

2016

A theory of flow network typings and its optimization problems

<https://hdl.handle.net/2144/19750>

"Downloaded from OpenBU. Boston University's institutional repository."

BOSTON UNIVERSITY
GRADUATE SCHOOL OF ARTS AND SCIENCES

Dissertation

**A THEORY OF FLOW NETWORK TYPINGS AND ITS OPTIMIZATION
PROBLEMS**

by

MOHAMMAD SABER MIRZAEI

M.S., Iran University of Science and Technology, 2011
B.S., Sharif University of Technology, 2003

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

2016

© Copyright by
MOHAMMAD SABER MIRZAEI
2016

Approved by

First Reader

Assaf Kfoury, PhD
Professor of Computer Science

Second Reader

Azer Bestavros, PhD
Professor of Computer Science

Third Reader

Lorenzo Orecchia, PhD
Professor of Computer Science

Acknowledgments

The complexities of this project would not have been possible without the help and guidance of professors, friends, staff and many other members of the Boston University community. Special gratitude is directed towards my advisor, professor Assaf Kfoury for his support and encouragement throughout my PhD.

I dedicate this thesis to my loving parents, without their support and devotion I would not have accomplished this great feat. .

A THEORY OF FLOW NETWORK TYPINGS AND ITS OPTIMIZATION

PROBLEMS

MOHAMMAD SABER MIRZAEI

Boston University, Graduate School of Arts and Sciences, 2016

Major Professor: Assaf Kfoury, Professor of Computer Science

ABSTRACT

Many large-scale and safety critical systems can be modeled as flow networks. Traditional approaches of analysis of flow networks are *whole-system approaches* in that they require prior knowledge of the entire network before an analysis is undertaken, which can quickly become intractable as the size of the network increases.

In this thesis we study an alternative approach to the analysis of flow networks, which is *modular, incremental* and *order-oblivious*. The formal mechanism for realizing this compositional approach is an appropriately defined theory of *network typings*. Typings are formalized differently depending on how networks are specified and which of their properties is being verified. We illustrate this approach by considering a particular family of flow networks, called *additive flow networks*.

In additive flow networks, every edge is assigned a constant gain/loss factor which is activated provided a non-zero amount of flow enters that edge. We show that the analysis of additive flow networks, more specifically the max-flow problem, is NP-hard, even when the underlying graph is planar.

The theory of network typings gives rise to different forms of graph decomposition problems. We focus on one problem, which we call the *graph reassembling* problem. Given an abstraction of a flow network as a graph $G = (\mathbf{V}, \mathbf{E})$, one possible definition of this problem is specified in two steps: (1) We cut every edge of G into two halves to obtain a collection of $|\mathbf{V}|$ one-vertex components, and (2) we splice the two halves of all the edges, one edge at a time, in some order that minimizes the complexity of constructing a typing for G , starting from the typings of its one-vertex

components.

One optimization is minimizing “maximum” edge-boundary degree of components encountered during the reassembling of G (denoted as α measure). Another is to minimize the “sum” of all edge-boundary degrees encountered during this process (denoted by β measure). Finally, we study different variations of graph reassembling (with respect to minimizing α or β) and their relation with problems such as Linear Arrangement, Routing Tree Embedding, and Tree Layout.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Thesis Contributions and Organization	2
2	Related Work	7
2.1	A Theory of Typings for Flow Networks	7
2.2	Graph Reassembling and Graph Embedding	9
2.2.1	Minimum Congestion Communication Tree Embedding	10
2.2.2	Minimum Congestion Communication Tree Embedding	12
3	Compositional Design and Analysis of Flow Network	13
3.1	Background and Our Methodology	13
3.2	Formal Definitions and Preliminary Lemmas	18
3.3	Network Typings	23
3.4	Existence of Principal Network Typings	31
3.5	Complexity of Computing Principal Typings	41
3.6	Supplementary Proofs and Lemmas for Section 3.5	43
3.6.1	Proof of Lemma 13	43
3.6.2	Proof of Lemma 14	49
3.7	Conclusion and Future work	62
4	Max-Flow and Shortest Path Problems in Additive Flow Networks	64
4.1	Background and Motivation	64

4.2	Definitions and Preliminaries	66
4.3	Shortest Path Problem in Additive Flow Network	69
4.4	Maximum Flow Problem in Additive Flow Network	79
4.5	Conclusion and Future Work	83
5	Efficient Reassembling of Graphs	85
5.1	Background and Motivation	86
5.2	Problem Statement	87
5.3	Formal Definitions and Preliminary Lemmas	91
5.3.1	Binary Graph-Reassembling	93
5.3.2	Optimization Problems	96
5.3.3	Linear Graph-Reassembling	97
5.4	Examples	100
5.5	α -Optimization of Linear Reassembling Is NP-Hard	107
5.6	β -Optimization of Linear Reassembling Is NP-Hard	112
5.7	α -Optimization of Balanced Reassembling Is NP-Hard	122
5.8	β -Optimization of Balanced Reassembling Is NP-Hard	128
5.9	Supplementary Proofs for Sections 5.5 and 5.6	136
5.10	Conclusion and Future Work	144
6	Graph Reassembling as a Graph Embedding Problem	146
6.1	Background and Motivation	146
6.2	Definitions and Problem Statement	147
6.3	Minimum Length of Tree Layout	149
6.3.1	Min Tree Length of Complete Graphs	149
6.3.2	Min Tree Length of General Graphs	158
6.4	Layout Tree Problem in Relation with Graph Reassembling	160
6.5	Tree Length of Routing Trees	163
6.6	Conclusion and Future Work	165

7 Conclusion	167
Bibliography	169
Curriculum Vitae	176

List of Figures

3.1	Vertex v of degree 5 is transformed into a cycle with 5 vertices.	23
3.2	One-vertex network \mathcal{N} and its principal typings when $c = 10$, $c = 0$, and $c = -10$. . .	26
3.3	A 3-vertex additive network \mathcal{N} and its space of all feasible IO assignments when $c_1 = 4$ and $c_2 = 8$	28
3.4	For Example 9, a disconnected 2-vertex additive network \mathcal{N}	29
3.5	Example for the proof of Lemma 17.	47
3.6	Example of a planar network and its transformation into a 3-regular network	51
3.7	The planar network \mathcal{N}' is re-drawn on a rectangular grid	52
3.8	Macro expansion of $\text{Merge}(\mathcal{M}, \mathcal{M}', \Theta, \partial, \mathcal{C})$	54
3.9	Progress of Algorithm 1 on a good 3-outerplanar embedding	59
3.10	Main iteration of Algorithm 1 on a good 3-outerplanar embedding.	60
4.1	An example of an additive flow network and a feasible flow for it	68
4.2	The gadget that replaces a vertex of degree 4	74
4.3	The gadget replacing a vertex with some forbidden transitions.	75
4.4	The gadget replacing vertex v of degree 4 involve in forbidden transitions.	76
4.5	The additive flow network, constructed based on the graph G in 4.5a	77
4.6	Underlying graph of a 3CNF formula φ	80
4.7	The gadget that replaces every pair of literal-vertices $\{x, \bar{x}\}$	81
4.8	The flow network constructed from the graph of CNF formula φ	81
5.1	Two different drawings of the hypercube graph Q_3	101
5.2	Complete graph K_8	103
5.3	Star graph S_7	104

5.4	Comparison of arrangements in Example 66 and in Example 67.	106
5.5	For the statement of Lemma 94.	126
6.1	Planar embedding of T represented as the embedding of nodes on $k + 2$ lines L_i . .	151
6.2	Tree $\bar{T} \notin T(\mathcal{R}, \Delta)$ and the alternative tree \bar{T}' constructed from \bar{T}	152
6.3	Representation of initial tree T and its modified version \tilde{T}	156
6.4	Two planar embeddings of an optimal layout tree T of size $2n - 1$	158
6.5	The structure of rooted tree layout for augmented graph \bar{G}	162

List of Symbols and Abbreviations

PTAS	Polynomial-time Approximation Scheme
MinArr	minimum linear arrangement problem
f	flow function assigning flow values to edge of a network
f_{in}	amount of entering flow to a network
f_{out}	amount of departing a network
NNCC	not necessarily closed convex
g	gain function assigning gain values to edge of a network
U	capacity function assigning flow capacity values to edge of a network
$\mathbf{E}_{in,out}$	set of input/output edges
$\mathbf{E}_{\#}$	set of internal edges
\mathbb{R}	set of real values
\mathbb{R}_+	set of non-negative real values
PAFT	Path Avoiding Forbidden Transitions problem
\mathcal{T}	the set of all possible edge transitions in a graph
\mathcal{F}	the set of forbidden edge transitions in a graph
T	algebraic characterization of flow assignments to outer edges
Θ	reassembling sequence
\mathcal{B}	binary reassembling tree
$\partial(\mathcal{M})$	set of input/output edges of component \mathcal{M}
GRAPHS(2^*)	the class of all simple graphs with the size of power-of-2
α	max edge boundary degree encountered during the reassembling process
β	sum of edge boundary degrees encountered during the reassembling process
φ	bijective mapping from vertices of input graph to vertices of underlying graph
δ	congestion of an edge of the input graph after embedding
λ	dilation of an edge of the input graph after embedding
\mathbf{T}	layout tree
LA	tree length of a layout tree embedding
CutWidth	...	minimum cutwidth linear arrangement problem
MinArr	minimum linear arrangement problem

Chapter 1

Introduction

1.1 Motivation

Many large-scale and safety critical systems can be considered as interconnected sub-systems or modules communicating via *flows*. Each sub-system can be consumer and/or producer of flow of other(s). These flow connections are characterized by a set of variables and are regulated by a set of constraints thereof, reflecting inherent or assumed properties or rules governing how the modules operate and communicate as well as what constitutes safe operation. The general term flow abstractly represents streams of objects (such as commodities being shipped in a network, vehicles on a road, fluid in a pipe and *etc.*), data packets or expendable resources (such as electric energy, compute cycles) and *etc.*

Traditionally, the design and implementation of *flow networks*, follows a bottom-up approach providing analysis and certification of desirable invariants of the system as a whole network. Hence, most traditional flow network algorithms and frameworks can be considered as *whole network* approaches, in the sense that the view of the “complete” and “homogenous” system is required in order to be able to apply some property analysis on the system.

