion distribution X , a library L and a ship distribution Z , find an L -colouring $\Lambda : V \rightarrow \mathbb{N}^L$, which is plausible according to Z , such that P_Λ disperses to P with non-zero probability according to X (if such a colouring exists).

We might also want to consider the following: if there is an L -colouring Λ as sought above, can we find a maximum likelihood estimator $\hat{\Lambda}$, or sample from all possible Λ ?

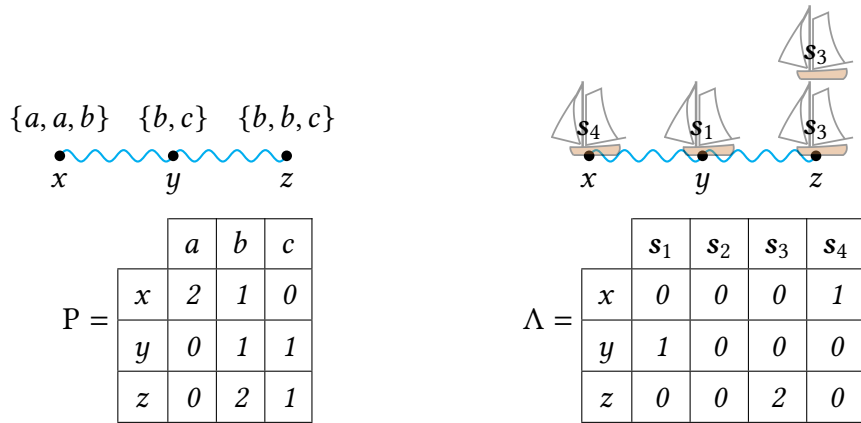
Example 5 (LHP-instance). Let $V = \{x, y, z\}$ and suppose V is the vertex set of the path xyz . We think of x, y and z as adjacent bearings, along which we observe radars. Now suppose $R = \{a, b, c\}$ and the library consists of four ships $s_1, \dots, s_4$:

$$M = \begin{array}{c|ccc} & a & b & c \\ \hline s_1 & 0 & 2 & 0 \\ s_2 & 1 & 1 & 0 \\ s_3 & 0 & 1 & 1 \\ s_4 & 2 & 0 & 0 \end{array}$$

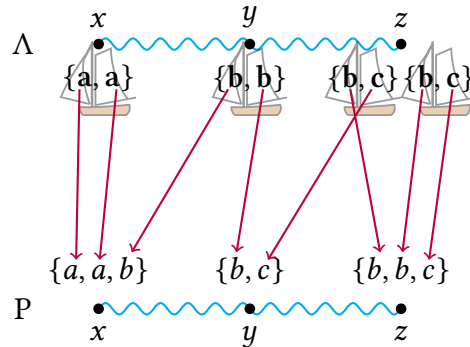
For $v, w \in V$ and $r \in R$,

$$\mathbb{P}(X_{v,r} = w) = \begin{cases} \frac{1}{2} & \text{if } w = v, \\ \frac{1}{2|\Gamma(v)|} & \text{if } w \in \Gamma(v), \\ 0 & \text{otherwise.} \end{cases}$$

and $p_{v,s_i} = \frac{1}{2}$ for $v \in V$ and $1 \leq i \leq 4$. Below is an example R -colouring on the left and a possible solution Λ on the right:



A possible dispersion is given below:



11.2.1 Calculating the probability of a solution

Now suppose we observe R -colouring P and we guess L -colouring Λ , then

$$\mathbb{P}(\Theta = \Lambda | \Upsilon = P) = \frac{\mathbb{P}((\Upsilon = P) \cap (\Theta = \Lambda))}{\mathbb{P}(\Upsilon = P)} \propto \mathbb{P}(\Upsilon = P | \Theta = \Lambda) \mathbb{P}(\Theta = \Lambda).$$

The $\mathbb{P}(\Upsilon = P)$ term is irrelevant, because it is constant (so independent of our choice of Λ). Given the distribution of $Z_{v,s}$ as described above, we have

$$\begin{aligned} \mathbb{P}(\Theta = \Lambda) &= \prod_{s \in L} \prod_{v \in V} \mathbb{P}(Z_{v,s} = \lambda(v, s)) \\ &= \prod_{s \in L} \prod_{v \in V} p_{v,s}^{\lambda(v,s)} (1 - p_{v,s}) \\ &= \left(\prod_{s \in L} \prod_{v \in V} (1 - p_{v,s}) \right) \prod_{s \in L} \prod_{v \in V} p_{v,s}^{\lambda(v,s)} \\ &\propto \prod_{s \in L} \prod_{v \in V} p_{v,s}^{\lambda(v,s)}, \end{aligned}$$

as $\prod_{s \in L} \prod_{v \in V} (1 - p_{v,s})$ is constant.

For given Λ and P , let $F(\Lambda, P)$ be the set of functions $f : P_\Lambda \rightarrow P$, which describe all ways Λ could disperse to P with non-zero probability. Observe that for this set to be non-empty, the total number of appearances of each colour in the direct R -colouring P_Λ must match that in P . Let $f \in F(\Lambda, P)$, and let $P_\Lambda \xrightarrow{f} P$ be the event that Λ disperses to P according to f . Then

$$\mathbb{P}(\Upsilon = P | \Theta = \Lambda) = \sum_{f \in F(\Lambda, P)} \mathbb{P}(P_\Lambda \xrightarrow{f} P \mid \Theta = \Lambda).$$

In many cases $F(\Lambda, P)$ is too big to work with. Let $f_m \in F(\Lambda, P)$ be such that $\mathbb{P}(P_\Lambda \xrightarrow{f_m} P \mid \Theta = \Lambda)$ is maximum. We have

$$|F(\Lambda, P)| \mathbb{P}(P_\Lambda \xrightarrow{f_m} P \mid \Theta = \Lambda) \geq \mathbb{P}(\Upsilon = P | \Theta = \Lambda) \geq \mathbb{P}(P_\Lambda \xrightarrow{f_m} P \mid \Theta = \Lambda).$$

Generally, we are interested in Λ with a particular dispersion function. This is formalised in Section 11.3 and it will be shown that it is enough to consider this version.

11.2.2 The coloured L -model versus the likelihood model

In the coloured L -model we consider a library of small R -coloured graphs, where each graph represents a (dispersion of a) ship. We wish to decompose a large R -coloured graph into these smaller graphs. Each H -type (the isomorphism class of R -coloured graph H) in L is a combination of radars on a graph. Types with the same colour set, but for which the graph is different, are considered to be the same type of ship, but observed in a different way. For example a K_1 -type with colours a and b is considered to be the same ship as a K_2 -type with colour a at one vertex and colour b at the other vertex, but the K_2 -type is somehow observed less accurately. We can think of this as dispersion of ships.

In the likelihood model we wish to decompose a larger R -coloured set V , but we consider a ship sitting at a single element v in V and each of its radars r is observed independently at random at some vertex $X_{v,r}$, which we called dispersion of radars. It seems like a reasonable assumption that properties of radars are independent of other radars, as well as the ship it is carried by, but there might be properties that are ship dependent as well. For example a small fishing vessel with a single radar is much less likely to turn this radar off than a big ferry which carries the same type of radar as a back up radar.

However, if we assume there is no dispersion (so no uncertainty or noise at all), both models are equivalent. In the coloured L -model this means all ships in L are K_1 -types and in the likelihood model this means that $\mathbb{P}(X_{v,r} = v) = 1$ for all $v \in V$ and $r \in R$.

11.3 The formalised likelihood problem

Definition 11.3.1. Let $P : V \rightarrow \mathbb{N}^R$ be an R -colouring and $\Lambda : V \rightarrow \mathbb{N}^L$ an L -colouring. For $v \in V$ and $r \in R$, let

$$c_P(v, r) = \{(v_r, i) : 1 \leq i \leq \rho(v, r)\}$$

be pairwise disjoint sets of new elements. Let

$$C(P) = \{c_P(v, r) : v \in V, r \in R\}.$$

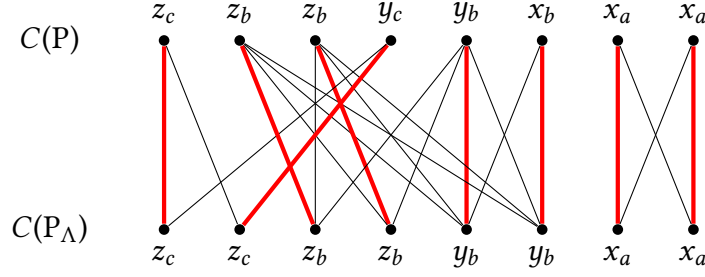


Figure 11.1: G_Λ from L -colouring Λ and R -colouring P in Example 5. The matching in red corresponds to a possible dispersion function.

To simplify notation, we will write $v_r \in C(P)$ for some $(v_r, i) \in C(P)$, where $i \in [\rho(v, r)]$. For $v_r \in C(P)$, $v_r^1 = v$ and $v_r^2 = r$. We say f is a **dispersion function** between Λ and P if f is a bijection between $C(P_\Lambda)$ and $C(P)$ such that if $f(v_r) = w_t$, then $r = t$ and $\mathbb{P}(X_{v,r} = w) > 0$. We will denote the event that Λ disperses to P according to f by $\Lambda \xrightarrow{f} P$.

Note that for $f(v_r) = w_t$, $(f(v_r))^1 = w$.

Lemma 11.3.2. *Given R -colouring P , an L -colouring Λ yields P with non-zero probability if and only if there exists a dispersion function between Λ and P and $\mathbb{P}(\Lambda) > 0$.*

Problem 6 (Formalised likelihood problem (FLH)). *Given a set of locations V and a set of colours R , an R -colouring $P : V \rightarrow \mathbb{N}^R$, a dispersion distribution X , a library L and a ship distribution Z , find an L -colouring $\Lambda : V \rightarrow \mathbb{N}^L$, which is plausible according to Z and a dispersion function f between Λ and P (if such a pair exists).*

Additionally we might like to know, if there is a solution (Λ, f) , can we find a pair that maximises

$$\mathbb{P} \left((\Theta = \Lambda) \cap \left(\Lambda \xrightarrow{f} \Upsilon \right) \mid \Upsilon = P \right)?$$

11.3.1 LHP and FLH

We assume that numbers are specified in unary, so we may take the size of an LHP or FLH instance to be $|C(P)|$. We assume that X and Z are stored as tables with an $O(1)$ look-up time. Recall that given $P : V \rightarrow \mathbb{N}^R$, we write $v_r \in C(P)$ for some $(v_r, i) \in C(P)$, where $i \in [\rho(v, r)]$. It is easy to see that $\text{LHP} \leq_p \text{FLH}$. More interestingly:

Lemma 11.3.3. $\text{FLH} \leq_p \text{LHP}$

Proof. The problem instance is the same for FLH and LHP, but for FLH, as well as finding an L -colouring Λ , we want to find a dispersion function f from Λ to P . We show that given a solution to LHP, that is Λ which realises P with non-zero probability, we can find a dispersion function f in polynomial time.

Let $G_\Lambda = (V_\Lambda, E_\Lambda)$ be a bipartite graph with vertex classes $C(P_\Lambda)$ and $C(P)$. For $v_r \in C(P_\Lambda), w_t \in C(P)$,

$$v_r w_t \in E_\Lambda \Leftrightarrow r = t \text{ and } \mathbb{P}(X_{v,r} = w) > 0.$$

See Fig. 11.1 for an example. It is easy to see that G_Λ can be constructed in $O(|C(P)|^2)$ time and that a perfect matching in G_Λ is equivalent to a dispersion function between Λ and P . A perfect matching in a bipartite graph can be found in $O(n^{2.5})$ time [69], where $n = |V_\Lambda| = O(|C(P)|)$, so $\text{FLH} \leq_p \text{LHP}$. $\square$

So it is enough to show the complexity of FLH, which indeed we do in Section 11.4. However, finding a maximum likelihood estimator or sampling is not the same for both problems.

11.3.2 Calculating the probability of a solution

Given R -colouring P , let Λ be an L -colouring and f a dispersion function between Λ and P . Then

$$\begin{aligned} \mathbb{P}\left((\Theta = \Lambda) \cap \left(\Lambda \xrightarrow{f} \Upsilon\right) \mid \Upsilon = P\right) &= \frac{\mathbb{P}\left((\Theta = \Lambda) \cap \left(\Lambda \xrightarrow{f} \Upsilon\right) \cap (\Upsilon = P)\right)}{\mathbb{P}(\Upsilon = P)} \\ &\propto \mathbb{P}\left((\Theta = \Lambda) \cap \left(\Lambda \xrightarrow{f} \Upsilon\right) \cap (\Upsilon = P)\right) \\ &= \mathbb{P}\left((\Theta = \Lambda) \cap \left(\Lambda \xrightarrow{f} P\right)\right) \\ &= \mathbb{P}(\Theta = \Lambda) \mathbb{P}\left(\Lambda \xrightarrow{f} P\right) \\ &\propto \left(\prod_{v \in V} \prod_{s \in L} p_{v,s}^{\lambda(v,s)}\right) \left(\prod_{v_r \in C(P_\Lambda)} \mathbb{P}\left(X_{v,r} = (f(v_r))^1\right)\right) \end{aligned} \tag{11.2}$$

11.4 The complexity of FLH

In this section we show that FLH is NP-hard in general and we discuss some tractable special cases.