Analysis and verification of different properties across large-scale systems is a critical task which can quickly become costly or intractable as the size and complexity of a model increases. Putting the issue of algorithmic complexity aside, it may be the case that sub-systems must be modeled or combined without the full knowledge of their internal details. This limitation may arise for the sake of privacy, modular structure and other reasons such as having heterogeneous or unknown parts (to be developed in future). The need for an alternative view toward flow network

systems is the background to this work; a group effort to develop an integrated framework suitable for compositional analysis of flow networks that is simultaneously: *modular* (“distributed in space”), *incremental* (“distributed in time”), and *order-oblivious* (“components can be modeled and analyzed in any order”). These are the three defining properties of what we call a seamlessly compositional approach to system modeling and system analysis. Several previous works explain the methodology behind this environment, as well as its state of development and implementation [14, 15, 16, 59, 60, 62, 63, 88]. This incremental group effort provides an algebraic framework (called *typings theory*) providing an efficient and compositional analysis of flow networks in their standard definition. The standard flow networks are regulated by the *flow conservation* at every edge and every vertex. In other words, the amount of flow entering a flow network or a sub-network is equivalent to the amount of flow departing that.

In this thesis we cover an augmented variation of this framework which enables the compositional analysis of a more general definition of standard flow networks, called *additive flow networks*. Additive flow networks are a special case of so called *generalized flow networks* in the sense that to every edge is assigned a constant gain/loss factor which upon entering any amount of flow is activated and hence rules out the flow conservation constraint at every edge.

Due to compositional approach used in this framework, the efficiency of so called typings theory gives rise to some optimization problems with the most important one called *graph re-assembling* problem. Abstractly, graph reassembling problem targets the problem of finding the optimal possible order of combining different components of a flow network which maximizes the efficiency of the typing framework’s algorithms. Different possible constraints on the modular structure of the system enforce limitations or constraints on the order of combining sub-networks, which in turn bring about different variations of graph reassembling problem to be investigated.

1.2 Thesis Contributions and Organization

The inherent properties of many large-scale and safety critical flow systems calls for an integrated framework suitable for compositional analysis of flow networks that is simultaneously: modular

(distributed in space), incremental (distributed in time), and order-oblivious (components can be modeled and analyzed in any order). Hence, in this report we cover a framework which enables the compositional analysis of flow network systems as well as the algorithmic and optimization problems arising in the development of this framework. In this context, the contributions of this thesis span following aspects:

- **A compositional framework for flow networks (Chapter 3):** The background leading to this work, is a group effort to develop an integrated compositional framework for modeling and analysis of flow network systems. The central concept of this approach is an abstract representation of flow information of a network (or sub-network) \mathcal{N} using what we call a *network typing*. Given a network \mathcal{N} , with multiple input edges and multiple output edges, a typing for \mathcal{N} is a formal algebraic characterization of flow assignments to its set of input/output edges. A typing for \mathcal{N} is *principal* if it comprises exactly all the feasible flow information in \mathcal{N} , restricted to its input/output edges. Thus, a principal typing, in particular, captures the value of all maximal feasible flows (*i.e.*, the information on value of maximum in/out flow for \mathcal{N}). In this thesis we study a customized typings framework for the analysis of a more general definition of standard flow networks, called *additive flow networks*. Additive flow networks are a special case of so called *generalized flow networks* in the sense that to every edge is assigned a constant gain/loss factor which is activated provided a non-zero amount of flow enters that edge (and hence rules out the flow conservation constraint at every edge).

In our formulation, a typing T for network \mathcal{N} is in the form of a finite set of disjoint and not necessarily closed convex (NNCC) polyhedra $T = \{P_1, \dots, P_\ell\}$, in higher-dimensional Euclidean space \mathbb{R}^d , where d is the edge-boundary degree of \mathcal{N} (the number of \mathcal{N} 's input/output edges). Having so defined T , an input-output assignment $\mathbf{r} = \langle r_1, \dots, r_d \rangle$ to \mathcal{N} 's outer edges is said to satisfy T if \mathbf{r} is a point in P_i for some $i \in \{1, \dots, \ell\}$.

Let T_1 and T_2 be typings for networks \mathcal{N}_1 and \mathcal{N}_2 . One of our results shows that the typing of bigger component built by connecting some edges of \mathcal{N}_1 and \mathcal{N}_2 can be obtained by direct

algebraic operations on T_1 and T_2 , without any need to re-examine the internal details of the two components \mathcal{N}_1 and \mathcal{N}_2 . Also the compositionality of our framework guarantees that when analyzing \mathcal{N}_1 and \mathcal{N}_2 separately, neither T_1 nor T_2 depends on the other, even if they are communicating parts of a bigger network.

Finally, given a network \mathcal{N} partitioned into finitely many components $\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3, \dots$ (possibly as small as one-vertex components) with respective principal typings T_1, T_2, T_3, \dots , our framework propose an approach to assemble these typings in any order and compute a principal typing for the original network \mathcal{N} . The efficiency of computing the final typing T depends on a judicious partitioning of \mathcal{N} and the order of combining sub-network, which is to decrease as much as possible the number of edges running between separate components, and again recursively when assembling larger components from smaller components. Using this methodology, we present an efficient algorithm for finding the typing for a network \mathcal{N} when the underlying graph of \mathcal{N} is k -outerplanar and k is a small integer value relative to the size of \mathcal{N} .

- **Max-flow problem in additive flow networks (Chapter 4):** In standard definition of flow network problems, such as the max-flow problem, it is assumed that if f units of flow enter the edge $e = \langle v, w \rangle$ at its tail v , then exactly f units will reach its head u . In practice this assumption in many flow models does not hold. For instance, in the well-known *generalized flow networks*, if $f(e)$ units of flow enter v , and a gain factor $g(e)$ is assigned to e , then $g(e) \times f(e)$ units reach u .

In contrast to the well-studied generalized flow networks, are the flow networks where an additive fixed gain factor is assigned to every edge *if used*. Flow networks with additive gains and losses (*additive flow networks* for short), are different in many aspects from standard and generalized flow networks due to some properties such as *flow discontinuity*, lack of max-flow/min-cut *duality*, and unsuitability of *augmented path* methods. Similarly, in the context of shortest path problem which is tightly related to max-flow problem in generalized flow networks, some basic properties of the shortest path do not hold in the additive case. For

instance, it is not anymore the case that the sub-path, the prefix or the suffix of a shortest path must themselves be shortest. These properties make additive flow networks harder to analyze. We show that the shortest path problem and the maximum in/out-flow problems are NP-hard for the class of planar additive flow networks.

- **(Chapter 5) efficient reassembling of graphs:** Besides the questions of optimization and study of different reassembling variations which are naturally suggested, graph *reassembling* is a part of the execution by programs in flow network typings framework, which drastically effects their computational efficiency. Simply speaking, in network reassembling, the network is taken apart and reassembled in an order determined by the designer. For instance, a typing T for a network component \mathcal{N} formally expresses a constraining relation between the flow variables denoting the input/output edges of \mathcal{N} . The smaller the set of input/output edges of \mathcal{N} is, the easier and more efficient it is to formulate and analyze the typing T . Hence, in the process of finding principal typing T incrementally from the smaller components, it is very important to keep the set of input/output edges of the intermediate components as small as possible. This give rise to the *graph reassembling* problem. Given an abstraction of a flow network in the form of a connected graph $G = (\mathbf{V}, \mathbf{E})$, one possible definition of the reassembling of G is specified in two steps: (1) We cut every edge of G into two halves to obtain a collection of $n = |\mathbf{V}|$ one-vertex components, and (2) we splice the two halves of all the edges, one edge at a time, in some order that minimizes the "maximum" edge-boundary degree of any component encountered during the reassembling of G (called α measure) or "sum" of all edge-boundary degrees encountered during the reassembling of G (called β measure). We define and study different variations of graph reassembling problem, *i.e.*, *linear graph reassembling* and *balanced graph reassembling*. We show that the both problem variations are NP-hard problems.
- **Graph reassembling as an graph embedding problem (Chapter 6):** Every reassembling sequence for graph G corresponds to a unique rooted binary tree \mathcal{B} (called *binary reassembling tree*), where the set of its leaf nodes is bijectively related to the set of one-vertex com-

ponents of G and the root node correspond to the reassembled graph G . Every internal node $w \in V_I(\mathcal{B})$, as the root of a subtree, represented by \mathcal{B}_w , correspond to the vertex induced sub-graph of G comprising the set of vertices represented by leaf nodes of \mathcal{B}_w . Accordingly, the graph reassembling problem can be translated into a special case of graph embedding problem where, given an input graph G , the goal is to find a binary tree \mathcal{B} and a bijective mapping from the set of vertices of G onto the leaf nodes of \mathcal{B} . We study the graph reassembling problem as a variation of graph embedding problem, called *tree layout* problem (as a special case of *communication tree embedding* problems). Informally speaking, tree layout problem concerns the task of embedding the vertices of an input graph G into the leaf nodes of some binary tree T . Different optimization problems have been studied in the context of communication tree problems such as well-known minimum edge *dilation* and minimum edge *congestion* problems. We investigate a third and less investigate measure *i.e. tree length*, which is a representative for average edge dilation (communication delay) measure and also for average edge congestion measure. We show that finding a tree layout \mathbf{T} for an arbitrary graph G with minimum tree length is an NP-hard task.

Chapter 2

Related Work

2.1 A Theory of Typings for Flow Networks

Many large-scale and safety critical systems can be considered as collection of interconnected sub-systems or modules communicating via flows connections. These flow connections are characterized by a set of variables and are regulated by a set of constraints thereof, reflecting inherent or assumed properties or rules governing how the modules operate and communicate as well as what constitutes safe operation.

Traditionally, the design and implementation of such *constrained-flow networks*, follows a bottom-up approach providing analysis and certification of desirable invariants of the system as a whole network. For instance, the development of real-time applications necessitates the use of real-time kernels so that timing properties at the application layer (top) can be established through knowledge and/or tweaking of much lower-level system details (bottom), such as worst-case execution or context-switching times [28, 67, 82], specific scheduling and power parameters [9, 80, 85, 89], among many others.

The analysis and verification of different properties across large-scale systems is a critical task which can quickly become intractable as the size and complexity of a model increases. Hence, while justifiable in some instances, such vertical and whole-network approaches do not lend themselves well to emerging practices in the assembly of complex large-scale systems.

The need for an alternative and *horizontal* approach toward network systems is the background to this thesis, *i.e.* a group effort for development of an integrated framework suitable for compositional analysis of flow networks that is simultaneously: modular, incremental, and order-oblivious.

These are the three defining properties of what we call a seamlessly compositional approach to system modeling and system analysis.

This group work was initiated by introducing NetSketch, an integrated environment for the modeling, design and analysis of large-scale safety-critical systems with interchangeable parts [14, 15, 88]. NetSketch is conceived to assist system integrators in two types of activities: (1) as a modeling tool, it enables the abstraction of an existing system while retaining sufficient information about it to carry out future analysis of safety properties, and (2) as a design tool, NetSketch enables the exploration of alternative safe designs as well as the identification of minimal requirements for outsourced subsystems. NetSketch is an embodiment of a lightweight formal verification philosophy, whereby the power of a rigorous formalism is made accessible via a user friendly interface. NetSketch does so by exposing tradeoffs between exactness of analysis and scalability, and by combining traditional whole-system analysis with a more flexible compositional analysis. The compositional analysis is based on a strongly-typed Domain-Specific Language (DSL) for describing and reasoning about systems at various levels of sketchiness along with invariants that need to be enforced thereupon. Formal analysis is at the heart of modeling and design of systems using NetSketch. For example, regarding a modeling activity in which the user identifies the boundaries of interconnecting network components that need to be encapsulated into a single sub-network, NetSketch infers an appropriate set of constraints of that encapsulated subsystem. This set of constraints representing the flow properties of different components is called *typing*. Similarly, in conjunction with a design activity in which the user specifies a subsystem as a set of alternative designs (or else as a *hole* to be designed later), NetSketch must perform type checking to establish the safety of the design or the minimal requirements to be expected of any network component that can fill the hole in future.

A simplified version of a DSL for NetSketch along side with a formal semantics and type system for it was introduced in [16] and further studied in [59]. This DSL is further specialized for compositional analysis of flow networks. This DSL provides a formal approach for inductive assembling of flow networks from small networks or modules to produce arbitrarily large ones, with interchangeable functionally-equivalent parts. In these works, associated with this DSL is

presented a *type theory*, a system of formal annotations to express desirable properties of flow networks together with rules that enforce them as invariants across their interfaces, *i.e.*, the rules guarantee the properties are preserved as larger networks are built from smaller ones. A prerequisite for a type theory is a formal semantics, *i.e.*, a rigorous definition of the entities that qualify as feasible flows through the networks, possibly restricted to satisfying additional efficiency or safety requirements. This is carried out and presented as a denotational semantics in [16, 59].

Further more, the theory of typings for standard networks is studied in [61, 62, 63] where a representation of so called typing customized for standard networks is defined. In these reports the concept of *principal typing* for an arbitrary flow network \mathcal{N} is formulated as a typing which comprises exactly all the feasible flow information in \mathcal{N} , restricted to its input/output edges. Additionally, authors study the different elements of a typing theory for standard flow networks as well as presenting efficient and customized algorithms in this framework for standard flow networks (such as efficient Bind operator and an algorithm for finding a principal typing for standard flow networks). Particularly, in [63], authors adapt the concept of principal typing to the particular problem of computing a maximum-value feasible flow. Using the same methodology, in [62], author uses typings framework for finding the value of max flow in standard flow networks when the underlying graph is k -outerplanar.

2.2 Graph Reassembling and Graph Embedding

The compositional approach used in typings theory gives rise to some optimization problems with the most important one called *graph reassembling* problem. Given an abstraction of a flow network in the form of a connected graph $G = (\mathbf{V}, \mathbf{E})$, one possible definition of the reassembling of G is specified in two steps: (1) We cut every edge of G into two halves to obtain a collection of $n = |\mathbf{V}|$ one-vertex components, and (2) we splice the two halves of all the edges, one edge at a time, in some order that minimizes the complexity of constructing a typing for G , starting from the typings of its one-vertex components. One optimization is to minimize the “maximum” edge-boundary degree of any component encountered during the reassembling of G (denoted as α measure). An-

other optimization is to minimize the “sum” of all edge-boundary degrees encountered during the reassembling of G (denoted by β measure).

Every reassembling sequence for graph G correspond to a unique binary tree \mathcal{B} (called *binary reassembling tree*), where the set of leaf nodes is bijectively related to the set of one-vertex components and the root node correspond to the reassembled graph G . Every internal node $w \in \mathbf{V}_I(\mathcal{B})$, as the root of a subtree, represented by \mathcal{B}_w , correspond to the vertex induced sub-graph of G comprising the set of vertices represented by leaf nodes of \mathcal{B}_w . Accordingly, the graph reassembling problem can be viewed as a special case of graph embedding problems in the sense that, given an input graph G , the goal is to find a binary tree \mathcal{B} and a bijective mapping from the set of vertices of G to the leaf nodes of \mathcal{B} .

In the rest of this chapter we cover the relation between graph reassembling problem and some graph embedding problems. This set of graph embedding problems belong to the category of general *communication tree embedding problems*.

Consider a group of terminals communicating via a finite network $G = (\mathbf{V}, \mathbf{E})$, where the set of vertices (finite set \mathbf{V}) and edges (finite set \mathbf{E}), respectively represent the collection of terminals and their direct communication links. Communication tree embedding problem is the problem of mapping the set of terminals onto the set of vertices of some physical network represented by a tree \mathbf{T} . Accordingly, the two vertices $v, u \in \mathbf{V}(G)$ that are directly connected via $e \in \mathbf{E}(V)$, are connected indirectly via some path $P_{\mathbf{T}}(v, u)$ in \mathbf{T} .

2.2.1 Minimum Congestion Communication Tree Embedding

In the context of communication tree embedding, there are different measures to be minimized. Corresponding to our α measure in this report, is the *edge congestion* measure, which represents the maximum communication traffic on the edges of the host tree \mathbf{T} .

When the host tree \mathbf{T} is a simple path, this problem is a case of the well-studied *Cutwidth* problem (CutWidth) where first was used as a theoretical model for the number of channels in an optimal *linear layout* of a circuit [1, 68]. In this thesis we show that CutWidth and a variation of graph reassembling, which we call α -optimal *linear graph reassembling*, are polynomially reducible to

each other. This problem originally was shown to be NP-hard for general graphs in [38]. This NP-hardness result is later extended to the class of planar graphs with maximum degree 3 [75]. Among several classes of graphs where CutWidth is polynomially solvable are trees [24, 99], bipartite permutation graphs [49] and unit interval graphs [53]. In [7] it is shown that there is polynomial-time approximation scheme (PTAS) for CutWidth when the input graph is “dense”. In the general case, the best approximation result, to the best of our knowledge, has approximation ratio $\mathcal{O}(\log^2 n)$, where $n = |\mathbf{V}|$ is the size of the input graph [66].

In the case where the vertices of the input graph G are mapped onto the leaf nodes of some general host tree, the underlying tree is called a *routing tree* and the related problems are referred to as routing tree embedding problems. In a particular case that all internal nodes of a routing tree have degree 3, we speak of a *tree layout*. Seymour and Thomas in [86] show that minimum congestion tree layout problem, referred to as the *minimum carving-width problem* is NP-hard for general graphs. Later, the problem of finding tree layouts with minimum congestion was studied by Khuller [65], who presented a polynomial time algorithm for constructing a layout \mathbf{T} whose congestion is within a $\mathcal{O}(\log n)$ factor from the optimal. In [86] authors also study the relation between minimum carving-width (*i.e.* minimum congestion of tree layout) of planar graphs and their *branch decomposition*. Facilitating this relation, authors present a practical implementation of an algorithm for computing the optimal carving-width of planar graphs in $\mathcal{O}(n^4)$. The computation efficiency of the method of Seymour and Thomas for planar graphs, is initially improved in [50] and later in [48], where it is improved to $\mathcal{O}(n^3)$.

An undirected graph H is said to be immersed in G if a graph, isomorphic to H , can be obtained from a subgraph of G after lifting pairs of adjacent edges, *i.e.*, removing two adjacent edges $\{v, w\}$ and $\{w, u\}$ and adding the edge $\{v, u\}$. Robertson and Seymour in their Graph Minors series prove that any set of graphs contains a finite number of immersion minimal elements [83]. Simply speaking, for any class C of graphs, the set of graphs not in C contains a finite set (called *immersion obstruction set* of C) of immersion minimal elements. It is easy to check that the class of graphs with carving-width (*i.e.* minimum congestion tree layout) bounded by k is immersion-closed for any k . In other word, the carving-width of a graph is not smaller than the carving-width of any

of its immersions. While obstruction sets are well-studied for parameters that are closed under minors [6, 19, 30, 90, 92], the study of obstruction characterization for immersion closed graph classes is relatively new to be studied. Using this idea the characterizations of graphs that have carving-width at most k for $k \leq 3$ are presented in [13].

2.2.2 Minimum Congestion Communication Tree Embedding

Our β measure in graph reassembling problem is related to *tree length* measure, which is defined in the context of communication tree embedding problems. Tree length is equivalent to the summation of edge congestions (carving-width) in the underlying host tree \mathbf{T} . The tree length of a tree embedding represents the average distance (hence, average communication delay) between vertices of the source graph, and equivalently the average traffic on the edges of the host tree.

When the host tree \mathbf{T} is a simple path, the problem of finding an optimal tree layout with minimum tree length is equivalent to *Minimum Linear Arrangement* problem (MinArr). MinArr is shown to be NP-hard for general graphs [37], bipartite graphs [32], interval graphs and permutation graphs [25]. Among several classes of graphs where MinArr is solvable in polynomial time are trees [45, 87], certain Halin graphs [31, 72], outerplanar graphs [35], unit interval graphs [53] and Chord graphs [84]. In this Thesis we further study the relation between this problem and our problem of β -optimal linear graph reassembling.

While the problem of finding optimal tree layouts with respect to minimum congestion is relatively well-studied, to our best knowledge, this problem was not investigated before our work in [71]. Along with some other graph embedding problem, we will cover this problem in Chapter 6.