11.4.1 FLH is NP-hard

Lemma 11.4.1. *FLH is NP-hard, even when L consists only of ships with exactly three (distinct) colours and $|V| = 1$.*

Proof. In this case there is no dispersion at all, so FLH is equivalent to finding an L -decomposition. Then FLH is NP-hard by Lemma 7.2.1. $\square$

Given R -colouring P , recall that we write $v_b \in C(P)$ to mean $(v_b, i) \in C(P)$, where $1 \leq i \leq \rho(v, r)$.

Lemma 11.4.2. *FLH is NP-hard, even if L consists only of ships with exactly three colours and $|R| = 1$ (so there is only one ship).*

Proof. We will show that $X3C \leq_p FLH$ (see Problem 4), by showing that any instance of X3C can be solved by solving a related instance of FLH. Let (X, C) be an instance of X3C and let V_C be a set of vertices, one for each set in C , that is, $V_C = \{v_e : e \in C\}$. Set $V = X \cup V_C$ and $R = \{r\}$, where r is a new element. Now let L consist of one ship s with exactly three copies of r . For the scope of this proof, set $X_{v,r} = X_v$. We set

$$\mathbb{P}(X_v = w) = \begin{cases} \frac{1}{3} & \text{if } v = v_e \in V_C \text{ and } w \in e, \\ 1 & \text{if } v \in X \text{ and } w = v, \\ 0 & \text{otherwise.} \end{cases}$$

and we set $p_{v,s} > 0$ for all $v \in V_C$ and zero otherwise. Now set

$$P = (\mathbf{1}_{v \in X} : v \in V, r \in R) = (\mathbf{1}_{v \in X} : v \in V),$$

that is, there is exactly one observation r at each $v \in X$. We show that there is an exact cover of X if and only if there is an L -colouring of V , which disperses to P with non-zero probability.

Let $C' \subseteq C$ be an exact cover of X and for $v \in V$, set $\lambda(v, s) = 1$ if $v = v_e$ for some $e \in C'$, and zero otherwise. so there is exactly one ship at each v_e for which $e \in C'$.

11.4. The complexity of FLH

For all $e \in C'$, where $e = \{x, y, z\}$, we let each radar at $v_e = u$ in P_Λ be observed at one distinct vertex in e . Then

$$\mathbb{P}(X_u = x)\mathbb{P}(X_u = y)\mathbb{P}(X_u = z) = \frac{1}{27} > 0,$$

and as $V(C') = X \simeq C(P)$, this is a valid dispersion. Finally $p_{v,s} > 0$ for all $v \in V_C$, so this is a solution to this FLH instance.

Now let (Λ, f) be a solution to this FLH instance. Then $\lambda(v, s) > 0$ means $p_{v,s} > 0$, which implies $v \in V_C$, as $p_{v,s} = 0$ for all $v \notin V_C$. For $v_e = u \in V_C$, let

$$C_u = \{x_r \in C(P) : \mathbb{P}(X_u = x) > 0\}.$$

Then $|C_u| = 3$ for all $u \in V_C$. Moreover for each $u \in V_C$

$$f : c_{P_\Lambda}(u, r) \rightarrow C_u,$$

so

$$3 = |C_u| \geq |c_{P_\Lambda}(u, r)| = \sum_{s \in L} \lambda(u, s) s_r = 3\lambda(u, s).$$

Therefore $\lambda(u, s) \in \{0, 1\}$. So there is either one ship at u , or no ship. Set

$$C' = \{e : \lambda(v_e, s) > 0\}.$$

Then $C' \subseteq C$ and C' is an exact cover of X , because f is a bijection between $C(P_\Lambda)$ and $C(P) \simeq X$, that is, there is exactly one set/ship that covers each $v_r \in C(P) \simeq X$. $\square$

11.4.2 Tractable cases

11.4.2.1 One colour per ship is trivial

If each ship consists of exactly one colour, it is always best to be greedy by estimating that each ship is exactly where it is most likely to be, given the observed position of its unique radar. In most cases the most likely position of a ship is where its radar (colour) is observed.

11.4.2.2 Two colours per ship is tractable

Suppose that for all $s \in L$

$$\sum_{r \in R} s_r = 2. \tag{11.3}$$

Definition 11.4.3. For $a, b \in R$, we let $s(a, b)$ be the set of ships in L with one radar a and one radar b , that is

$$s(a, b) = \begin{cases} \{s \in L : s_a = s_b = 1\} & \text{if } a \neq b, \\ \{s \in L : s_a = 2\} & \text{if } a = b. \end{cases}$$

Note that Eq. 11.3 implies that $s(a, b)$ has size at most one. Given R -colouring P , recall that we write $v_b \in C(P)$ to mean $(v_b, i) \in C(P)$, where $1 \leq i \leq \rho(v, r)$.

Definition 11.4.4. Given an R -colouring $P : V \rightarrow \mathbb{N}^R$, let $G_C(V_C, E_C)$ be an edge-weighted graph we shall call the **coupling graph**. Let

$$V_C = C(P)$$

$$E_C = \{u_a v_b : s(a, b) \in L \text{ and } \exists x \in V \text{ such that } \mathbb{P}(X_{x,a} = u) \mathbb{P}(X_{x,b} = v) p_{x,s(a,b)} > 0\},$$

and for $u_a v_b \in E_C$,

$$w(u_a, v_b) = \max_{x \in V} \log \left(\mathbb{P}(X_{x,a} = u) \mathbb{P}(X_{x,b} = v) p_{x,s(a,b)} \right).$$

See Fig 11.2 for an example.

Definition 11.4.5. For each $u_a v_b \in E_C$, let

$$t(u_a, v_b) \in \arg \max_{x \in V} \mathbb{P}(X_{x,a} = u) \mathbb{P}(X_{x,b} = v) p_{x,s(a,b)}.$$

In other words, if u_a and v_b would be two radar instances originating from the same ship, a most likely location of this ship is at $t(u_a, v_b)$. Note that we have no way to differentiate between locations in $\arg \max_{x \in V} \mathbb{P}(X_{x,a} = u) \mathbb{P}(X_{x,b} = v) p_{x,s(a,b)}$, so a random element from this set is chosen in some arbitrary deterministic way.

Let $t = t(u_a, v_b)$. Given a matching M in G_C , the weight $w(M)$ is

$$\begin{aligned} w(M) &= \sum_{u_a v_b \in M} w(u_a, v_b) \\ &= \sum_{u_a v_b \in M} \log \left(\mathbb{P}(X_{t,a} = u) \mathbb{P}(X_{t,b} = v) p_{t,s(a,b)} \right) \\ &= \sum_{u_a v_b \in M} \log \mathbb{P}(X_{t,a} = u) + \log \mathbb{P}(X_{t,b} = v) + \log p_{t,s(a,b)}. \end{aligned} \quad (11.4)$$

Definition 11.4.6. Given a matching M in G_C , let

$$\Lambda(M) = (\lambda(x, s) : x \in V, s \in L),$$

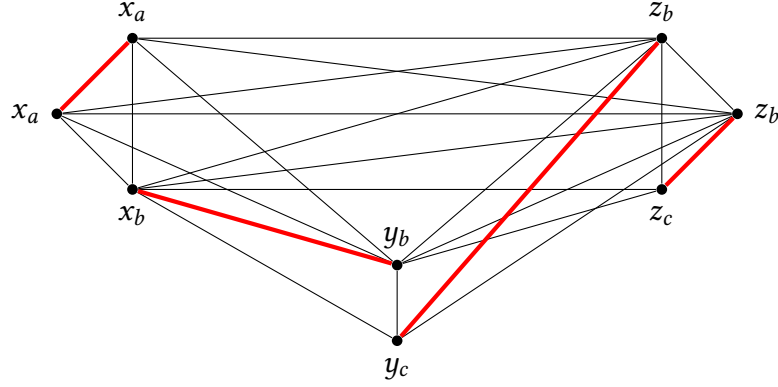


Figure 11.2: G_C from R -colouring P in Example 5. The matching in red corresponds to Λ .

where

$$\lambda(x, s) = \sum_{u_a v_b \in M} \mathbf{1}_{t(u_a, v_b)=x} \mathbf{1}_{s(a,b)=s}.$$

In other words $\Lambda(M)$ specifies a positioning of ships, in one-to-one correspondence with M . For each $u_a v_b \in M$, there is a ship $s(a, b)$ at vertex $t = t(u_a, v_b)$ in $\Lambda(M)$ and for the corresponding $t_a, t_b \in C(P_{\Lambda(M)})$, we set

$$\begin{aligned} f(t_a) &= u_a, \text{ and} \\ f(t_b) &= v_b, \end{aligned}$$

in the dispersion function f .

Lemma 11.4.7. *Given a perfect matching M in G_C , $(\Lambda(M), f)$ as defined above is a solution to the FLH instance.*

Proof. Note that f is a bijection between $C(P_{\Lambda(M)})$ and $C(P) = V_C = V(M)$ by construction. For each $u_a v_b \in M$ with $t = t(u_a, v_b)$, we have $f(t_a) = u_a$ and $f(t_b) = v_b$. As $M \subseteq E_C$, $u_a v_b \in E_C$. By considering the definition of E_C , we obtain

$$\mathbb{P}(X_{t,a} = u) \mathbb{P}(X_{t,b} = v) p_{t,s(a,b)} > 0. \quad (11.5)$$

So $\mathbb{P}(X_{t,a} = u), \mathbb{P}(X_{t,b} = v) > 0$ for all $t_a, t_b \in C(P_{\Lambda(M)})$ and $f(t_a), f(t_b) \in C(P)$. Therefore f is a valid dispersion function between $\Lambda(M)$ and P .

Moreover, for each ship $s(a, b)$ at vertex $t = t(u_a, v_b)$ in $\Lambda(M)$, there is a unique $u_a v_b \in M$. Eq. 11.5 implies $p_{t,s(a,b)} > 0$. This is true for each ship in Λ . Consequently $\mathbb{P}(\Lambda) > 0$. Thus $(\Lambda(M), f)$ is a solution to the FLH instance. $\square$

Lemma 11.4.8. *Given a perfect matching M in G_C ,*

$$\log \mathbb{P} \left((\Theta = \Lambda(M)) \cap \left(\Lambda(M) \xrightarrow{f} \Upsilon \right) \middle| \Upsilon = P \right) \propto w(M).$$

Proof. For $u_a v_b \in M$, set $t = t(u_a, v_b)$. Then by Eq. 11.2 we need to calculate

$$\log \mathbb{P}(\Theta = \Lambda(M)) + \log \mathbb{P} \left(\Lambda(M) \xrightarrow{f} P \right),$$

and

$$\begin{aligned} \log \mathbb{P} \left(\Lambda(M) \xrightarrow{f} P \right) &= \log \prod_{u_a v_b \in M} \mathbb{P}(X_{t,a} = u) \mathbb{P}(X_{t,b} = v) \\ &= \sum_{u_a v_b \in M} \log \mathbb{P}(X_{t,a} = u) + \log \mathbb{P}(X_{t,b} = v). \end{aligned} \quad (11.6)$$

Moreover,

$$\log \mathbb{P}(\Theta = \Lambda(M)) \propto \log \prod_{x \in V} \prod_{s \in L} p_{x,s}^{\lambda(x,s)} \quad (11.7)$$

$$= \log \prod_{u_a v_b \in M} p_{t,s(a,b)} \quad (11.8)$$

$$= \sum_{u_a v_b \in M} \log p_{t,s(a,b)}. \quad (11.9)$$

Adding Eq. 11.6 and 11.9 gives Eq. 11.4, as required. $\square$

Corollary 11.4.9. *Given a maximum weight perfect matching M in G_C , $(\Lambda(M), f)$ is a maximum likelihood estimator for the FLH instance.*

11.5 The two-time model

In this section we introduce the two-time model in relation to the two-time problem. We consider two instances of the likelihood problem with the same parameters. We imagine one instance to be the situation at time 1 and the other instance to be the situation at time 2. Between two points in time a ship s at vertex v moves to a random vertex $Y_{v,s}$ independently from other ships, according to some fixed distribution. Thus the distribution of ships at time 2 may be different from that at time 1.

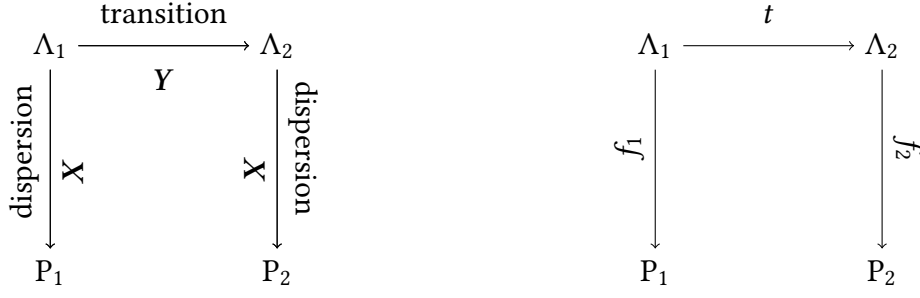


Figure 11.3: The two-time problem.

Definition 11.5.1. Let Y be a $V \times V \times L$ matrix we call the **transition distribution**:

$$Y = (\mathbb{P}(Y_{v,s} = w) : v \in V, w \in V, s \in L).$$

Suppose V is the vertex set of a graph G . We say that Y is a **G -transition distribution** if for each $v, w \in V$ and $s \in L$

$$\mathbb{P}(Y_{v,s} = x) > 0 \quad \Leftrightarrow \quad w \in \{v\} \cup \Gamma(v).$$

Definition 11.5.2. We let $\Theta_i : V \rightarrow \mathbb{N}^R$ be a **random L -colouring at time i** , for $i \in \{1, 2\}$, where Θ_1 follows the ship distribution, and Θ_2 is derived from Θ_1 according to the transition distribution

Problem 7 (Two-time problem (TTP)). Given a set of locations V and a set of colours R , a dispersion distribution X , a library L , a transition distribution Y , a ship distribution Z and R -colourings $P_1 : V \rightarrow \mathbb{N}^R$ and $P_2 : V \rightarrow \mathbb{N}^R$ at time 1 and 2 respectively, find L -colourings $\Lambda_1 : V \rightarrow \mathbb{N}^L$ and $\Lambda_2 : V \rightarrow \mathbb{N}^L$, such that Λ_1 is plausible according to Z , such that P_{Λ_1} disperses to P_1 and P_{Λ_2} disperses to P_2 with non-zero probability according to X and so that ships can move from their place in Λ_1 to a place in Λ_2 with non-zero probability, according to Y .

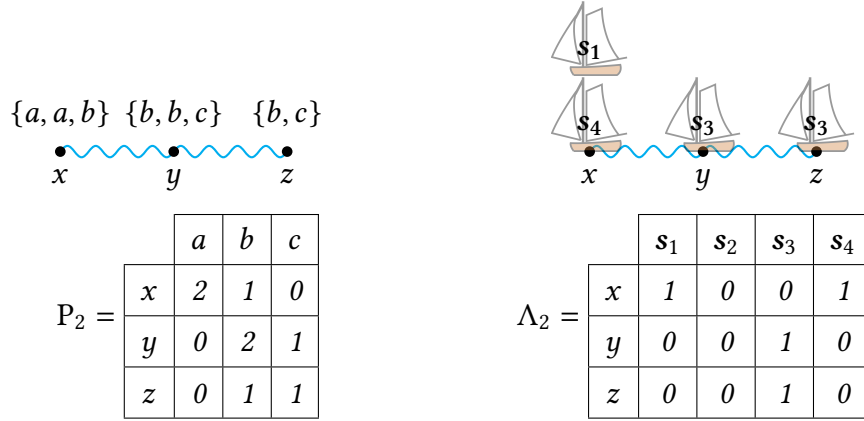
Moreover, we might ask, if there is a solution (Λ_1, Λ_2) , can we find maximum likelihood estimators $\hat{\Lambda}_1$ and $\hat{\Lambda}_2$ or sample from all Λ_1 and Λ_2 ? Fig 11.3 shows a diagram of the two-time problem.

Example 6 (TTP instance). Let V, R, L, X , and Z be as in Example 5 and for $v \in V, s \in L$, let

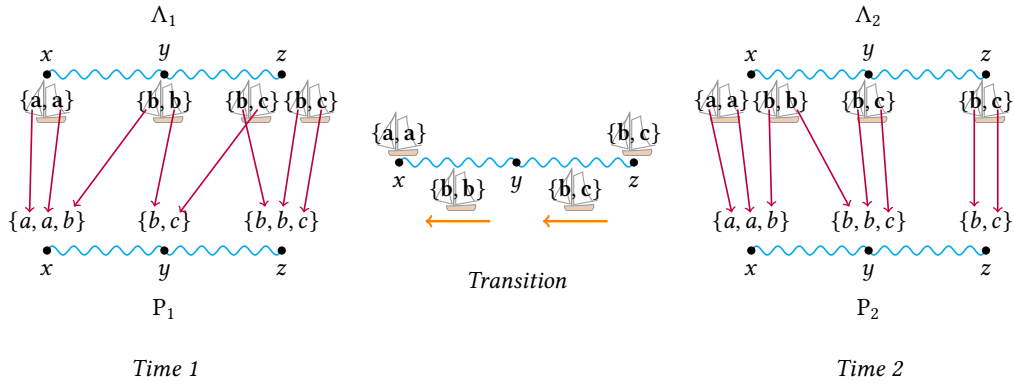
$$\mathbb{P}(Y_{v,s} = w) = \begin{cases} \frac{1}{2} & \text{if } w = v, \\ \frac{1}{2|\Gamma(v)|} & \text{if } w \in \Gamma(v), \\ 0 & \text{otherwise.} \end{cases}$$

Chapter 11. The likelihood and two-time models

Set $P_1 = P$ and $\Lambda_1 = \Lambda$ as in Example 5 and set



Then (Λ_1, Λ_2) is a solution to the above instance via the following transition: a ship s_1 at y moves to x , a ship s_3 at z moves to y and all other ships stay in the same place. Below is an L -colouring at time 1 and a dispersion (in purple) to R -colouring P_1 . Two ships move between time 1 and time 2 (in yellow). On the right is a new L -colouring Λ_2 and a dispersion (in purple) to R -colouring P_2 .



11.5.1 Calculating the probability of a solution

Given P_1 and P_2 , as we observe the scene at time 1 and 2 respectively, and possible L -colourings Λ_1 and Λ_2 ,

$$\begin{aligned}
 & \mathbb{P}((\Theta_1 = \Lambda_1) \cap (\Theta_2 = \Lambda_2) | (\Upsilon_1 = P_1) \cap (\Upsilon_2 = P_2)) \\
 &= \frac{\mathbb{P}((\Theta_1 = \Lambda_1) \cap (\Theta_2 = \Lambda_2) \cap (\Upsilon_1 = P_1) \cap (\Upsilon_2 = P_2))}{\mathbb{P}((\Upsilon_1 = P_1) \cap (\Upsilon_2 = P_2))} \\
 &\propto \mathbb{P}((\Theta_1 = \Lambda_1) \cap (\Theta_2 = \Lambda_2) \cap (\Upsilon_1 = P_1) \cap (\Upsilon_2 = P_2))
 \end{aligned}$$

Note that as before, the $\mathbb{P}((\Upsilon = P_1) \cap (\Upsilon_2 = P_2))$ term is constant.

11.6 The formalised two-time problem

Let us formalise the two-time problem, much as we did for the likelihood problem.

Definition 11.6.1. Let $\Lambda_1 : V \rightarrow \mathbb{N}^L$ and $\Lambda_2 : V \rightarrow \mathbb{N}^L$ be L -colourings and let

$$c_j(v, s) = \{(v_s, i) : 1 \leq i \leq \lambda_j(v, s)\} \quad \text{for } j \in \{1, 2\}$$

be a set of new elements and let all $c_j(v, s)$ be pairwise disjoint. For $j = 1, 2$, let

$$C_j = \{c_j(v, s) : v \in V, s \in L\}.$$

To simplify notation, we write $v_s \in C_j$ for some $(v_s, i) \in C_j$, where $1 \leq i \leq \lambda_j(v, s)$. For $v_s \in C_j$, let $v_s^1 = v$ and $v_s^2 = s$. We call $t : C_1 \rightarrow C_2$ a **transition function** between Λ_1 and Λ_2 if t is a bijection between C_1 and C_2 , such that if $t(v_s) = w_q$, then $s = q$ and $\mathbb{P}(Y_{v,s} = w) > 0$. We write $\Lambda_1 \xrightarrow{t} \Lambda_2$ for the event that ships moved from a place in Λ_1 to a place in Λ_2 according to t .

Note that for $t(v_s) = w_q$, $(t(v_s))^1 = w$.

Problem 8 (Formalised Two-Time Problem (FTT)). Given a set of locations V and a set of colours R , a dispersion distribution X , a library L , a transition distribution Y , a ship distribution Z and R -colourings $P_1 : V \rightarrow \mathbb{N}^R$ and $P_2 : V \rightarrow \mathbb{N}^R$ at time 1 and 2 respectively, find L -colourings $\Lambda_1 : V \rightarrow \mathbb{N}^L$ and $\Lambda_2 : V \rightarrow \mathbb{N}^L$, such that Λ_1 is plausible according to Z , dispersion functions f_1 from Λ_1 to P_1 and f_2 from Λ_2 to P_2 , and a transition function t from Λ_1 to Λ_2 .

Moreover, we might ask, if there are such (Λ_1, f_1) , (Λ_2, f_2) and t , can we find a tuple maximising

$$\mathbb{P} \left((\Theta_1 = \Lambda_1) \cap (\Theta_2 = \Lambda_2) \cap \left(\Lambda_1 \xrightarrow{f_1} P_1 \right) \cap \left(\Lambda_2 \xrightarrow{f_2} P_2 \right) \cap \left(\Lambda_1 \xrightarrow{t} \Lambda_2 \right) \mid (\Upsilon_1 = P_1) \cap (\Upsilon_2 = P_2) \right)?$$

Fig 11.3 shows a diagram of the maps in the formalised two-time problem.

11.6.1 TTP and FTT

It is easy to see that $TTP \leq_p FTT$. More interestingly we have:

Lemma 11.6.2. $FTT \leq_p TTP$.

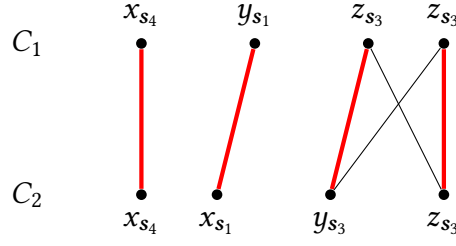


Figure 11.4: The graph G_{12} corresponding to $\Lambda = \Lambda_1$ in Example 5 and Λ_2 in Example 6. A possible transition is shaded in red.

Proof. FTT and TTP have the same problem instance. For FTT we require to find Λ_1 and Λ_2 , as well as dispersion functions f_1 from Λ_1 to P_1 and f_2 from Λ_2 to P_2 , and transition function t from Λ_1 to Λ_2 . We will show that, given a solution to TTP, that is Λ_1 and Λ_1 , which realise P_1 and P_2 with non-zero probability and so that ships can move from their place in Λ_1 at time 1 to a place in Λ_2 at time 2, we can find a solution to FTT in polynomial time.

We can find f_1 from Λ_1 to P_1 and f_2 from Λ_2 to P_2 as in the proof of Lemma 11.3.3. Let $G_{12} = (V_{12}, E_{12})$ be a bipartite graph with vertex classes C_1 and C_2 , as defined in Definition 11.6.1. For $v_s \in C_1, v'_{s'} \in C_2$, we set

$$v_s v'_{s'} \in E_{12} \Leftrightarrow s = s' \text{ and } \mathbb{P}(Y_{v_s} = v') > 0.$$

It is easy to see that G_{12} can be constructed in $O(|C_1|^2) = O(|C(P_1)|^2)$ time and that finding a perfect matching in G_{12} is equivalent to finding a transition function between Λ_1 and Λ_2 . A perfect matching can be found in $O(n^{2.5})$ time [69] and therefore $\text{FTT} \leq_p \text{TTP}$. $\square$

So it is enough to determine that the complexity of FTT, which indeed we will do in Section 11.7. Note that as for the likelihood problem, the optimisation and sampling problems of TTP and FTT are not equivalent.

11.6.2 Calculating the probability of a solution

Given (Λ_1, f_1) , (Λ_2, f_2) and t as a solution to a problem instance P_1 and P_2 , let A be the event

$$(\Theta_1 = \Lambda_1) \cap (\Theta_2 = \Lambda_2) \cap \left(\Lambda_1 \xrightarrow{f_1} P_1 \right) \cap \left(\Lambda_2 \xrightarrow{f_2} P_2 \right) \cap \left(\Lambda_1 \xrightarrow{t} \Lambda_2 \right).$$

Then

$$\begin{aligned}\mathbb{P}(A | (\Upsilon_1 = P_1) \cap (\Upsilon_2 = P_2)) &= \frac{\mathbb{P}(A \cap (\Upsilon_1 = P_1) \cap (\Upsilon_2 = P_2))}{\mathbb{P}((\Upsilon_1 = P_1) \cap (\Upsilon_2 = P_2))} \\ &\propto \mathbb{P}(A \cap (\Upsilon_1 = P_1) \cap (\Upsilon_2 = P_2)).\end{aligned}$$

Moreover

$$\mathbb{P}\left(\Upsilon_1 = P_1 \mid \Lambda_1 \xrightarrow{f_1} P_1\right) = 1 = \mathbb{P}\left(\Upsilon_2 = P_2 \mid \Lambda_2 \xrightarrow{f_2} P_2\right),$$

so

$$\mathbb{P}(A \cap (\Upsilon_1 = P_1) \cap (\Upsilon_2 = P_2)) = \mathbb{P}(A).$$

By independence,

$$\mathbb{P}(A) = \mathbb{P}\left(\Lambda_1 \xrightarrow{f_1} P_1\right) \mathbb{P}\left(\Lambda_2 \xrightarrow{f_2} P_2\right) \mathbb{P}\left((\Theta_1 = \Lambda_1) \cap (\Theta_2 = \Lambda_2) \mid \Lambda_1 \xrightarrow{t} \Lambda_2\right) \mathbb{P}\left(\Lambda_1 \xrightarrow{t} \Lambda_2\right),$$

where

$$\begin{aligned}\mathbb{P}\left((\Theta_1 = \Lambda_1) \cap (\Theta_2 = \Lambda_2) \mid \Lambda_1 \xrightarrow{t} \Lambda_2\right) &= \mathbb{P}\left(\Theta_2 = \Lambda_2 \mid (\Theta_1 = \Lambda_1) \cap \left(\Lambda_1 \xrightarrow{t} \Lambda_2\right)\right) \\ &\quad \mathbb{P}\left(\Theta_1 = \Lambda_1 \mid \left(\Lambda_1 \xrightarrow{t} \Lambda_2\right)\right) \\ &= \mathbb{P}(\Theta_1 = \Lambda_1),\end{aligned}$$

and

$$\mathbb{P}\left(\Lambda_1 \xrightarrow{t} \Lambda_2\right) = \prod_{v_s \in C_1} \mathbb{P}(Y_{v,s} = (t(v_s))^1).$$

11.7 The complexity of FTT

In this section we show that FTT is NP-hard already when all ships have exactly two colours and there is no movement. We show that FTT is tractable if all ships have exactly one colour.