Chapter 3

Compositional Design and Analysis of Flow Network

Many large-scale and safety critical systems can be modeled as flow networks. Traditional approaches of analysis of flow networks are *whole-system approaches* in that they require prior knowledge of the entire network before an analysis is undertaken, which can quickly become intractable as the size of the network increases.

In this chapter we study an alternative approach to the analysis of flow networks, which is *modular, incremental* and *order-oblivious*. The formal mechanism for realizing this compositional approach is an appropriately defined theory of *network typings*. Typings are formalized differently depending on how networks are specified and which of their properties is being verified. We illustrate this approach by considering a particular family of flow networks, called *additive flow networks*. In additive flow networks, every edge is assigned a constant gain/loss factor which is activated provided a non-zero amount of flow enters that edge.

Using this methodology, we present an efficient algorithm for finding the typing for an additive network \mathcal{N} when the underlying graph of \mathcal{N} is k -outerplanar and k is a small integer value relative to the size of \mathcal{N} .

3.1 Background and Our Methodology

Background and motivation. The background to the compositional design and analysis of flow networks in this chapter is a group effort to develop an integrated environment for system modeling and system analysis that are simultaneously: *modular* (“distributed in space”), *incremental* (“distributed in time”), and *order-oblivious* (“components can be modeled and analyzed in *any* order”).

These are the three defining properties of what we call a seamlessly *compositional* approach to system modeling and system analysis. Several previous works explain the methodology behind this environment, as well as its state of development and implementation [14, 15, 16, 59, 60, 62, 63, 88].

Besides our main goal (*i.e.* providing a compositional framework for analysis of flow networks), to illustrate our methodology, we consider the classical *maximum in/out flow problem* (max-flow for short) as a direct application of this framework. A solution for this problem is an algorithm which, given an arbitrary network \mathcal{N} with one source vertex s and one sink vertex t , computes a maximal feasible flow f in \mathcal{N} departing s , or reaching t . That f is a *feasible flow*, means f is an assignment of non-negative values to the edges of \mathcal{N} satisfying *capacity constraints* at every edge and *flow conservation* at every vertex other than s and t . That f is *maximal* in-flow (similarly out-flow), means the net outflow departing vertex s (or, the net inflow arriving the vertex t) is maximized. A standard assessment of a max-flow algorithm measures its run-time complexity as a function of the size of \mathcal{N} . Our methodology is broader, in that it can be applied again to tackle other network-related problems with different measures of what qualifies as an acceptable solution.

In this chapter we consider flow networks in standard form as well as a special case called flow networks with additive gains and losses (additive flow networks for short). In contrary with standard flow networks, additive flow networks are networks where a fixed value, positive or negative, is added to the flow $f(e)$ on an edge e , *provided* $f(e) \neq 0$. The discontinuity in the flow in such networks, *i.e.*, a fixed non-zero value is added or subtracted along an edge only if that edge is used, makes the complexity of standard optimizations considerably harder.

In the context of standard networks, starting with the algorithm of Ford and Fulkerson in the 1950's [34], several different solutions have been found for the max-flow problem, based on the same fundamental concept of *augmenting path*. A refinement of the augmenting-path method is the *blocking flow* method [29], which several reseedgehers have used to devise better-performing algorithms. Another family of max-flow algorithms uses the so-called *preflow push* method (also called *push relabel* method), initiated by Goldberg and Tarjan in the 1980's [39, 44]. A survey of these families of max-flow algorithms to the end of the 1990's can be found in several reports [8, 40]. Further developments introduced variants of the augmenting-path algorithms and the closely re-

lated blocking-flow algorithms, variants of the preflow-push algorithms, and algorithms combining different parts of all of these methodologies [41, 43, 70]. More recently, an altogether different approach to the max-flow problem uses the notion of *pseudoflow* [22, 51].

On the other hand, no polynomial solution is known for max-flow problem in additive case. In fact this problem has been shown to be NP-hard in [20]. In this chapter we study the compositional analysis of additive flow networks, while in Chapter 4 we show that the max-flow problem stays NP-hard for the class of planar graphs.

The design and analysis of any of the aforementioned algorithms presumes that the given network \mathcal{N} is known in its entirety. No design of an algorithm and its analysis are undertaken until all the pieces (nodes, edges, and their capacities) are in place. We may therefore qualify such an approach to design and analysis as a *whole-network* approach.

Overview of our methodology. The central concept of our approach is what we call a *network typing*. To make this work, a network (or network component) \mathcal{N} is allowed to have “dangling edges”; in effect, \mathcal{N} is allowed to have multiple sources or *input edges* (*i.e.*, edges whose tails are not incident on any vertex) and multiple sinks or *output edges* (*i.e.*, edges whose heads are not incident on any vertex). Given a network \mathcal{N} , with multiple input edges and multiple output edges, a typing for \mathcal{N} is a formal algebraic characterization of all the feasible flows in \mathcal{N} – including, in particular, all maximal feasible flows.

More precisely, a *sound* typing T for network \mathcal{N} specifies conditions on input/output edges of \mathcal{N} such that every assignment f of values to the input/output edges satisfying these conditions can be extended to a feasible flow g in \mathcal{N} . Moreover, if the input/output conditions specified by T are satisfied by *every* input/output assignment f extendable to a feasible flow g , then we say that T is not only sound but also *principal* for \mathcal{N} .

In our formulation, a typing T for network \mathcal{N} is in the form of a finite set of disjoint *not necessarily closed convex* (NNCC) polyhedra,¹ $T = \{P_1, \dots, P_\ell\}$, in higher-dimensional Euclidean

¹For want of a better one, we use the acronym “NNCC” as a shorthand for *non-necessarily-closed convex*, which we adopted from earlier work by other researchers related to static analysis of programs [10, 11, 12]. A *closed convex polyhedron* in the space \mathbb{R}^d is the intersection of finitely many closed half-spaces, with each closed half-space being

space \mathbb{R}^d , where d is the *edge-boundary degree* of \mathcal{N} (the number of \mathcal{N} 's outer edges). Having so defined T , an input-output assignment $\mathbf{r} = \langle r_1, \dots, r_d \rangle$ to \mathcal{N} 's outer edges is said to satisfy T if \mathbf{r} is a point in P_i for some $i \in \{1, \dots, \ell\}$. Due to possible discontinuity in the flow (in additive flow networks) the state of all feasible flows and accordingly typing T is not always possible to be represented by one compact convex set (“polytope” for short). Therefore, the formulation of typings for standard network corresponds to one compact convex set.

Let T_1 and T_2 be principal typings for networks \mathcal{N}_1 and \mathcal{N}_2 . If we connect \mathcal{N}_1 and \mathcal{N}_2 by linking some of their output edges to some of their input edges, we obtain a new network which we denote (only in this introduction) $\mathcal{N}_1 \oplus \mathcal{N}_2$. One of our results shows that the principal typing of $\mathcal{N}_1 \oplus \mathcal{N}_2$ can be obtained by direct (and relatively easy) algebraic operations on T_1 and T_2 , without any need to re-examine the internal details of the two components \mathcal{N}_1 and \mathcal{N}_2 . Put differently, an analysis (to produce a principal typing) for the assembled network $\mathcal{N}_1 \oplus \mathcal{N}_2$ can be directly and easily obtained from the analysis of \mathcal{N}_1 and the analysis of \mathcal{N}_2 .

What we have just described is the counterpart of what type theorists of programming languages call a *modular* (or *syntax-directed*) *analysis* (or *type inference*) – which infers a type for the whole program from the types of its subprograms, and the latter from the types of their respective subprograms, and so on recursively, down to the types of the smallest program fragments.

Because our network typings are denoted by NNCCs (or polytopes for standard networks), we can in fact make our approach not only modular but also *compositional*, in the following sense. If T_1 and T_2 are principal typings for networks \mathcal{N}_1 and \mathcal{N}_2 , then neither T_1 nor T_2 depends on the other; that is, the analysis (to produce T_1) for \mathcal{N}_1 and the analysis (to produce T_2) for \mathcal{N}_2 can be carried out independently of each other without knowledge that the two will be subsequently assembled together.

Given a network \mathcal{N} partitioned into finitely many components $\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3, \dots$ with respective

defined by a linear inequality of the form $a_1x_1 + a_2x_2 + \dots + a_dx_d \leq b$ for some coefficients $a_1, a_2, \dots, a_d, b \in \mathbb{R}$. An *open convex polyhedron* is defined in the same way, with all the non-strict inequalities in the formulas replaced by strict ones; and a *not-necessarily-closed convex polyhedron* is again defined in the same way, with some (possibly none, possibly all) of the non-strict inequalities replaced by strict ones. All these polyhedra are also *bounded* in this report, but we omit this qualifier for succinctness. We reserve the name “polytope” for *compact* (i.e., *bounded and closed*) *convex polyhedron*.

principal typings T_1, T_2, T_3, \dots , we can then assemble these typings in *any* order – first in pairs, then in sets of four, then in sets of eight, etc. – to obtain a principal typing T for the whole of \mathcal{N} . Efficiency in computing the final principal typing T depends on a judicious partitioning of \mathcal{N} , which is to decrease as much as possible the number of edges running between separate components, and again recursively when assembling larger components from smaller components. At the end of this procedure, every input/output function f extendable to a maximal feasible flow g in \mathcal{N} can be directly read off the final typing T – but observe: not f' itself.

In contrast to the prevailing *whole-network* approaches, we call ours a *compositional* approach to the design and analysis of flow network algorithm.

Highlights. Our main contribution in this chapter is therefore a different framework for the design and analysis of additive network algorithms, which we here illustrate by presenting a new algorithm for the classical problem of computing a maximum flow. Our final algorithm combines several intermediate algorithms, each of independent interest for computing network typings. We mention some salient features distinguishing our approach from others:

1. As formulated in this chapter, using our framework one can calculate and return only the *value(s)* of flows on (subsets of) input/output edges, including the value of a maximum flow, without specifying a set of actual *paths* from source vertices to sink vertices that will carry such a flow. Other approaches handle the two problems simultaneously: Inherent in their operation is that, in order to compute a maximum-flow *value*, they need to determine a set of maximum-flow *paths*; ours does not need to.
2. We view the uncoupling of the two problems just described as an advantage. It underlies our need to be able to replace components – broken or defective – by other components as long as their principal typings are equal, without regard to how they may direct flow internally from input ports to output ports.
3. As far as run-time complexity is concerned, our final algorithm performs badly on some networks, *e.g.*, networks whose graphs are dense. However, on other special classes of net-

works, ours outperforms the best currently available algorithms (*e.g.*, on networks whose graphs are outerplanar or k -outerplanar for a small k in general).