11.7.1 One colour per ship is tractable

We may assume that for each $r \in R$,

$$\sum_{v \in V} \rho_1(v, r) = \sum_{v \in V} \rho_2(v, r),$$

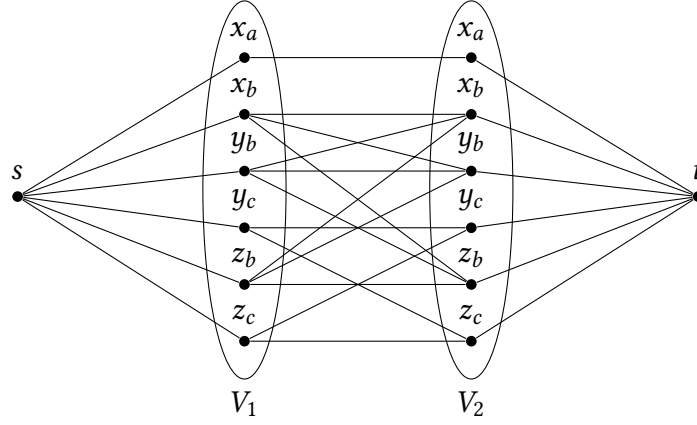


Figure 11.5: Example G_f with P_1 and P_2 from Example 6 and library $L = R = \{a, b, c\}$.

since otherwise there can be no solution. If there is exactly one colour per ship, then $L \simeq R$ and we can represent the problem as a (directed) bipartite flow graph $G_f = (V_f, E_f)$ with (new) disjoint vertex classes $V_1 = \{v_r : \rho_1(v, r) > 0\}$ and $V_2 = \{v_r : \rho_2(v, r) > 0\}$ and source s and sink t . For all $v_r \in V_1$, $sv_r \in E_f$ with capacity $\rho_1(v, r)$ and weight zero. For all $v_r \in V_2$, $v_rt \in E_f$ with capacity $\rho_2(v, r)$ and weight zero. For $u_a \in V_1, v_b \in V_2$, $u_av_b \in E_f$ if and only if $a = b$ and there are $x, y \in V$ such that

$$\mathbb{P}(X_{x,a} = u)\mathbb{P}(X_{y,a} = v)\mathbb{P}(Y_{x,a} = y)p_{x,a} > 0.$$

Such an edge has capacity $\min(\rho_1(u, a), \rho_2(v, a))$ and weight

$$\max_{x,y \in V} \log \left(\mathbb{P}(X_{x,a} = u)\mathbb{P}(X_{y,a} = v)\mathbb{P}(Y_{x,a} = y)p_{x,a} \right).$$

It is straightforward to see that finding a solution to the FTT problem is equivalent to finding an integral maximum weight maximum feasible flow in G_f . The volume must be

$$\sum_{r \in R} \sum_{v \in V} \rho_1(v, r),$$

since otherwise there is no solution, and the flow yields a transition function as required. Indeed if the maximum feasible flow also has maximum weight, then it yields a maximum likelihood transition. For an example see Fig. 11.5.

11.7.2 FTT is NP-hard

We can solve an instance of FLH by solving a related instance of FTT: set $P_1 = P_2$ and $\mathbb{P}(Y_{v,s} = v) = 1$ for all $v \in V$ and $s \in L$. Then a solution $(\Lambda_1, f_1), (\Lambda_2, f_2), t$, must be such that $\Lambda_1 = \Lambda_2$ and $(\Lambda_1 \xrightarrow{t} \Lambda_2)$ is the identity function. Because FLH is NP-hard, even if L is restricted to ships with three colours, FTT is as well, even when there is no movement.

11.7.3 FTT is NP-hard, even if all ships have two colours

Definition 11.7.1. Let X, Y , and Z be finite, disjoint sets, and let T be a subset of $X \times Y \times Z$. That is, T consists of triples (x, y, z) such that $x \in X, y \in Y$, and $z \in Z$. Now $M \subseteq T$ is a **3-dimensional matching** or **3D matching** if the following holds: for any two distinct triples $(x_1, y_1, z_1) \in M$ and $(x_2, y_2, z_2) \in M$, we have $x_1 \neq x_2, y_1 \neq y_2$, and $z_1 \neq z_2$. A 3D matching M is **perfect** if every element in $X \cup Y \cup Z$ appears in some $e \in M$.

Problem 9 (3-dimensional matching (3DM)). Given finite, disjoint sets X, Y , and Z such that $|X| = |Y| = |Z|$ and $T \subseteq X \times Y \times Z$, is there a perfect 3-dimensional matching?

3DM is NP-complete [48, 72].

Definition 11.7.2. A **balanced bipartite hypergraph** $G = (U, V, E)$ is a triple of two disjoint vertex sets U, V and a set of hyperedges $E \subseteq 2^{U \cup V}$. We assume $|U| = |V|$ and that $|e \cap U| = |e \cap V| > 0$ for all $e \in E$. We denote by $|e|$ the size of the hyperedge and by $v \in e$ that vertex v is incident with edge e .

Definition 11.7.3. Let $G = (U, V, E)$ be a bipartite hypergraph. A **hyper-assignment** in G is a subset $H \subseteq E$ of hyperedges, such that every $v \in U \cup V$ is contained in exactly one hyperedge.

Problem 10 (Hyper-Assignment Problem (HAP)). Given a balanced bipartite hypergraph G , find a hyper-assignment in G .

This is a minor improvement on a result of Borndörfer and Heismann [20].

Theorem 11.7.4. HAP is NP-hard, even for balanced hypergraphs $G = (U, W, E)$ for which $2 = |e \cap U| = |e \cap W|$ for all $e \in E$.

Proof. By reduction from 3DM. Let X, Y, Z be disjoint sets of vertices such that $|X| = |Y| = |Z| = n$ and let T be a set of triples $(x, y, z) \in X \times Y \times Z$. We create a related HAP instance, which has a solution if and only if this 3DM instance has a solution.

Let $W = \{w_1, w_2, \dots, w_n\}$ be a set of elements disjoint from $X \cup Y \cup Z$ and set $U = W \cup X$ and $V = Y \cup Z$. Then $|U| = |V| = 2n$. Now let $Q = \{(w, x, y, z) : w \in W, (x, y, z) \in T\}$, that is, for each triple (x, y, z) in T there is a quadruple (w, x, y, z) in Q for every $w \in W$. Then $|Q| = n|T|$ and each $e \in Q$ is such that $|e \cap U| = |e \cap V| = 2$.

Let $T' \subseteq T$ be a 3D-matching, then $|T'| = n$. Let f be a bijection between T' and W and let $Q' = \{(f(x, y, z), x, y, z) : (x, y, z) \in T'\}$. Then $Q' \subseteq Q$ and Q' is a hyper-assignment in G .

Let Q' be a hyper-assignment. Then $|Q'| = n$ and $Q' \subseteq Q$. Moreover, every $(a, b, c, d) \in Q$ has exactly one element from W, X, Y and Z in this order, so this is also true for Q' . Set $T' = \{(b, c, d) : (a, b, c, d) \in Q'\}$. As each edge (w, x, y, z) in Q is such that there is a $(x, y, z) \in T$, $T' \subseteq T$. Moreover, as Q' is a hyper-assignment, T' is a 3D matching. $\square$

Lemma 11.7.5. *FTT is NP-hard, even when R consists of only one colour r , L consists of one ship with two copies of r , and there is no movement.*

Proof. We will show that $\text{HAP} \leq_p \text{FTT}$, by showing that HAP can be solved by solving a related instance of FTT, for which R consists of only one colour r , L consists of one ship with two copies of r , and there is no movement.

Let $G = (U, W, E)$ be a balanced bipartite hypergraph, such that $2 = |e \cap U| = |e \cap V|$ for all $e \in E$. We transform this to an FTT instance as follows. Let r be a new element and set $R = \{r\}$ and let L consist of one ship s with two copies of r . Create a vertex v_e for each $e \in E$ and let $V_E = \{v_e : e \in E\}$. Set $V = U \cup W \cup V_E$ and for $v, w \in V$,

$$\mathbb{P}(X_{v,r} = w) > 0 \text{ if and only if } v = v_e \in V_E \text{ and } w \in e \text{ or } v = w \in U \cup W, \quad (11.10)$$

$$\mathbb{P}(Y_{v,s} = w) > 0 \text{ if and only if } v = w \in V, \quad (11.11)$$

$$p_{v,s} > 0 \text{ if } v \in V_E \text{ and } p_{v,s} = 0 \text{ otherwise.} \quad (11.12)$$

Now let

$$\rho_1(v, r) = \begin{cases} 1 & \text{if } v \in U, \\ 0 & \text{otherwise.} \end{cases} \quad \text{and} \quad \rho_2(v, r) = \begin{cases} 1 & \text{if } v \in W, \\ 0 & \text{otherwise.} \end{cases}$$

and set $P_1 = (\rho_1(v, r) : v \in V)$ and $P_2 = (\rho_2(v, r) : v \in V)$. We now construct a related hypergraph G_T and we show that there is a solution to this instance of FTT if and only if there is a hyper-assignment in G_T . We conclude the proof by showing that $G_T \simeq G$.

Set $G_T = (C(P_1), C(P_2), E_T)$, where $E_T \subseteq 2^{C(P_1) \cup C(P_2)}$. For $v_r, w_r \in C(P_1)$ and $a_r, b_r \in C(P_2)$, $\{v_r, w_r, a_r, b_r\} \in E_T$ if and only if there are $x, y \in V$ such that

$$\mathbb{P}(X_{x,r} = v) \mathbb{P}(X_{x,r} = w) \mathbb{P}(X_{y,r} = a) \mathbb{P}(X_{y,r} = b) \mathbb{P}(Y_{x,s} = y) p_{x,s} > 0. \quad (11.13)$$

It is easy to see that FTT has a solution if and only if G_T has a hyper-assignment, as each edge $e \in E_T$ corresponds to a ship at time 1 and time 2, two feasible dispersions and a feasible transition between them. Now it is left to show that $G_T \simeq G$.

We have

$$C(P_1) = \{(v_r, i) : 1 \leq i \leq \rho_1(v, r), v \in V, r \in R\} = \{(v_r, 1) : v \in U\} \simeq U, \quad \text{and similarly} \\ C(P_2) = \{(v_r, 1) : v \in W\} \simeq W.$$

As $R = \{r\}$ and there is at most one colour at each vertex, for the scope of this proof we write v for $(v_r, 1) \in C(P_j)$, $j \in [2]$, and X_v for $X_{v,r}$. It is left to show that $E = E_T$.

Suppose $e \in E$ and $e = \{v, w, a, b\}$. Then

$$\mathbb{P}(X_{v_e} = v), \mathbb{P}(X_{v_e} = w), \mathbb{P}(X_{v_e} = a), \mathbb{P}(X_{v_e} = b) > 0,$$

by (11.10). Moreover $\mathbb{P}(Y_{v_e,s} = v_e) > 0$ by (11.11) and $p_{v_e,s} > 0$ by (11.12), so

$$\mathbb{P}(X_{v_e} = v) \mathbb{P}(X_{v_e} = w) \mathbb{P}(X_{v_e} = a) \mathbb{P}(X_{v_e} = b) \mathbb{P}(Y_{v_e,s} = e) p_{v_e,s} > 0,$$

and $e \in E_T$ by (11.13).

Now suppose $e \in E_T$ and $e = \{v, w, a, b\}$, then by (11.13) there exist $x, y \in V$ such that

$$\mathbb{P}(X_x = v) \mathbb{P}(X_x = w) \mathbb{P}(X_y = a) \mathbb{P}(X_y = b) \mathbb{P}(Y_{x,s} = y) p_{x,s} > 0.$$

Then $x = y$ by (11.11) and $x \in V_E$ by (11.12). Finally $x = v_e$ by (11.10), so $\{v, w, a, b\} \in E$. Therefore $E_T = E$. This concludes the proof. $\square$

	1 colour per ship	2 colours per ship	3 colours per ship
FLH	Trivial (Section 11.4.2)	Perfect matching (Lemma 11.4.7)	NP-hard (Lemma 11.4.2)
FTT	Maximum flow (Section 11.7.1)	NP-hard (Lemma 11.7.5)	NP-hard (Section 11.7.2)

Table 11.1: Overview of the complexity of different FLH and FTT instances.

11.8 Conclusion

We have seen that $\text{FLH} \leq_p \text{LHP}$ and $\text{FTT} \leq_p \text{TTP}$. Table 11.1 summarises the complexity results of this chapter. The green background indicates a tractable special case and the red background indicate when the problem is NP-hard to solve already.

Moreover, the tractable cases can be extended to find a maximum likelihood estimator. A maximum weight perfect matching in the coupling graph is equivalent to the maximum likelihood extension of FLH with 2 colours per ship, as shown in Lemma 11.4.8. A maximum weight maximum flow is equivalent to the maximum likelihood extension of FTT with 1 colour per ship, as described in Section 11.7.1. This concludes the discussion of modelling the reverse radar problem.

Chapter 12 summarises the main results in this thesis and discusses some possible future directions.

Art is never finished, only abandoned.

Leonardo da Vinci (1452–1519)

12

Concluding remarks

This chapter briefly summarises the main results of this thesis and outlines some possible future directions for addressing the problems discussed in this thesis.

12.1 Colour degree matrices of graphs with few cycles

This thesis describes simple necessary and sufficient conditions for a colour demand matrix to be realisable by a c -edge-coloured forest, for the monochromatic case ($c = 1$), as well as for general c . Next we focus on colour degree matrices of graphs with at most one cycle. Necessary and sufficient conditions are given. We proceed by showing analogous results for multi-graphs and pseudoforests with at most k cyclic components, by applying similar arguments as for graphs with at most one cycle.

A natural next step would be to consider a more general class of allowed realisations, for example graphs with at most two cycles. To the best of the author's knowledge, this has as of yet not been studied as a class of graphs *an sich*, but general necessary and sufficient conditions for demand sequences to be realisable are known [38, 53, 58]. Many people have suggested to investigate 2-trees as an extension, but our current approach would not work for those, for it relies on having $n + O(1)$ edges, as well as few cycles, neither of which are true for 2-trees. Additionally we

might consider, what happens if we relax condition (U2) to $t(I) \leq 2s(I) + 2$ for each $I \subseteq [c]$.

Another direction could be to consider directed graphs. This is already a well-studied case for monochromatic degree sequence problems [46, 54, 96]. Considering directed graphs would allow for a more direct link with for example discrete tomography. In fact, it is not hard to verify that Theorem 4.2.1 and 4.2.3 may be generalised to directed graphs in the obvious way, but the same does not hold for graphs with at most one cycle, as exemplified by the monochromatic example below Lemma 5.8.2.

12.2 Modelling the reverse radar problem

In this thesis several models for the reverse radar problem are presented: the coloured L -model, which the coloured $\{K_1, K_2\}$ -model is a special case of, the likelihood model, and the two-time model. The likelihood model generalises the notion of uncertainty and the two-time model extends the likelihood model to two points in time. These models are a study of the combinatorial properties of the problem, something that has not been investigated before. Although they are simplifications of the reverse radar problem, restricting the platform structure, removing time and in some special cases making it 1-dimensional, they bring some fundamental characteristics of the problem to light. It was shown that the complexity of the problem is dependent on the number of radars each platform carries. In addition to that, tractable problem instances were discussed and new algorithms were described. What's more, some of these instances were shown to be equivalent to finding matchings, a well-studied problem, for which fast algorithms are known. Others were shown to be equivalent to flow problems, for which many polynomial time algorithms are known as well.

12.3 Future directions for Thales UK

It has been shown that the reverse radar problem and the two-time problem are NP-hard to solve in general. Some tractable special cases have been found and solved, but these cases do not seem to have many relevant counterparts in the real world. A natural next step would be to explore approximation algorithms or heuristic algorithms. There are many well-known approximation and heuristic algorithms [28] and the combinatorial properties of the problems established in this thesis could be helpful for selecting the most appropriate algorithms. Another way forward is to

try and establish more definite properties of problem instances, so that these can be used in algorithmic approaches.

This thesis has highlighted many combinatorial aspects of the reverse radar and two-sensor problem, therefore it seems valuable to have people with a working knowledge of this field working on this problem, as well as on other related problems. A close(r) collaboration between the ‘mathematicians’ and the ‘engineers’ could be very fruitful, because they approach problems from such different angles. In particular, it could be beneficial to have someone with a background in complexity theory work with the Thales engineers on a periodic basis. S/he can help distinguish which problems are hopeless to solve precisely* and should be approximated, and which problems may be equivalent to some other well-known problems. This could save a lot of time working on approaches that are doomed to fail, or that have been solved already.

*or point out that the engineer has just shown that $P=NP$ and can thus cash in \$1 million.

Bibliography

- [1] Report from Thales-OCIAM workshop on a graph optimisation problem for signal sorting. Technical report, OCIAM, 2009. Subject to NDA.
- [2] Report on Thales-OCIAM graph algorithms workshop. Technical report, OCIAM, 2011. Subject to NDA.
- [3] Rotterdam Class Landing Platform Dock (LPD), Netherlands, 2015. URL <http://www.naval-technology.com/projects/rotterdam/>. Online; accessed 16 July 2015.
- [4] R. Ahuja, T. Magnanti, and J. Orlin. *Network flows: theory, algorithms, and applications*. Englewood Cliffs: Prentice Hall, 1st edition, 1993.
- [5] T. Armour. Emitter-platform association. Technical report, The University of Warwick and The Knowledge Transfer Network for Industrial Mathematics and EPSRC, 2010. URL https://www.maths.nottingham.ac.uk/personal/jb/selexReport2010_final.pdf. [Online; accessed 09-July-2012].
- [6] S. Arnborg, D. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, April 1987. ISSN 0196-5212. doi:10.1137/0608024.
- [7] F. Bäßler. Über eine spezielle Klasse Euler’scher Graphen. *Commentarii Mathematici Helvetici*, 27(1):81–100, 1953. ISSN 0010-2571. doi:10.1007/BF02564555.
- [8] L. Beineke and E. Schmeichel. Degrees and cycles in graphs. *Annals of the New York Academy of Sciences*, 319(1):64–70, 1979. ISSN 1749-6632. doi:10.1111/j.1749-6632.1979.tb32774.x.
- [9] C. Bentz, M.-C. Costa, C. Picouleau, B. Ries, and D. de Werra. Degree-constrained edge partitioning in graphs arising from discrete tomography. *Journal of Graph Algorithms and Applications*, 13(2):99–118, 2009. ISSN 1526-1719.
- [10] C. Berge. Two theorems in graph theory. *Proceedings of the National Academy of Sciences*, 43(9):842–844, 1957. URL <http://www.pnas.org/content/43/9/842.short>.
- [11] A. Berger. *Directed Degree Sequences*. PhD thesis, Martin-Luther-Universität Halle-Wittenberg, October 2011.

Bibliography

- [12] H. Bodlaender. Dynamic programming on graphs with bounded treewidth. In Timo Lepistö and Arto Salomaa, editors, *Automata, Languages and Programming*, volume 317 of *Lecture Notes in Computer Science*, pages 105–118. Springer Berlin Heidelberg, 1988. ISBN 978-3-540-19488-0. doi:10.1007/3-540-19488-6_110.
- [13] H. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–23, 1993. URL <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.18.8755>.
- [14] H. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 226–234, New York, NY, USA, 1993. ACM. ISBN 0-89791-591-7. doi:10.1145/167088.167161.
- [15] H. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1–2):1 – 45, 1998. ISSN 0304-3975. doi:10.1016/S0304-3975(97)00228-4.
- [16] F. Boesch and F. Harary. Unicyclic realizability of a degree list. *Networks*, 8(2): 93–96, 1978. ISSN 1097-0037. doi:10.1002/net.3230080202.
- [17] B. Bollobás. *Extremal Graph Theory*, chapter 2, pages 50–101. Dover, 1st edition, 1978. ISBN 0-486-43596-2.
- [18] B. Bollobás. *Modern Graph Theory*, chapter 1, pages 1–37. Springer New York, 1st edition, 1998. ISBN 0-387-98488-7.
- [19] A. Bondy and U. Murty. *Graph Theory*, chapter 1, pages 1–38. Graduate Texts in Mathematics. Springer London, 1st edition, 2008. ISBN 9781846289705.
- [20] R. Borndörfer and O. Heismann. The hypergraph assignment problem. *Discrete Optimization*, 15:15–25, 2015.
- [21] A. Busch, M. Ferrara, S. Hartke, M. Jacobson, H. Kaul, and D. West. Packing of graphic n -tuples. *Journal of Graph Theory*, 70(1):29–39, 2012. ISSN 1097-0118. doi:10.1002/jgt.20598.
- [22] J. Carroll and G. Isaak. Degree matrices realized by edge-colored forests. Online; accessed 03-July-2012, October 2009. URL <http://www.lehigh.edu/~gi02/ecforest.pdf>.
- [23] K. Chowdhary. *Fundamentals of discrete mathematical structures*, chapter 3, pages 37–60. PHI Learning, 2nd edition, 2012. ISBN 9788120345065.
- [24] M. Chrobak and C. Dürr. Reconstructing polyatomic structures from discrete x-rays: NP-completeness proof for three atoms. In Luboš Brim, Jozef Gruska, and Jiří Zlatuška, editors, *Mathematical Foundations of Computer Science 1998*, volume 1450 of *Lecture Notes in Computer Science*, pages 185–193. Springer Berlin Heidelberg, 1998. ISBN 978-3-540-64827-7. doi:10.1007/BFb0055767.

- [25] Clay Mathematics Institute. The millennium prize problems. Online; accessed 03-July-2015, June 2015. URL <http://www.claymath.org/millennium-problems/millennium-prize-problems>.
- [26] S. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd annual ACM symposium on Theory of computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM. doi:10.1145/800157.805047.
- [27] S. Cook. The P versus NP problem. In *Clay Mathematical Institute; The Millennium Prize Problems*, 2000.
- [28] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2007. ISBN 0-262-03293-7.
- [29] M.-C. Costa, D. de Werra, C. Picouleau, and D. Schindl. A solvable case of image reconstruction in discrete tomography. *Discrete Applied Mathematics*, 148(3):240 – 245, 2005. ISSN 0166-218X. doi:10.1016/j.dam.2005.03.006.
- [30] G. Dantzig and D. Fulkerson. On the max flow min cut theorem of networks. *RAND Corporation*, 1955.
- [31] G. Dantzig and D. Fulkerson. On the min-cut max-flow theorem in networks. *Annals of Mathematical Study*, 38:215–222, 1956.
- [32] P. Davies. Emitter to platform association CASE studentship. Powerpoint, February 2011.
- [33] D. de Werra, M.-C. Costa, C. Picouleau, and B. Ries. On the use of graphs in discrete tomography. *4OR*, 6(2):101–123, 2008. ISSN 1619-4500. doi:10.1007/s10288-008-0077-5.
- [34] R. Diestel. *Graph Theory*. Springer, 3rd edition, 2005. ISBN 3-540-26182-6.
- [35] C. Dürr, F. Guíñez, and M. Matamala. Reconstructing 3-colored grids from horizontal and vertical projections is NP-hard: A solution to the 2-atom problem in discrete tomography. *SIAM Journal on Discrete Mathematics*, 26(1): 330–352, 2012. doi:10.1137/100799733.
- [36] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Math*, 17:449–467, 1965. doi:10.4153/CJM-1965-045-4.
- [37] D. Eppstein. Partial 3-tree forbidden minors, Wikimedia Commons. Online; accessed 01-August-2014, November 2007. URL http://commons.wikimedia.org/wiki/File:Partial_3-tree_forbidden_minors.svg.
- [38] P. Erdős and T. Gallai. Gráfok előírt fokszámú pontokkal. *Matematikai Lapok*, 11:264–274, 1960.
- [39] J. Erickson. Treewidth. Online; accessed 09-January-2012, 2009. URL <http://compgeom.cs.uiuc.edu/~jeffe/teaching/comptop/notes/treewidth.pdf>. Lecture Notes.

Bibliography

- [40] L. Euler. Solutio problematis ad geometriam situs pertinentis (E53). *Academiae Scientiarum Imperialis Petropolitanae*, 8:128–140, 1736.
- [41] K. Farmer. Emitter identification algorithm for Windows NT ESM display version 3. Technical report, Thales Proprietary, February 2002. Working Paper.
- [42] M. Ferrara. Some problems on graphic sequences. *Graph Theory Notes of New York*, 64:19–25, 2013.
- [43] J. Fiala. Introduction to graph minors and treewidth. Online; accessed 10-January-2012, August 2007. URL <http://kam.mff.cuni.cz/~fiala/tw.pdf>. Lecture Notes.
- [44] L. Ford and D. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, February 1956.
- [45] D. Fulkerson, A. Hoffman, and M. McAndrew. Some properties of graphs with multiple edges. *Canadian Journal of Mathematics*, 17:166–177, 1965.
- [46] D. Gale. A theorem on flows in networks. *Pacific Journal of Mathematics*, 7(2):1073–1082, 1957. URL <http://projecteuclid.org/euclid.pjm/1103043501>.
- [47] R. Gardner, P. Gritzmann, and D. Prangenberg. On the computational complexity of reconstructing lattice sets from their X-rays. *Discrete Mathematics*, 202(1–3):45 – 71, 1999. ISSN 0012-365X. doi:10.1016/S0012-365X(98)00347-1.
- [48] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1st edition, 1979. ISBN 0-716-71044-7.
- [49] M. Gavish and A. Weiss. Performance analysis of bearing-only target location algorithms. *Transactions on aerospace and electronic systems*, 28(3):817 – 828, jul 1992. doi:10.1109/7.256302.
- [50] J. Guerci, R. Goetz, and J. DiModica. Passive emitter location system, April 1995. URL <https://www.google.com/patents/US5406291>. US Patent 5,406,291.
- [51] F. Guíñez. *Disjoint realisations of degree sequences and some applications to discrete tomography*. PhD thesis, Universidad de Chile, 2009.
- [52] F. Guíñez, M. Matamala, and S. Thomassé. Realizing disjoint degree sequences of span at most two: A tractable discrete tomography problem. *Discrete Applied Mathematics*, 159(1):23 – 30, 2011. ISSN 0166-218X. doi:10.1016/j.dam.2010.09.011.
- [53] S. Hakimi. On realizability of a set of integers as degrees of the vertices of a linear graph. I. *Journal of the Society for Industrial and Applied Mathematics*, 10(3):pp. 496–506, 1962. ISSN 03684245. URL <http://www.jstor.org/stable/2098746>.