4. In all cases, our algorithms do not impose any restrictions on flow capacities, in contrast to some of the best-performing algorithms of other approaches. In this report, flow capacities can be arbitrarily large or small, independent of each other, and not restricted to integral values.

3.2 Formal Definitions and Preliminary Lemmas

We give two equivalent definitions of flow networks with additive gains and losses. The first, perhaps more natural, was used by other researchers and ourselves in earlier work [20, 73]; the second is more convenient for our later analysis in this report. Note that our definitions for additive flow networks generalize and comprise the definition of the standard flow networks. Hence, our typing framework and positive results represented in this chapter are directly implied for standard flow networks. For a costumed representation of this framework as well as efficient algorithms for standard case, one can refer to our earlier report in [63]. Throughout, we write \mathbb{R} for the set of real numbers, and \mathbb{R}_+ for the set of non-negative real numbers.

Definition 1 (*Flow Networks with Additive Gains and Losses*). A flow network \mathcal{N} with additive gains and losses (*additive flow network* for short) is a quadruple $\mathcal{N} = (\mathbf{V}, \mathbf{E}, U, g)$ where \mathbf{V} is a finite set of vertices, \mathbf{E} is a finite set of edges, $U : \mathbf{E} \rightarrow \mathbb{R}_+$ is an (*upper bound*) *capacity function* on edges, and $g : \mathbf{E} \rightarrow \mathbb{R}$ is an *additive-gain function* on edges.

With every edge $e \in \mathbf{E}$ we associate two *endpoints*, denoted $tail(e)$ and $head(e)$, with the informal understanding that flow “moves” from $tail(e)$ to $head(e)$. The set \mathbf{E} is the disjoint union of three sets: the set $\mathbf{E}_\#$ of *internal* edges, the set \mathbf{E}_{in} of *input* edges, and the set \mathbf{E}_{out} of *output* edges:

$$\mathbf{E}_\# := \{ e \in \mathbf{E} \mid head(e) \in \mathbf{V} \text{ and } tail(e) \in \mathbf{V} \} \quad (\text{the internal edges of } \mathcal{N}),$$

$$\mathbf{E}_{in} := \{ e \in \mathbf{E} \mid head(e) \in \mathbf{V} \text{ and } tail(e) \notin \mathbf{V} \} \quad (\text{the input edges of } \mathcal{N}),$$

$$\mathbf{E}_{\text{out}} := \{ e \in \mathbf{E} \mid \text{head}(e) \notin \mathbf{V} \text{ and } \text{tail}(e) \in \mathbf{V} \} \quad (\text{the output edges of } \mathcal{N}).$$

The members of input/output edges $\mathbf{E}_{\text{in}} \cup \mathbf{E}_{\text{out}}$ for short are called the *outer* edges of \mathcal{N} . For brevity, we write $\mathbf{E}_{\text{in,out}}$ to denote the disjoint union $\mathbf{E}_{\text{in}} \cup \mathbf{E}_{\text{out}}$; the notation is ambiguous, but the context will make clear which member of $\mathbf{E}_{\text{in,out}}$ is an input edge and which is an output edge. Note that an outer edge has only one of its two endpoints in \mathbf{V} . We make two assumptions:

- for every internal edge $e \in \mathbf{E}_{\#}$, the two endpoints of e are distinct, $\text{head}(e) \neq \text{tail}(e)$,
i.e., there are no self-loops,
- for all internal edges $e, e' \in \mathbf{E}_{\#}$, if $e \neq e'$ then $\{\text{head}(e), \text{tail}(e)\} \neq \{\text{head}(e'), \text{tail}(e')\}$,
i.e., there are no multi-edges.

If $e \in \mathbf{E}_{\#}$, with $\text{head}(e) = v$ and $\text{tail}(e) = w$, we may write the vertex pair $\langle v, w \rangle$ to uniquely represent e .² □

Let $\mathcal{N} = (\mathbf{V}, \mathbf{E}, U, g)$ be defined as above. The *underlying graph* of \mathcal{N} is $G = (\mathbf{V}, \mathbf{E})$, which is obtained by omitting \mathcal{N} 's capacity function U and \mathcal{N} 's additive-gain function g . The underlying graph $G = (\mathbf{V}, \mathbf{E})$ is simple (no self-loops, no multi-edges), finite, and directed, with a few “dangling” edges (the outer edges).

We do not assume \mathcal{N} is connected as a directed graph – an assumption often made in studies of network flows, which is sensible when there is only one input edge (or “source node”) and only one output edge (or “sink node”).³

The power-set operator is denoted by $\mathcal{P}(\)$, so that $\mathcal{P}(\mathbf{E}) = \{A \mid A \subseteq \mathbf{E}\}$. Two functions $\text{in}(\cdot) : \mathbf{V} \rightarrow \mathcal{P}(\mathbf{E})$ and $\text{out}(\cdot) : \mathbf{V} \rightarrow \mathcal{P}(\mathbf{E})$ respectively represent the set of incoming edges and the set of outgoing edges for every vertex $v \in \mathbf{V}$:

$$\text{in}(v) := \{ e \in \mathbf{E} \mid \text{head}(e) = v \} \quad \text{and} \quad \text{out}(v) := \{ e \in \mathbf{E} \mid \text{tail}(e) = v \}.$$

²The notation of vertex pairs is not available to represent outer edges; that is, although we may take $\mathbf{E}_{\#} \subseteq \mathbf{V} \times \mathbf{V}$, it is not the case that $\mathbf{E}_{\text{in,out}} \subseteq \mathbf{V} \times \mathbf{V}$, if $\mathbf{E}_{\text{in,out}} \neq \emptyset$.

³Presence of multiple sources and multiple sinks is not incidental, but crucial to the way we develop and use our compositional analysis.

The *indegree* and *outdegree* functions are defined for every vertex $v \in \mathbf{V}$, respectively as $\text{deg}^+(v) := |\mathbf{in}(v)|$ and $\text{deg}^-(v) := |\mathbf{out}(v)|$. We also pose $\text{degree}(v) := \text{deg}^+(v) + \text{deg}^-(v)$.

A *flow* in \mathcal{N} is a function $f : \mathbf{E} \rightarrow \mathbb{R}_+$ that assigns a non-negative real number to every $e \in \mathbf{E}$. A flow f is *feasible* if it satisfies the *capacity constraint* on every edge and *flow conservation* at every vertex:

Capacity constraint. For every $e \in \mathbf{E}$, it holds that $0 \leq f(e) \leq U(e)$.

Flow conservation. For every $v \in \mathbf{V}$, it holds that

$$\sum_{e \in \mathbf{in}(v), f(e) > 0} \max(0, f(e) + g(e)) = \sum_{e \in \mathbf{out}(v)} f(e).$$

Observe carefully how the left-hand side of the flow-conservation equality is set up: The amount $g(e)$ is added to $f(e)$ provided $f(e) > 0$, *i.e.*, provided the link e carries some non-zero flow. However, if the link e is *lossy*, with $g(e)$ being a relatively large negative amount such that $f(e) + g(e) \leq 0$, then $\max(0, f(e) + g(e)) = 0$, *i.e.*, no flow carried by the link e reaches the vertex v .

Let $\mathcal{N} = (\mathbf{V}, \mathbf{E}, U, g)$ and $\mathcal{N}' = (\mathbf{V}', \mathbf{E}', U', g')$ be two additive flow networks. We say the outer edges of \mathcal{N} and \mathcal{N}' can be *identified* iff there is a one-one renaming of the outer edges of \mathcal{N} (or \mathcal{N}') such that $\mathbf{E}_{\text{in}} = \mathbf{E}'_{\text{in}}$ and $\mathbf{E}_{\text{out}} = \mathbf{E}'_{\text{out}}$. If the outer edges of \mathcal{N} and \mathcal{N}' can be identified, we will not need to explicitly mention the renaming function (which is not necessarily unique) that does the identification and it will suffice to know that it exists. We say that \mathcal{N} and \mathcal{N}' are *equivalent* iff:

- the outer edges of \mathcal{N} and \mathcal{N}' can be identified, and
- for every feasible flow $f : \mathbf{E} \rightarrow \mathbb{R}_+$ there is a feasible flow $f' : \mathbf{E}' \rightarrow \mathbb{R}_+$, and for every feasible flow $f' : \mathbf{E}' \rightarrow \mathbb{R}_+$ there is a feasible flow $f : \mathbf{E} \rightarrow \mathbb{R}_+$, such that $f(e) = f'(e)$ for every $e \in \mathbf{E}_{\text{in, out}} = \mathbf{E}'_{\text{in, out}}$.

We next give the definition of flow networks which we use in the rest of the report.

Definition 2 (*Modified Flow Networks with Additive Gains and Losses*). The definition of an addi-

tive flow network $\mathcal{N} = (\mathbf{V}, \mathbf{E}, U, g)$ in a modified form is identical to Definition 1, except for two parts:

1. The additive-gain function g is set up differently, by satisfying the following conditions:
 - $g : \mathbf{V} \rightarrow \mathbb{R}$ (instead of $g : \mathbf{E} \rightarrow \mathbb{R}$ in Definition 1),
 - for every $v \in \mathbf{V}$, if $g(v) \neq 0$, then $\deg^+(v) = \deg^-(v) = 1$.
2. For every $v \in \mathbf{V}$, it holds that $\text{degree}(v) \leq 3$.

The rest of Definition 1 is repeated here with no change. □

The definition of a *flow* $f : \mathbf{E} \rightarrow \mathbb{R}_+$ is the same for additive flow networks in both forms, the original (Definition 1) and the modified (Definition 2), and so are the **capacity-constraint** inequalities the same. However, the flow-conservation equalities have to be adjusted.