- [54] S. Hakimi. On the degrees of the vertices of a directed graph. *Journal of the Franklin Institute*, 279(4):290 – 308, 1965. ISSN 0016-0032. doi:10.1016/0016-0032(65)90340-6. URL <http://www.sciencedirect.com/science/article/pii/0016003265903406>.
- [55] M. Halappanavar. *Algorithms for vertex-weighted matching in graphs*. PhD thesis, Old Dominion University, May 2009. URL <http://hpc.pnl.gov/people/hala/files/HalappanavarThesisVertexWtdMatching.pdf>.
- [56] P. Hall. On representatives of subsets. *Journal of the London Mathematical Society*, s1-10(1):26–30, 1935. doi:10.1112/jlms/s1-10.37.26.
- [57] F. Harary. *Graph Theory*, page 62. Addison-Wesley Publishing Company, 1st edition, 1969. ISBN 0-201-41033-8.
- [58] V. Havel. Poznámka o existenci konečných grafů. *Časopis pro pěstování matematiky*, 80(4):477–480, 1955. URL <http://eudml.org/doc/19050>.
- [59] D. Hefetz and U. Wagner. Dynamic programming on trees. Online; accessed 15-January-2015, October 2009. URL <http://www.ti.inf.ethz.ch/ew/lehre/GA09/lec-treewidth.pdf>.
- [60] P. Heggernes. Treewidth, partial k-trees, and chordal graphs. *Partial curriculum in INF334-Advanced algorithmical techniques, Department of Informatics, University of Bergen, Norway*, 2005.
- [61] G. Herman and A. Kuba. *Discrete tomography: Foundations, algorithms, and applications*, chapter 1, pages 3–34. Springer Science & Business Media, 1st edition, 1999.
- [62] G. Herman and A. Kuba. *Advances in discrete tomography and its applications*, volume 1. Springer, 1st edition, 2007. ISBN 0-817-63614-5.
- [63] I. Hicks, A. Koster, and E. Kolotoglu. Branch and tree decomposition techniques for discrete optimization. *TutORials in Operation Research: INFORMS–New Orleans*, 2005:1–29, 2005.
- [64] A. Hillebrand. Anne @ Thales and all the things I learned there. Technical report, University of Oxford, April 2012.
- [65] A. Hillebrand. Summary meeting 13-03-13. March 2013.
- [66] A. Hillebrand and C. McDiarmid. Finding edge-coloured realisations of graphs with at most one cycle. 2015+. In preparation.
- [67] A. Hillebrand and C. McDiarmid. Colour degree matrices of graphs with at most one cycle. *Discrete Applied Mathematics Special Issue: Graph Theory: ICGT 2014*, 2015. doi:10.1016/j.dam.2015.07.025. Accepted.

Bibliography

- [68] S. Holzer. Bounded treewidth. Online; accessed 15-March-2012, April 2008. URL http://www14.informatik.tu-muenchen.de/.../1/holzer/Holzer_Paper.pdf.
- [69] J. Hopcroft and R. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973. doi:10.1137/0202019.
- [70] JabRef Development Team. *JabRef*, 2014. URL <http://jabref.sf.net>.
- [71] F. Jane. *Jane's All the World's Fighting Ships, 1898*. David & Charles, 1st edition, 1898. ISBN 9780715344767.
- [72] R. Karp. Reducibility among Combinatorial Problems. *Complexity of Computer Computations: Proc. of a Symp. on the Complexity of Computer Computations*, pages 85–103, 1972.
- [73] H. Kierstead, A. Kostochka, and G. Yu. Extremal graph packing problems: Ore-type versus Dirac-type. *Surveys in combinatorics*, 365:113–135, 2009.
- [74] D. Kleitman and D. Wang. Algorithms for constructing graphs and digraphs with given valences and factors. *Discrete Mathematics*, 6(1):79 – 88, 1973. ISSN 0012-365X. doi:[http://dx.doi.org/10.1016/0012-365X\(73\)90037-X](http://dx.doi.org/10.1016/0012-365X(73)90037-X).
- [75] V. Kolmogorov. Blossom V: a new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation*, 1:43–67, 2009. doi:10.1007/s12532-009-0002-8.
- [76] A. Krause. Bounded treewidth graphs—a survey. *German Russian Winter School St. Petersburg, Russia*, 2003.
- [77] L. Lovász and D. Plummer. *Matching Theory*. Annals of Discrete Mathematics 29, 1st edition, 1986. ISBN 0-080-87232-8.
- [78] V. Menon. On the existence of trees with given degrees. *Sankhyā: The Indian Journal of Statistics, Series A*, pages 63–68, 1964.
- [79] S. Micali and V. Vazirani. An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs. In *Proc. 21st Annual IEEE Symposium in Foundation of Computer Science*, pages 17–27, 1980.
- [80] Miym. Maximal-matching, Wikimedia Commons. Online; Accesed 22-July-2015, February 2009. URL <https://commons.wikimedia.org/wiki/File:Maximal-matching.svg#/media/File:Maximal-matching.svg>.
- [81] T. Nagao and N. Ueda. NP-completeness results for nonogram via parsimonious reductions. Technical report, Tokyo Institute of Technology, 1996.
- [82] S. Noble. Evaluating a weighted graph polynomial for graphs of bounded tree-width. *Electronic Journal of Combinatorics*, 16, 2009. URL http://www.combinatorics.org/Volume_16/PDF/v16i1r64.pdf.

- [83] Encyclopædia Britannica Online. Combinatorics, 2013. URL <http://www.britannica.com/EBchecked/topic/127341/combinatorics>. [Online; accessed 15-February-2013].
- [84] Encyclopædia Britannica Online. Radar, 2013. URL <http://www.britannica.com/EBchecked/topic/488278/radar>. [Online; accessed 15-February-2013].
- [85] O. Ore. Graphs and matching theorems. *Duke Mathematical Journal*, 22(4): 625–639, 1955.
- [86] O. Ore. Graphs and subgraphs. *Transactions of the American Mathematical Society*, 84:109–136, 1957.
- [87] O. Ore. Graphs and subgraphs, II. *Transactions of the American Mathematical Society*, 93(2):185–204, 1959. ISSN 00029947. URL <http://www.jstor.org/stable/1993450>.
- [88] C. Papadimitriou. *Computational complexity*. Addison-Wesley, 1st edition, 1994.
- [89] N. Robertson and P. Seymour. Graph minors. I. Excluding a forest. *Journal of Combinatorial Theory, Series B*, 35(1):39 – 61, 1983. ISSN 0095-8956. doi:10.1016/0095-8956(83)90079-5.
- [90] N. Robertson and P. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309 – 322, 1986. ISSN 0196-6774. doi:10.1016/0196-6774(86)90023-4.
- [91] N. Robertson and P. Seymour. Graph minors. X. Obstructions to tree-decomposition. *Journal of Combinatorial Theory, Series B*, 52(2):153 – 190, 1991. ISSN 0095-8956. doi:10.1016/0095-8956(91)90061-N.
- [92] N. Robertson and P. Seymour. Graph minors. XX. Wagner’s conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325 – 357, 2004. ISSN 0095-8956. doi:10.1016/j.jctb.2004.08.001. Special Issue Dedicated to Professor W.T. Tutte.
- [93] N. Robertson and P. Seymour. Graph minors. XXI. Graphs with unique linkages. *Journal of Combinatorial Theory, Series B*, 99(3):583 – 616, 2009. ISSN 0095-8956. doi:10.1016/j.jctb.2008.08.003.
- [94] N. Robertson and P. Seymour. Graph minors XXIII. Nash-Williams’ immersion conjecture. *Journal of Combinatorial Theory, Series B*, 100(2):181–205, March 2010. ISSN 0095-8956. doi:10.1016/j.jctb.2009.07.003.
- [95] C. Rose. Method of passively estimating an emitter’s position and velocity using bearings-only without requiring observer acceleration, March 2004. URL <http://www.google.com.uy/patents/US6714155>. US Patent 6,714,155.

Bibliography

- [96] H. Ryser. Combinatorial properties of matrices of zeros and ones. *Canadian Journal of Mathematics*, 9:371–377, 1957. ISSN 0008-414X.
- [97] A. Satyanarayana and L. Tung. A characterization of partial 3-trees. *Networks*, 20(3):299–322, 1990. ISSN 1097-0037. doi:10.1002/net.3230200304.
- [98] S. Saunders. *Jane’s Fighting Ships 2014-2015: Yearbook*. Jane’s Fighting Ships. Jane’s Information Group, 1st edition, 2014. ISBN 9780710631015.
- [99] M. Schocker. Über Graphen mit vorgegebenen Valenzen. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 69(1):265–270, 1999. ISSN 0025-5858. doi:10.1007/BF02940878.
- [100] A. Schrijver. *Combinatorial Optimization, Volume A: Polyhedra and Efficiency*, chapter 10, pages 148–168. Springer-Verlag NY, 1st edition, 2003. ISBN 3-540-44389-4.
- [101] T. Seacrest. *Packings and realizations of degree sequences with specified substructures*. PhD thesis, University of Nebraska at Lincoln, 2011.
- [102] J. Senior. Partitions and their representative graphs. *American Journal of Mathematics*, 73(3):pp. 663–689, 1951. ISSN 00029327. URL <http://www.jstor.org/stable/2372318>.
- [103] J. Senior. Unimerism. *The Journal of Chemical Physics*, 19(7):865–873, 1951. doi:<http://dx.doi.org/10.1063/1.1748398>.
- [104] G. Sierksma and H. Hoogeveen. Seven criteria for integer sequences being graphic. *Journal of Graph theory*, 15(2):223–231, 1991.
- [105] M. Sipser. *Introduction to the Theory of Computation*, pages 165–211, 245–264. Thomson Course Technology, 2nd edition, 2006. ISBN 0-619-21764-2.
- [106] T. Spencer and E. Mayr. Node weighted matching. In *Automata, Languages and Programming*, pages 454–464. Springer, 1984.
- [107] G. Stimson. *Introduction to Airborne Radar*, pages 3–45. SciTech Publishing, 2nd edition, 1998. ISBN 0-852-96942-2.
- [108] A. Stove. Unclassified ESM scenario - version 2. Technical report, Thales Proprietary, 2009.
- [109] A. Stove. ESM scenario generation and analysis of compressive sensing. Technical report, Thales Proprietary, 2012. Subject to NDA.
- [110] R. Thomas. Tree-decompositions of graphs. Online; accessed 10-January-2012, June 1996. URL <http://people.math.gatech.edu/~thomas/tree.ps>. Lecture Notes.

- [111] D. Torrieri. Statistical theory of passive location systems. In I. Cox and G. Wilfong, editors, *Autonomous Robot Vehicles*, pages 151–166. Springer New York, 1990. ISBN 978-1-4613-8999-6. doi:10.1007/978-1-4613-8997-2_13.
- [112] J. van Leeuwen. *Handbook of theoretical computer science*, chapter Graph Algorithms, pages 525–631. Elsevier, 1st edition, 1990. ISBN 0-444-88071-2.
- [113] F. van Schelle. Report of ECS platform association algorithm PhD at Oxford. Technical report, Thales Proprietary, March 2011.
- [114] V. Vazirani. A theory of alternating paths and blossoms for proving correctness of the $O(\sqrt{V}E)$ general graph maximum matching algorithm. *Combinatorica*, 14:71–109, 1994. ISSN 0209-9683. doi:10.1007/BF01305952.
- [115] P. Ventress. ESM identification- (a view from the user). Technical report, Thales Proprietary, 2002. Subject to NDA.
- [116] D. West. *Introduction to Graph Theory*, chapter 1, pages 1–66. Prentice-Hall, 2nd edition, 2001. ISBN 0-130-14400-2.
- [117] M. Woźniak. Packing of graphs and permutations—a survey. *Discrete mathematics*, 276(1):379–391, 2004.

Appendices

Ik

*Ik weet het: het mooiste zou zijn
als ik onvindbaar was
en altijd naar mijzelf bleef zoeken.
Hoe interessant zou dat niet zijn!
Maar ik ben zo vindbaar...
zo voor het oprapen...*

doe het licht uit, struikel over mij!

Toon Tellegen – Gedichten 1977-1999

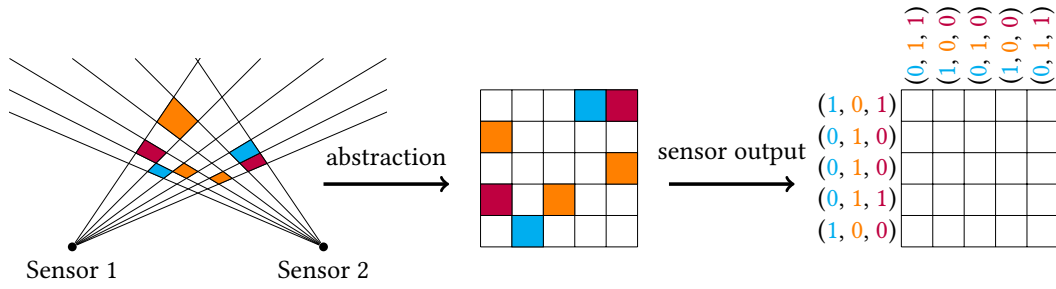


Nederlandse samenvatting*

Als ik mijn scriptie in één zin zou samenvatten, zou ik zeggen dat ik schepen zoek en identificeer. Ik doe dit voornamelijk in de context van het twee-sensoren probleem en het omgekeerde radar probleem. Daarover later meer, maar ik zal alvast verklappen dat ik probeer platformen te identificeren aan de hand van de zenders die ze bij zich hebben. Deze twee problemen zijn aangedragen door Thales UK, een – Frans van origine – multi-nationaal bedrijf, dat specialiseert in luchtvaart, defensie en informatietechnologie. De vraagstukken en algoritmes waar ik aan gewerkt heb, hebben toepassingen in onder andere surveillance, maar het doel van mijn project was niet zozeer om praktische oplossingen te vinden, maar voornamelijk om fundamentele eigenschappen van dit soort vraagstukken te onderzoeken.

Er zijn een groot aantal dingen, waarvan gedacht wordt dat ze niet uitgerekend kunnen worden, zelfs met bijna oneindig veel rekentijd. Hoewel dit (nog?) niet bewezen is, gelooft men dit zo sterk, dat er zelfs cryptografie protocollen, zoals bijvoorbeeld RSA, op gebaseerd zijn. Het blijkt dat de vraagstukken die ik heb onderzocht in mijn thesis grotendeels in deze categorie vallen. Mijn onderzoek focust op de eigenschappen die deze problemen zo moeilijk maken en ik toon aan in welke gevallen ze wel opgelost kunnen worden.

*There is no new information in this appendix.



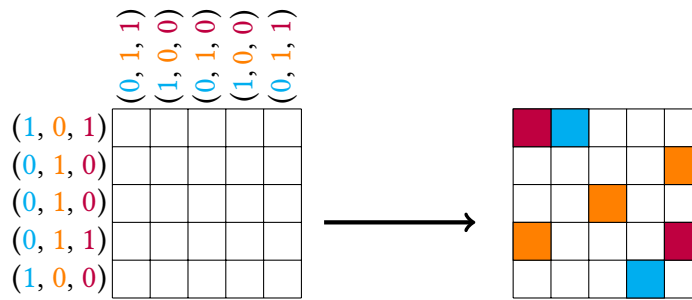
Figuur A.1: Links sensor 1 en 2 die een gebied observeren, in het midden het gekleurde raster in het gezichtsveld van sensor 1 en 2 en rechts wat sensor 1 en 2 detecteren.

A.1 Het twee-sensoren probleem

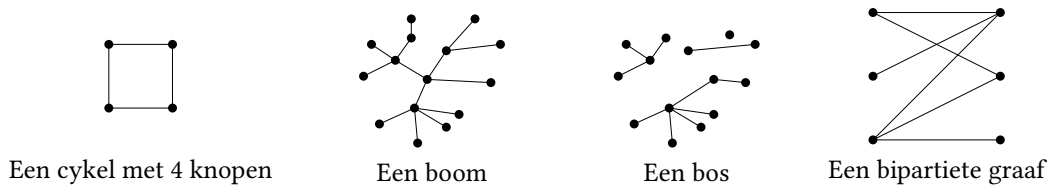
Het eerste deel van mijn scriptie gaat over het twee-sensoren probleem. Dit probleem is relevant na een succesvolle zender-naar-platform associatie voor beide sensoren (zie volgende sectie), of als elk platform een unieke zender heeft. In het twee-sensoren probleem zijn er twee sensoren die passief signalen van zenders in de buurt detecteren. Deze sensoren kunnen de richting van de signalen bepalen, maar niet van hoe ver de signalen komen. Om het probleem eenvoudiger te maken, zal ik ervan uit gaan dat alle platformen schepen zijn en dat er alleen radars gedetecteerd worden. Als je een lijn trekt vanaf sensor 1 voor elke richting die sensor 1 ‘ziet’ en hetzelfde doet voor sensor 2 zoals in Figuur A.1, ontstaat er een raster in het gebied dat allebei de sensoren zien. Elke richting die sensor 1 ‘ziet’ is een verticale rij hokjes in het raster is en elke richting die sensor 2 ‘ziet’ is een horizontale rij hokjes. Stel je voor dat er voor elk verschillend type signaal dat de sensoren detecteren een kleur is en dat er maximaal 1 signaal per hokje is. Dus de situatie die de sensoren ‘bekijken’, is een raster met gekleurde hokjes, maar de sensoren ‘zien’ alleen hoeveel hokjes van elke kleur er in elke rij zijn. Aan de hand van die getallen, is het doel van het twee-sensoren probleem om de hokjes in het raster zo in te kleuren, dat ze overeenkomen met de observaties. Dus het doel is om op basis van de observaties te bepalen, welk schip waar op het raster is. Dit is weergegeven in Figuur A.2 en A.4 links.

Dit lijkt erg op *Japanse puzzels* (ook wel *nonograms* genoemd), maar deze variant is beter bekend als *discrete tomografie*. Behalve het twee-sensoren probleem heeft discrete tomografie nog heel veel andere toepassingen, bijvoorbeeld in materiaalkunde en in de medische wetenschap [62]. Chrobak and Dürr [24], Dürr et al.

A.1. Het twee-sensoren probleem



Figuur A.2: Het twee-sensoren probleem.



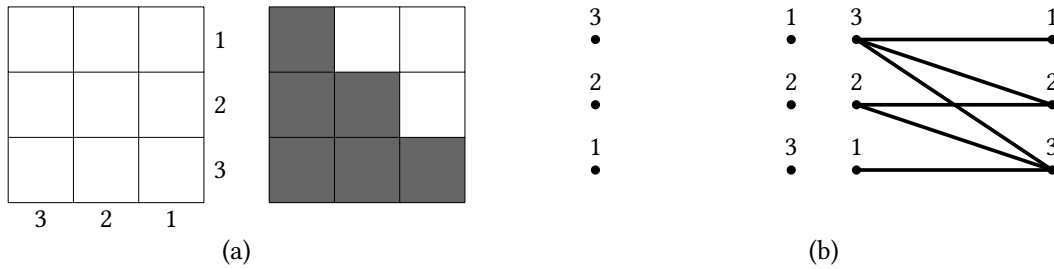
Figuur A.3: Verschillende grafen.

[35], Gardner et al. [47] bewezen dat dit probleem *NP-moeilijk* is zodra er twee of meer verschillende kleuren zijn. Dat wil zeggen dat dit probleem onderdeel is van een categorie problemen, waarvan niemand zeker weet of ze wel of niet efficiënt opgelost kunnen worden. Maar als het je lukt om één van deze problemen efficiënt op te lossen, dan kan dit voor elk probleem in deze categorie. Bovendien heb je dan een 100 jaar oud vraagstuk in theoretische informatica opgelost (namelijk $P=NP$) en \$1 miljoen verdiend [25]!

Nu maken we een kleine excursie naar grafen, het framework waarin ik werk. In Figuur A.3 zijn enkele voorbeelden van grafen die we vaak tegen komen. *Grafentheorie* is als het ware mijn gereedschapskist, waarmee ik de vraagstukken in mijn thesis te lijf ga. Een *graaf* is een netwerk met punten, ook wel *knopen* genoemd, en elk paar is al dan niet verbonden door een lijn, ook wel *zijde* genoemd. Ik heb het in mijn scriptie vooral over *enkelvoudige ongerichte grafen*. Dat is een mond vol, maar dat betekent alleen dat de zijdes geen richting hebben en dat er maar één zijde tussen twee knopen kan zijn. Als er een zijde tussen knoop v and w is, zeg ik ook wel dat v and w *incident* zijn met deze zijde. De *graad* van een knoop is het aantal zijdes incident met deze knoop. Bijvoorbeeld, de knoop links onder in de bipartiete graaf in Figuur A.3 heeft graad drie en elke knoop in een cykel heeft graad twee.

Een andere manier om discrete tomografie te beschrijven is als een *bipartiete graadrij*, zoals in Figuur A.4 is weergegeven. De reden waarom ik het probleem op deze manier formuleer, is omdat ik nu concepten en stellingen uit grafentheorie

Bijlage A. Nederlandse samenvatting

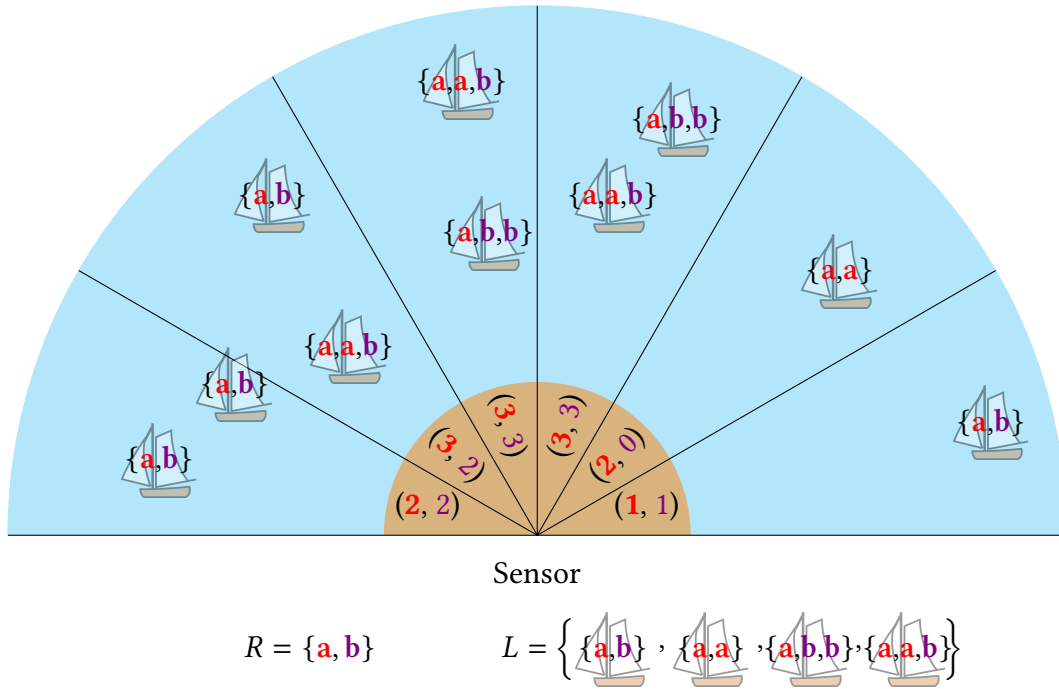


Figuur A.4: (a) Een discrete tomografie probleem met oplossing. (b) Hetzelfde probleem als een bipartiete graadrij met oplossing.

kan gebruiken om dit probleem te lijf te gaan. Het getal boven elke knoop is de graad van de knoop en dit correspondeert met de verticale projecties (links) en de horizontale projecties (rechts) van het discrete tomografie probleem, waar deze bipartiete graadrij van is afgeleid. We zijn op zoek naar een graaf, waarin elke knoop de juist graad heeft. Echter, in deze graaf mogen zijdes alleen tussen de knopen links en de knopen rechts gaan, want elke zijde is equivalent aan een gekleurd hokje in het raster. In Figuur A.4 (b) is een bipartiete graaf, waarin het nummer boven elke knoop ook de graad van de knoop is. Dit is dus een realisatie van de graadrijen $(3, 2, 1)$ en $(1, 2, 3)$.

In mijn thesis bespreek ik echter alleen algemene *graadrijen*. Dat wil zeggen: als ik je een rij positieve hele getallen geef, kan je dan een graaf vinden, zodat voor elk getal d , er precies één knoop is met graad d . Men zegt dat deze graaf de graadrij dan *realiseert*. Bijvoorbeeld $(2, 2, 2, 2)$ kan gerealiseerd worden door een cykel met vier knopen, zoals in Figuur A.3. Niet alle graadrijen kunnen gerealiseerd worden. Bijvoorbeeld $(2, 1)$ kan niet gerealiseerd worden. Dit is een heel oud probleem en het is precies bekend wanneer zo'n graaf bestaat en hoe die dan gevonden kan worden [38, 53, 58].