Flow conservation (modified form). For every $v \in \mathbf{V}$ such that $g(v) = 0$, it holds that

$$\sum_{e \in \mathbf{in}(v)} f(e) = \sum_{e \in \mathbf{out}(v)} f(e).$$

For every $v \in \mathbf{V}$ such that $g(v) \neq 0$, with $\mathbf{in}(v) = \{e_1\}$ and $\mathbf{out}(v) = \{e_2\}$, it holds that

$$\max(0, \text{if } f(e_1) \neq 0 \text{ then } f(e_1) + g(v) \text{ else } 0) = f(e_2).$$

We use the notion of equivalence to compare two additive networks, whether in the original form or in the modified form, possibly with each of the two in a different form.

Proposition 3. *For every additive network $\mathcal{N} = (\mathbf{V}, \mathbf{E}, U, g)$ in the original form (Definition 1), with $n := |\mathbf{V}|$ and $m := |\mathbf{E}|$, there is an equivalent additive network $\mathcal{N}' = (\mathbf{V}', \mathbf{E}', U', g')$ in the modified form (Definition 2) such that $|\mathbf{V}'| \leq n + 5m$ and $|\mathbf{E}'| \leq 6m$. Moreover, if the underlying graph of \mathcal{N} is planar, then so is the underlying graph of \mathcal{N}' .*

Proof. We first transform \mathcal{N} into an additive network $\mathcal{N}_1 = (\mathbf{V}_1, \mathbf{E}_1, U_1, g_1)$ satisfying part 1 in Definition 2. For this, we insert a fresh vertex in the middle of edge e if $g(e) \neq 0$, for every

$e \in \mathbf{E}$, and appropriately define $g_1 : \mathbf{V}_1 \rightarrow \mathbb{R}$ from $g : \mathbf{E} \rightarrow \mathbb{R}$ to make \mathcal{N} and \mathcal{N}_1 equivalent (all straightforward details omitted). For the resulting \mathcal{N}_1 , we have $n_1 := |\mathbf{V}_1| \leq n + m$ and $m_1 := |\mathbf{E}_1| \leq 2m$.

We next transform \mathcal{N}_1 into the desired \mathcal{N}' , satisfying both parts in Definition 2, by replacing every vertex $v \in \mathbf{V}_1$ such that $p = \text{degree}(v) \geq 3$ by an appropriate cycle with p vertices, say $\{v_1, \dots, v_p\}$. Note that $g(v) = 0$ in \mathcal{N}_1 , because $\text{degree}(v) \geq 3$. Let the edges incident to v be:

$$\{e_1, \dots, e_p\} = \{e \in \mathbf{E}_1 \mid \text{head}(e) = v \text{ or } \text{tail}(e) = v\}.$$

We introduce p new edges $\{e'_1, \dots, e'_p\}$ to form a directed cycle connecting the new vertices $\{v_1, \dots, v_p\}$ and we make each new vertex v_i the new endpoint of edge e_i instead of vertex v , for every $1 \leq i \leq p$. An example of the transformation from the vertex v to the directed cycle replacing it is shown in Figure 3.1. We complete the transformation by setting $g(v_i) = 0$ and $U(e'_i) = \text{“very large number”}$ for every $1 \leq i \leq p$.

This transformation is repeated for every vertex in \mathcal{N}_1 of degree ≥ 3 to produce the desired network \mathcal{N}' . It is straightforward to check that \mathcal{N}' is a network in the modified form of Definition 2 and \mathcal{N}' is equivalent to \mathcal{N} . Because the number of newly introduced vertices and the number of newly introduced edges are each at most $\sum_{v \in \mathbf{V}_1} \text{degree}(v)$, and since $\sum_{v \in \mathbf{V}_1} \text{degree}(v) = 2m_1$, we have:

$$|\mathbf{V}'| \leq n_1 + 2m_1 \leq n + m + 4m = n + 5m \quad \text{and} \quad |\mathbf{E}'| \leq m_1 + 2m_1 = 3m_1 \leq 6m.$$

It is readily seen that the transformation from \mathcal{N} to \mathcal{N}_1 , and then the transformation from \mathcal{N}_1 to \mathcal{N}' , both preserve planarity. \square

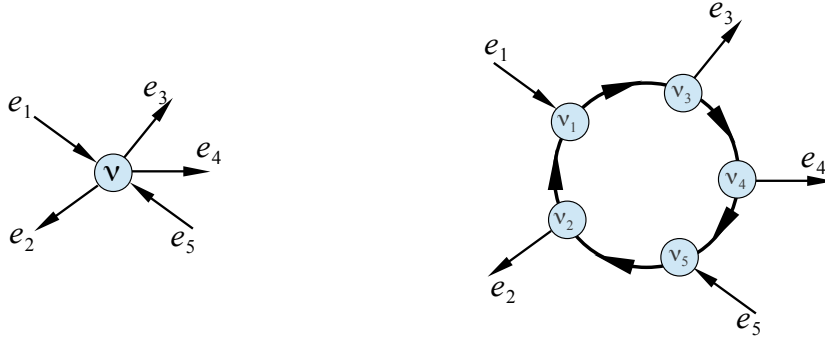


Figure 3.1: For the proof of Proposition 3. Vertex v of degree 5 (on the left) is transformed into a cycle with 5 vertices (on the right) each of degree 3. The capacity of every new edge in the cycle is a “very large number”.

3.3 Network Typings

For a precise definition of typings (Definition 4), we use standard notions of vector spaces and polyhedral analysis. A *closed half-space* H in \mathbb{R}^d is specified by a *non-strict* linear inequality: $H = \{ \mathbf{x} \in \mathbb{R}^d \mid \mathbf{a}^T \mathbf{x} \leq b \}$ for some coefficient vector $\mathbf{a} \in \mathbb{R}^d$ and scalar $b \in \mathbb{R}$. The boundary of H is the *hyperplane* specified by the corresponding linear equality $\{ \mathbf{x} \in \mathbb{R}^d \mid \mathbf{a}^T \mathbf{x} = b \}$. The complement of H is an *open half-space* H' specified by a *strict* linear inequality: $H' = \{ \mathbf{x} \in \mathbb{R}^d \mid \mathbf{a}^T \mathbf{x} > b \}$.⁴

A *not-necessarily-closed convex* (NNCC) *polyhedron* P in the Euclidean space \mathbb{R}^d is *either* the intersection of a finite number of not-necessarily-closed half-spaces, *or* $d = 0$ and $P = \emptyset$. More specifically, when $d \geq 1$, P is the set of simultaneous solutions of finitely many linear inequalities, strict and non-strict, each of the form:

$$a_1 x_1 + \dots + a_d x_d \bowtie b \tag{3.1}$$

where $a_1, \dots, a_d, b \in \mathbb{R}$ and $\bowtie \in \{\leq, <\}$. In the rest of this chapter, we only need to consider inequalities⁵ of form 3.1 where $a_1, \dots, a_d \in \{0, \pm 1\}$. As a result, every not-necessarily-closed

⁴We reserved boldface names, *e.g.*, \mathbf{a} and \mathbf{x} , to denote vectors. To explicitly mention the entries of \mathbf{a} , we write them between angle brackets as $\langle a_1, \dots, a_d \rangle$ and view them as a column vector. As usual, \mathbf{a}^T is the transpose of \mathbf{a} and $\mathbf{a}^T \mathbf{x}$ is the dot product $\mathbf{a} \cdot \mathbf{x}$.

⁵Note that every equality can be uniquely represented using two inequalities of the form 3.1 using symbol \leq . In the

half-space has a unique representation in the form of equation 3.1. Although our definition does not require it, all of our NNCC polyhedra will be *bounded*; we omit the qualifier for succinctness. A special case of NNCC polyhedra are the *compact convex polyhedra*, commonly called *polytopes*, when all defining linear inequalities are non-strict.

Definition 4 (*Network Typings*). Let $\mathcal{N} = (\mathbf{V}, \mathbf{E}, U, g)$ be an additive flow network, with $|\mathbf{E}_{\text{in,out}}| = d \geq 2$, accounting for at least one input edge and at least one output edge. A *typing* for \mathcal{N} is a finite set of disjoint NNCC polyhedra in the space \mathbb{R}^d , denoted by $T = \{P_1, \dots, P_\ell\}$, with $P_i \cap P_j = \emptyset$ for all $1 \leq i < j \leq \ell$. \square

An *input/output assignment* (or *IO assignment* for short) for the network $\mathcal{N} = (\mathbf{V}, \mathbf{E}, U, g)$ is a function $f : \mathbf{E}_{\text{in,out}} \rightarrow \mathbb{R}_+$ which assigns a non-negative value to every outer edge. Assuming a fixed ordering of the outer edges in $\mathbf{E}_{\text{in,out}}$, an IO assignment f specifies a point in the space \mathbb{R}^d , namely, the point $\langle f(e) \mid e \in \mathbf{E}_{\text{in,out}} \rangle$. For convenience, we use the names of the outer edges as the names of the d coordinates of the space \mathbb{R}^d relative to that fixed ordering of $\mathbf{E}_{\text{in,out}}$.

Definition 5 (*Typing Satisfaction*). Let \mathcal{N} be an additive flow network, and $T = \{P_1, \dots, P_\ell\}$ a typing for \mathcal{N} , as in Definition 4. We say an IO assignment $f : \mathbf{E}_{\text{in,out}} \rightarrow \mathbb{R}_+$ *satisfies* T iff $\langle f(e) \mid e \in \mathbf{E}_{\text{in,out}} \rangle$ is a point in one of the NNCC polyhedra of T , *i.e.*, there is $i \in \{1, \dots, \ell\}$ such that $\langle f(e) \mid e \in \mathbf{E}_{\text{in,out}} \rangle \in P_i$.

We say a flow $f : \mathbf{E} \rightarrow \mathbb{R}_+$ *satisfies* T iff the corresponding IO assignment $f' : \mathbf{E}_{\text{in,out}} \rightarrow \mathbb{R}_+$ obtained by restricting f to $\mathbf{E}_{\text{in,out}}$ satisfies T . \square

Let \mathcal{N} be an additive flow network and $T = \{P_1, \dots, P_\ell\}$ be a typing for \mathcal{N} , as in Definitions 4 and 5. We introduce two notions, *soundness* and *completeness*, with which we measure the extent to which the typing T is too restrictive or too liberal, as a constraint on IO assignments guaranteeing their safety, *i.e.*, guaranteeing that they can be directed through the network \mathcal{N} without violating any of its internal constraints.