Daarom heb ik het probleem een beetje moeilijker gemaakt. Wat als ik je niet één graadrij geef, maar verschillende graadrijen. De eerste graadrij is voor alle blauwe zijdes, de tweede is voor alle rode zijdes, enzovoorts. Kan je nu een (enkelvoudige) graaf met gekleurde zijdes vinden, zodat je elke graadrij realiseert? Bijvoorbeeld $(1, 1, 0)$, $(0, 1, 1)$ en $(1, 0, 1)$ kan gerealiseerd worden door een driehoek met verschillende kleuren op elke zijde, maar $(1, 1)$, $(1, 0)$ kan niet gerealiseerd worden. Het graadrijen probleem is een moeilijkere variant van discrete tomografie, dus over het algemeen denkt men dat dit niet efficiënt opgelost worden. Voor mijn scriptie vroeg ik me daarom af, wat er gebeurt als ik niet zomaar alle grafen goedkeur als realisatie, maar

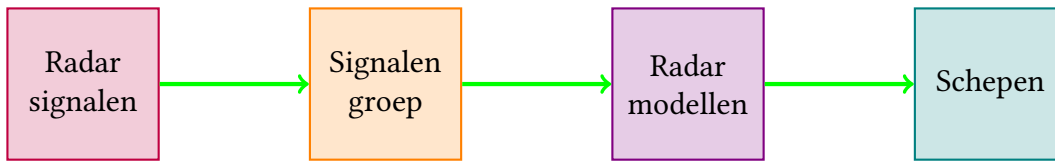


Figuur A.5: The sensor in het omgekeerde radar probleem ‘luistert’ naar radar modellen. Bibliotheek L geeft de verschillende schepen weer.

alleen maar grafen toelaat met een bepaalde structuur. Deze structuur heb ik zo uitgekozen, dat ik het alleen maar over grafen hoeft te hebben die niet te veel zijdes hebben. De eerste klasse grafen waar ik iets over kan zeggen bestaat uit grafen zonder *cykels*, ook wel een *boom* of een *bos* genoemd [9, 22] (zie Figuur A.3 voor voorbeelden). Dit resultaat heb ik kunnen uitbreiden naar grafen met hooguit één cykel; ik heb vier eenvoudige voorwaarden gevonden waaraan graadrijen moeten voldoen, als er een graaf met hooguit één cykel bestaat, die deze rijen realiseert. Dit is in [67] gepubliceerd. Bovendien heb ik een methode ontworpen om deze voorwaarden efficiënt te checken en een graaf te vinden als die bestaat [66].

A.2 Het omgekeerde radar probleem

Thales UK sponsorde mijn doctoraal studie en wilde dat ik *zender-naar-platform associatie* zou onderzoeken. Dit was dus het beginpunt van mijn thesis project. Het *omgekeerde radar probleem* is een specialisatie van *zender-naar-platform associatie*, waarin alle zenders radars en alle platformen schepen zijn. Er is een sensor, die ‘luistert’ naar de signalen die radars in de buurt uitzenden. Met die informatie probeer ik te bepalen welke schepen in de buurt zijn. Een soort zeeslagje dus. Net



Figuur A.6: De verschillende stadia van het omgekeerde radar probleem.

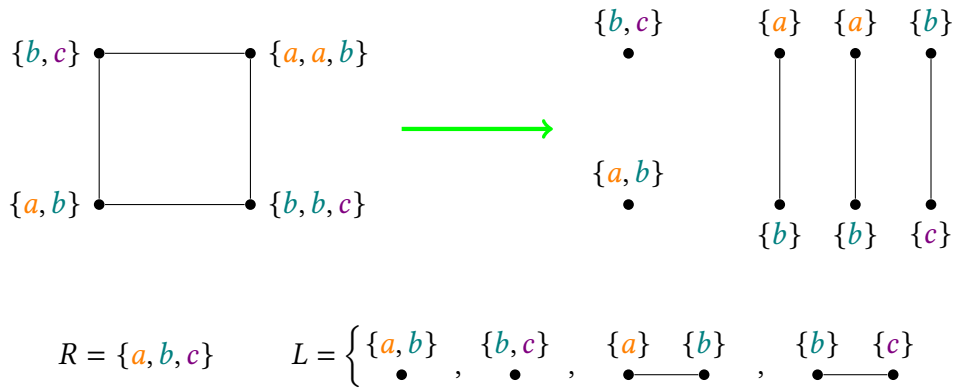
zoals in het twee-sensoren probleem kan de sensor wel de richting van het signaal bepalen, maar niet van hoever het komt. Figuur A.5 laat een sensor zien die ‘luistert’ naar alle radar signalen op de schepen in de verschillende segmenten. Op het eerste gezicht lijkt het misschien alsof dit probleem weinig met mijn ‘gereedschapskist’ te maken heeft. Dit komt doordat er verschillende stadia van het probleem zijn.

Eerst moeten de radar signalen gesorteerd worden, zodat alle signalen waarvan men denkt dat ze van dezelfde radar komen, bij elkaar in een groep zitten. Voor elke groep wordt bepaald wat voor radar model zulke signalen uitzendt. Aan de hand van een bibliotheek met schepen en de verschillende combinaties radar modellen die deze schepen over het algemeen bij zich hebben, probeert men dan te bepalen welke schepen waar zijn. Thales UK heeft al methodes om de eerste twee stadia van het probleem op te lossen, dus ik hoef me alleen maar bezig te houden met het laatste deel. Dit deel is het belangrijkste voor de veiligheidsdienst, want zo weten ze met wat voor schip ze te maken hebben, vriend of vijand.

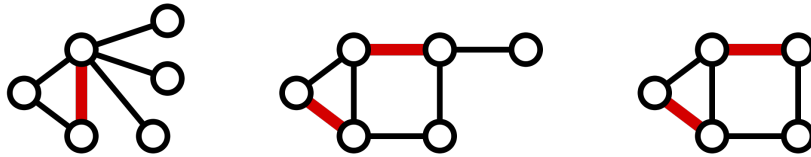
Net als het twee-sensoren probleem, is dit probleem NP-moeilijk, dus er is weinig hoop om het in het algemeen op te lossen. Eerst heb ik een model bedacht dat een geïdealiseerde versie van het probleem weergeeft, wat ik het gekleurde L -model heb genoemd; Neem een grote graaf G , waarin elke knoop een locatie voorstelt. Op elke knoop is een aantal radar modellen van een set R . Stel je elk radar model voor als zijnde een kleur. Daarnaast is er een bibliotheek L met kleine grafen, ook elk met een set kleuren op elke knoop. Elke graaf in L stelt een type schip voor. Het doel is om de kleine grafen uit L zo op de graaf G te ‘plakken’, dat alle kleuren in G precies een keer bedekt worden door kleuren in de ‘geplakte’ grafen. Grafen uit L kunnen meerdere keren gebruikt worden in dit proces, want het is goed mogelijk dat er meerdere schepen van een bepaald soort zijn. Ik noem dit proces een L -decompositie in G en er is een voorbeeld in Figuur A.7.

Ik heb eerst precies in kaart gebracht wanneer het probleem zeker NP-moeilijk is. Het blijkt dat de structuur van G niet uitmaakt, noch de hoeveelheid kleuren in R ; dit probleem wordt min of meer onmogelijk zodra alle schepen in L drie of meer kleuren

A.2. Het omgekeerde radar probleem



Figuur A.7: Een L -decompositie van een graaf.

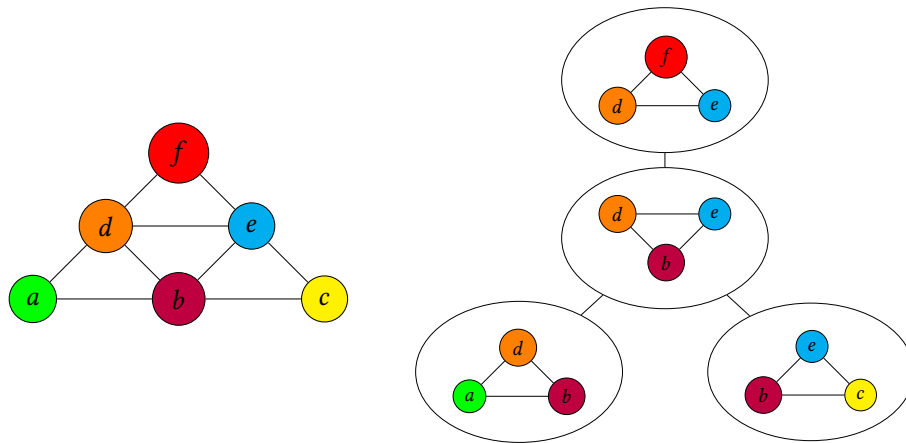


Figuur A.8: Koppelingen in verschillende grafen [80].

hebben. Daarom focus ik alleen op schepen in L met twee of minder kleuren. In dit geval is een L -decompositie hetzelfde als een *koppeling* in een gerelateerde graaf. Een koppeling is een set zijdes, waarin geen twee zijdes een gemeenschappelijke knoop hebben, zoals in Figuur A.8.

Hoewel er al snelle methodes bekend zijn om een koppeling in een willekeurige graaf te vinden [36, 79, 114], blijkt dat de grafen in het omgekeerde radarprobleem een bepaalde structuur hebben, waardoor er nog snellere methodes zijn. Deze grafen hebben namelijk *beperkte boombreedte*. Boombreedte is een getal wat weergeeft hoe erg de graaf op een boom lijkt. Bomen en bossen hebben boombreedte één. Een cykel heeft boombreedte twee. De boombreedte van G is berekend aan de hand van een *boomdecompositie* van G , zoals in Figuur A.9. Dit is een boom T , waarin elke knoop een verzameling knopen van G heeft. Ik zal deze verzameling een *zak* noemen. In Figuur A.9 is elke ovaal een zak met drie knopen uit G . De manier waarop de knopen van G verdeeld worden moet aan een aantal voorwaarden voldoen, zodat, wanneer er iets berekend wordt over de graaf G , dit gedeeltelijk berekend kan worden binnen één zo'n zak. De boombreedte is de grootte van de grootste zak min één.[†] De boomdecompositie in Figuur A.9 heeft boombreedte twee, omdat geen enkele zak meer dan drie knopen bevat. Het is mogelijk om op een systematische, snelle manier

[†]Dit is een beetje vreemd, maar dit is omdat men wilde dat bomen boombreedte 1 hebben.

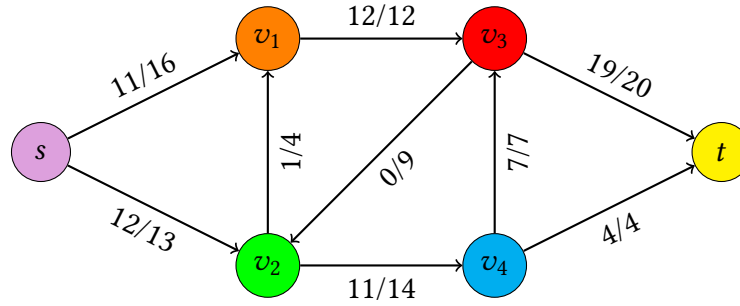


Figuur A.9: Een graaf links met boomdecompositie rechts.

door alle zakken in T te gaan en de nodige berekeningen te maken.

In mijn scriptie beschrijf ik hoe je een koppeling kan vinden in een *bipartiete graaf* met beperkte boombreedte. Een bipartiete graaf is een graaf die bestaat uit twee delen, zodat alle zijdes tussen deze twee delen zijn (en er dus geen enkele zijde binnen een deel is), zoals in Figuur A.3. In het omgekeerde radarprobleem betekent dit dat R slechts twee kleuren bevat. Ik beschrijf ook een langzamere methode voor algemene grafen met beperkte boombreedte, wat overeen komt met algemene R in het omgekeerde radarprobleem.

In het laatste hoofdstuk introduceer ik het *waarschijnlijkheidsmodel* en het *twee-tijden model*, beide uitbreidingen op het gekleurde L -model om het omgekeerde radar probleem te modelleren. Ook voor deze modellen hangt de complexiteit van het probleem af van het aantal kleuren per schip. Ik laat zien dat in het waarschijnlijkheidsmodel twee kleuren per schip equivalent is aan een koppeling in een gerelateerde graaf en dat het probleem NP-moeilijk is zodra schepen drie of meer kleuren hebben. Voor het twee-tijden model is het probleem al NP-moeilijk wanneer schepen twee of meer kleuren hebben, maar één kleur per schip kan opgelost worden aan de hand van *stromingen*. In een *gerichte graaf* (dus elke zijde heeft nu een richting) met een *bron* s en een *put* t , waarin elke zijde een *capaciteit* heeft, is een stroming f een functie op de zijdes, zodat er op elke zijde niet meer stroming dan de capaciteit van die zijde is, en er voor elke knoop (behalve s en t) evenveel stroming binnenkomt als eruit gaat. De stroming die uit de bron s komt is even groot als de stroming die in de put t gaat. In een gerichte graaf met capaciteiten op alle zijdes is het doel om de grootst mogelijke stroming van s naar t te vinden, zoals in Figuur A.10 is weergegeven. Gelukkig is het bekend hoe dat snel gedaan



Figuur A.10: Een maximale stroming in een gerichte graaf. Voor zijde e met a/b , is a de stroming en b de capaciteit [28].

kan worden en hoe groot zo'n stroming dan is [30, 31, 44].

A.3 Conclusie

In mijn scriptie heb ik aangetoond dat het twee-sensoren probleem en het omgekeerde radar probleem van nature combinatorieke, NP-moeilijke problemen zijn. Ik heb laten zien welke eigenschappen deze problemen zo moeilijk maken en onder welke voorwaarden deze problemen wel efficiënt opgelost kunnen worden. In sommige gevallen heb ik deze problemen gerelateerd aan constructies in grafentheorie, waarvan methodes bekend zijn om deze snel te vinden en in andere gevallen heb ik zelf methodes geconstrueerd om deze efficiënt op te lossen.

Hoewel er nu een stuk meer bekend is over deze problemen en sommige speciale gevallen zelfs snel opgelost kunnen worden, is er nog veel werk te doen. Ten eerste moet men proberen om deze speciale gevallen uit te breiden, zodat er meer gevallen opgelost kunnen worden. Een andere manier is om in plaats van naar een precieze oplossing, naar een benadering van de oplossing te zoeken. Dat wil zeggen, misschien hoeft je niet per se de beste oplossing te vinden, zolang het gegarandeerd is dat wat je vindt bijna net zo goed is. Er is al veel werk in dit veld verricht, en het werk in mijn thesis kan helpen om de beste benaderingsmethode te selecteren.