Soundness. T is *sound* iff every IO assignment $f : \mathbf{E}_{\text{in,out}} \rightarrow \mathbb{R}_+$ satisfying T is extendable to a feasible flow.

rest of this chapter, for the sake of brevity, we avoid representing equality constraints using two inequalities.

Completeness. T is *complete* iff every feasible flow $f : \mathbf{E} \rightarrow \mathbb{R}_+$ satisfies T .

Soundness means T is restrictive enough to exclude all “unsafe” IO assignments. Completeness means T is liberal enough to include all “safe” IO assignments. Expressed differently:

$$\bigcup T \subseteq \{ f : \mathbf{E}_{\text{in,out}} \rightarrow \mathbb{R}_+ \mid f \text{ is extendable to a feasible flow} \} \quad (\text{soundness of } T)$$

$$\bigcup T \supseteq \{ f : \mathbf{E}_{\text{in,out}} \rightarrow \mathbb{R}_+ \mid f \text{ is extendable to a feasible flow} \} \quad (\text{completeness of } T)$$

Definition 6 (Principal Typings). Let \mathcal{N} be an additive flow network and $T = \{P_1, \dots, P_\ell\}$ be a typing for \mathcal{N} , as in Definitions 4 and 5. T is a *principal typing* for \mathcal{N} iff T is both sound and complete for \mathcal{N} . \square

Example 7. At the top of Figure 3.2 is a one-vertex additive network \mathcal{N} in modified form (Definition 2). The additive gain is c . When $c = +10$, its principal typing T is a set of two disjoint NNCC polyhedra $\{P_1, P_2\}$, with $P_1 = \{(0, 0)\}$ and

$$P_2 = \{ \langle r_1, r_2 \rangle \mid 0 < r_1 \leq 20, 10 < r_2 \leq 30, r_1 + 10 = r_2 \}.$$

When $c = 0$, \mathcal{N} is a standard flow network and a principal typing T' for \mathcal{N} consists of a single NNCC polyhedron P' , which is in fact a polytope. In earlier reports [62, 64] it is shown that for every standard network \mathcal{N} , there exists a unique principal typing consisting of exactly one polytope. P' can be decomposed into two disjoint NNCC polyhedra $\{P'_1, P'_2\}$, with $P'_1 = \{(0, 0)\}$ and

$$P'_2 = \{ \langle r_1, r_2 \rangle \mid 0 < r_1 \leq 20, 0 < r_2 \leq 20, r_1 = r_2 \}.$$

When $c = -10$, a principal typing T'' can be written as a set of three disjoint NNCC polyhedra $\{P''_1, P''_2, P''_3\}$, with $P''_1 = \{(0, 0)\}$ and

$$P''_2 = \{ \langle r_1, 0 \rangle \mid 0 < r_1 \leq 10 \} \quad \text{and} \quad P''_3 = \{ \langle r_1, r_2 \rangle \mid 10 < r_1 \leq 20, 0 < r_2 \leq 10, r_1 - 10 = r_2 \},$$

conforming to the requirement that the members of T'' must be disjoint (Definition 4). \square

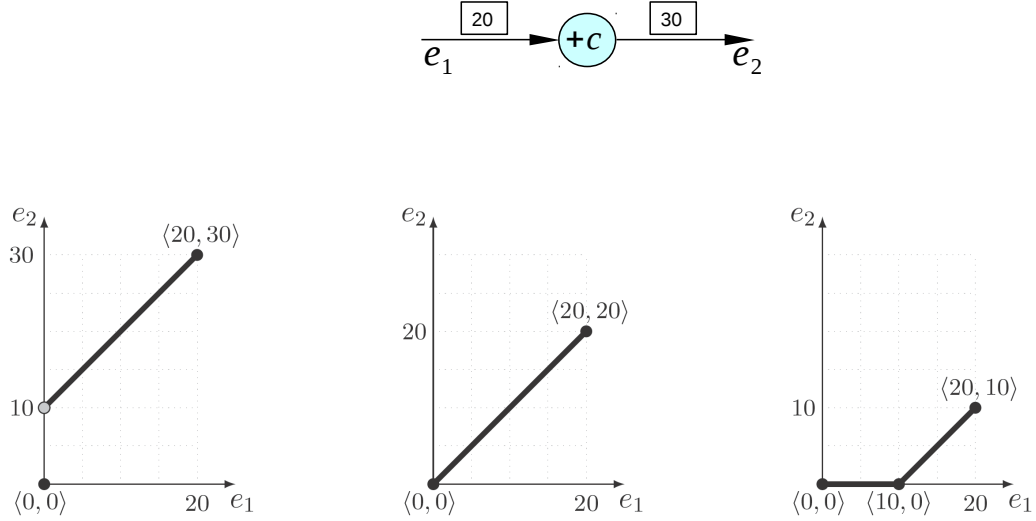


Figure 3.2: For Example 7, a one-vertex additive network \mathcal{N} (at the top) and its space of all feasible IO assignments when $c = 10$ (left), $c = 0$ (middle), and $c = -10$ (right), respectively. Additive gain c is inserted in the vertex, upper-bound capacities are boxed edge labels. The shaded point in \mathbb{R}^2 (on the left) is excluded from the feasible IO assignment space.

Example 8. On the left of Figure 3.3, there is a three-vertex additive network \mathcal{N} in modified form (Definition 2) where the additive gains c_1 and c_2 are left unspecified. When $c_1 \geq 0$ and $c_2 \geq 0$, a principal typing T for \mathcal{N} consists of four disjoint NNCC polyhedra $\{P_1, P_2, P_3, P_4\}$ in the Euclidean space \mathbb{R}^3 , with:

$$P_1 = \{ \langle 0, 0, 0 \rangle \},$$

$$P_2 = \{ \langle r_1, 0, r_3 \rangle \in \mathbb{R}^3 \mid 0 < r_1 \leq 20, c_1 < r_3 \leq \min\{20 + c_1, 42\}, r_1 + c_1 = r_3 \},$$

$$P_3 = \{ \langle 0, r_2, r_3 \rangle \in \mathbb{R}^3 \mid 0 < r_2 \leq 15, c_2 < r_3 \leq \min\{15 + c_2, 42\}, r_2 + c_2 = r_3 \},$$

$$P_4 = \{ \langle r_1, r_2, r_3 \rangle \in \mathbb{R}^3 \mid 0 < r_1 \leq 20, 0 < r_2 \leq 15, c_{1,2} < r_3 \leq \min\{35 + c_{1,2}, 42\}, r_1 + r_2 + c_{1,2} = r_3 \},$$

where we write $c_{1,2}$ for $c_1 + c_2$. When $c_1 = 4$ and $c_2 = 8$, a graphical representation of T is shown on the right of Figure 3.3. Observe that, when $c_1 = c_2 = 0$, the four NNCC polyhedra of T lie on the same hyperplane (namely, the hyperplane defined by the equation $e_1 + e_2 = e_3$) and together they collapse into a single polytope.

When one or both of $\{c_1, c_2\}$ is negative, a principal typing T' of \mathcal{N} consists of five or six NNCC polyhedra. For a specific case, consider $c_1 = -5$ and $c_2 = 8$, in which case a principal typing T' consists of five disjoint NNCC polyhedra $\{P'_1, P'_2, P'_3, P'_4, P'_5\}$ with:

$$P'_1 = \{ \langle 0, 0, 0 \rangle \},$$

$$P'_2 = \{ \langle r_1, 0, 0 \rangle \in \mathbb{R}^3 \mid 0 < r_1 \leq 5 \},$$

$$P'_3 = \{ \langle r_1, 0, r_3 \rangle \in \mathbb{R}^3 \mid 5 < r_1 \leq 20, 0 < r_3 \leq 15, r_1 - 5 = r_3 \},$$

$$P'_4 = \{ \langle 0, r_2, r_3 \rangle \in \mathbb{R}^3 \mid 0 < r_2 \leq 15, 8 < r_3 \leq 23, r_2 + 8 = r_3 \},$$

$$P'_5 = \{ \langle r_1, r_2, r_3 \rangle \in \mathbb{R}^3 \mid 5 < r_1 \leq 20, 0 < r_2 \leq 15, 8 < r_3 \leq 38, r_1 + r_2 + 3 = r_3 \}.$$

We omit the graphical representation of T' (a little more complicated to draw). □

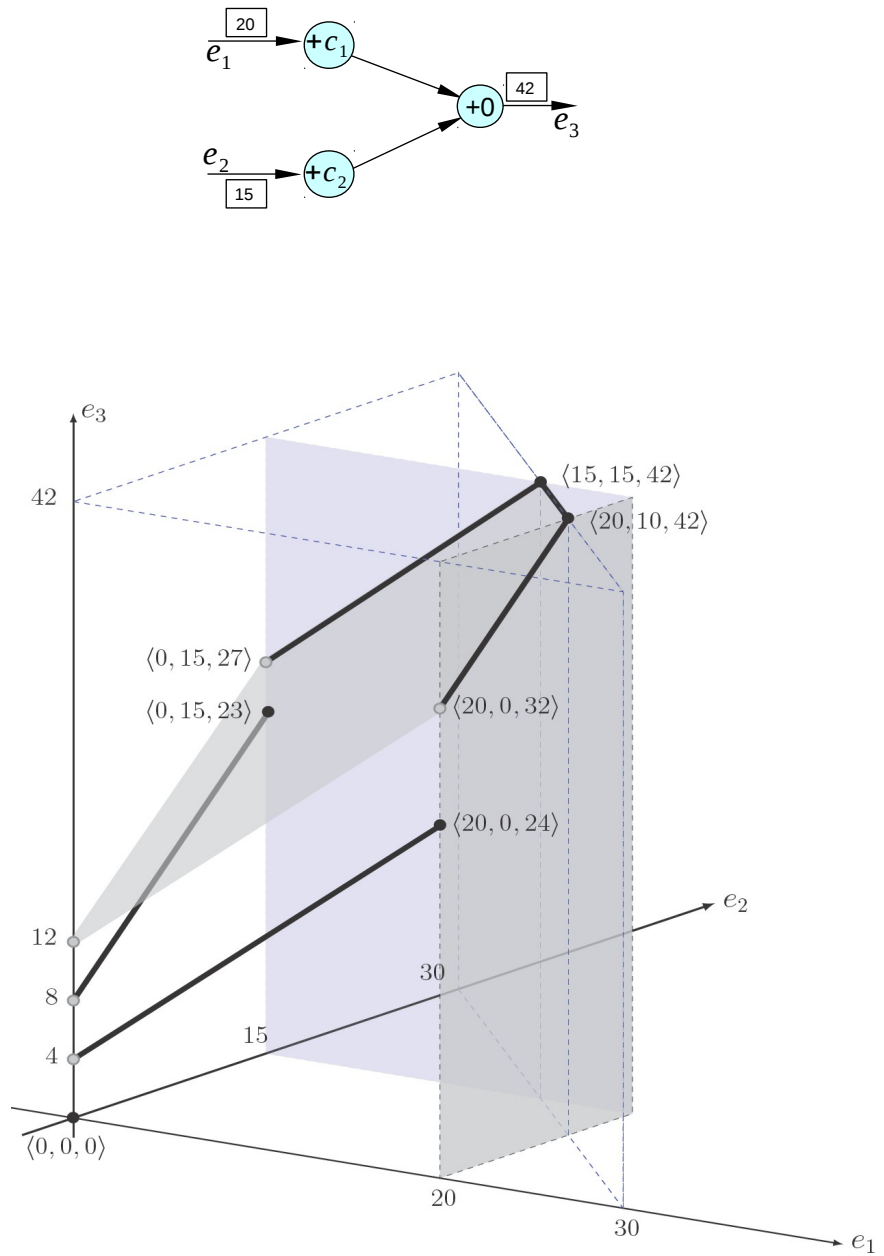


Figure 3.3: For Example 8, a 3-vertex additive network \mathcal{N} (at the top) and its principal typing $T \subseteq \mathbb{R}^3$ (at the bottom) when $c_1 = 4$ and $c_2 = 8$. Additive gains are inserted in vertices, upper-bound capacities are boxed numbers, missing upper-bound capacities are a “very large” number. Shaded points and edges in \mathbb{R}^3 are excluded from this networks feasible IO assignment space.

Example 9. Figure 3.4 depicts a disconnected network \mathcal{N} with two one-vertex components. The

additive gain factor of every vertex is inserted in that vertex. As mentioned before, in this framework we do not assume \mathcal{N} is connected as a directed graph. A principal typing T for \mathcal{N} consists of four disjoint NNCC polyhedra $\{P_1, P_2, P_3, P_4\}$ in the Euclidean space \mathbb{R}^4 , with:

$$P_1 = \{ \langle 0, 0, 0, 0 \rangle \},$$

$$P_2 = \{ \langle r_1, 0, r_3, 0 \rangle \in \mathbb{R}^4 \mid 0 < r_1 \leq 20, 5 < r_3 \leq 25, r_1 + 5 = r_3 \},$$

$$P_3 = \{ \langle 0, r_2, 0, r_4 \rangle \in \mathbb{R}^4 \mid 0 < r_2 \leq 20, 0 < r_4 \leq 20, r_2 = r_4 \},$$

$$P_4 = \{ \langle r_1, r_2, r_3, r_4 \rangle \in \mathbb{R}^4 \mid 0 < r_1 \leq 20, 0 < r_2 \leq 20, 5 < r_3 \leq 25, 0 < r_4 \leq 20, \\ r_1 + r_2 + 5 = r_3 + r_4, r_1 + 5 = r_3, r_2 = r_4 \},$$

□

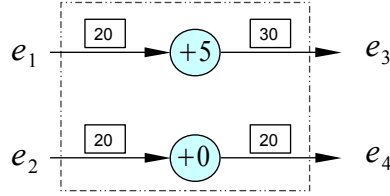


Figure 3.4: For Example 9, a disconnected 2-vertex additive network \mathcal{N} .

We use the names of the outer edges of network \mathcal{N} as the coordinates of the Euclidean space \mathbb{R}^d , with $d = |\mathbf{E}_{in,out}|$.

Let $\mathcal{C}(P)$ be the set of constraints representing a NNCC polyhedron $P \subseteq \mathbb{R}^d$, i.e. $P = \text{Poly}(\mathcal{C})$, where the operator $\text{Poly}(\dots)$ refers to the polyhedron in \mathbb{R}^d defined by the enclosed inequalities and equalities. We say constraint $C \in \mathcal{C}$ is redundant iff $\mathcal{C} - \{C\}$ defines the same NNCC as P , i.e. $P = \text{Poly}(\mathcal{C} - \{C\})$.

We say an NNCC polyhedron $P \subseteq \mathbb{R}^d$ is \emptyset -adequate iff P is a singleton set containing the point $\mathbf{0} = \langle 0, \dots, 0 \rangle$.

We define a bijection $idx : E_{in,out} \rightarrow \{1, \dots, d\}$ and take the usual ordering on the indices

$\{1, \dots, d\}$ to be the ordering on $E_{in,out}$.

Let $X \subseteq \mathbf{E}_{in,out}$ be a subset of the outer edges such that $X \cap \mathbf{E}_{in} \neq \emptyset$. We collect the indices induced by X by defining:

$$I_{in} = \{idx(e) \mid e \in X \cap \mathbf{E}_{in}\},$$

$$I_{out} = \{idx(e) \mid e \in X \cap \mathbf{E}_{out}\},$$

$$J = \{idx(e) \mid e \notin X\}.$$

We pose $I = I_{in} \uplus I_{out}$. We say an NNCC polyhedron $P \subseteq \mathbb{R}^d$ is X -adequate iff there exists a unique (possibly empty) set \mathcal{C}_0 of constraints defined on X (each of form 3.1) s.t. one of the following two cases holds:

(\dagger) $I_{in} \neq \emptyset \neq I_{out}$ and there are non-negative scalars $\{a_{m,i}, a_{M,i}\}_{i \in I} \cup \{b_j\}_{j \in J} \cup \{c_{in}\} \subseteq \mathbb{R}_+$ (subscripts “ m ” an “ M ” are for “min” and “max”, resp.) and a (possibly negative) scalar $c_{out} \in \mathbb{R}$ such that:

$$P = \text{Poly}(\{a_{m,i} < e_i \leq a_{M,i}\}_{i \in I} \cup \{e_j = b_j\}_{j \in J} \cup \{0 < \sum_{i \in I_{in}} e_i \leq c_{in}\} \cup \{\sum_{i \in I_{out}} e_i = c_{out} + \sum_{i \in I_{in}} e_i\} \cup \mathcal{C}_0).$$

(\ddagger) $I_{in} \neq \emptyset, I_{out} = \emptyset$, and there are non-negative scalars $\{a_{m,i}, a_{M,i}\}_{i \in I} \cup \{b_j\}_{j \in J} \cup \{c_{in}\} \subseteq \mathbb{R}_+$ such that:

$$P = \text{Poly}(\{a_{m,i} < e_i \leq a_{M,i}\}_{i \in I} \cup \{e_j = b_j\}_{j \in J} \cup \{0 < \sum_{i \in I_{in}} e_i \leq c_{in}\} \cup \mathcal{C}_0).$$

Every constraint in $\mathcal{C}(P)$ is linear, where the coefficients of the variables involved are in the set $\{0, -1, +1\}$. We also assume that in the adequate representation of an NNCC P , every constraint of $\mathcal{C}(P)$ is non-redundant. It is left to reader and not hard to check that every X -adequate NNCC P , representing a set of IO assignments of some network \mathcal{N} has a unique representation, where $X \subseteq \mathbf{E}_{in,out}$.

By this definition, an X -adequate NNCC polyhedron is entirely contained in the first orthant $(\mathbb{R}_+)^d$. Simply speaking, an NNCC polyhedron P is X -adequate only if for every point of P the

value of every dimension in X is positive and the value corresponding to every dimension not in X is fixed (*i.e.* for every two point of $p_1, p_2 \in P$ and every outer edge $e \notin X$, the values of the dimension corresponding to e in p_1 is the same as that of p_2).

Definition 10 (*Typings in Adequate Form*). Let \mathcal{N} be an additive flow network and $T = \{P_1, \dots, P_\ell\}$ be a typing for \mathcal{N} , as in Definitions 4 and 5, *i.e.*, T is a set of pairwise disjoint NNCC polyhedra in \mathbb{R}^d . We say T is in *adequate form* iff for every $P \in T$ there is a subset $X \subseteq \mathbf{E}_{\text{in, out}}$ such that P is X -adequate. \square

It is easy to extract an adequate representation of the typings shown in Examples 7, 8 and 9. Notice that in the X -adequate representation of NNCC P_4 in Example 9 we have $\mathcal{C}_0 = \{r_1 + 5 = r_3, r_2 = r_4\}$, where $X = \{e_1, e_2, e_3, e_4\}$. In adequate representation of other NNCCs the set \mathcal{C}_0 is empty (the same holds for all NNCCs in Examples 7 and 8).

3.4 Existence of Principal Network Typings

We prove the existence of a principal typing in adequate form for an arbitrarily given additive network \mathcal{N} (Theorem 11), from which we can extract the value of a maximum flow in \mathcal{N} (Corollary 15) efficiently. Theorem 11 does not estimate the cost of computing a principal typing in adequate form, which we study in Section 3.5. We first describe the process of “disassembling” and “reassembling” a network, which provides a compositional approach for calculating the principal typing of a flow network.

Let $\mathcal{N} = (\mathbf{V}, \mathbf{E}, U, g)$ be an arbitrary network. The process of disassembling \mathcal{N} involves “cutting in halves” its internal edges: If internal edge $e \in \mathbf{E}_\#$ is cut in two halves, then we remove e and introduce a new input edge e^+ with $\text{head}(e^+) = \text{head}(e)$ and a new output edge e^- with $\text{tail}(e^-) = \text{tail}(e)$. From the given \mathcal{N} , we define another network $\text{BreakUp}(\mathcal{N})$ where every internal edge $e \in \mathbf{E}_\#$ is cut into two halves. Let $\mathbf{E}_\#^{(+)}$ be the set of new input edges, and $\mathbf{E}_\#^{(-)}$ the set of new output edges. The input edges and output edges of $\text{BreakUp}(\mathcal{N})$ are therefore: $\mathbf{E}_{\text{in}} \cup \mathbf{E}_\#^{(+)}$ and $\mathbf{E}_{\text{out}} \cup \mathbf{E}_\#^{(-)}$, respectively.

If we call \mathcal{N}_0 the disassembled network resulting from $\text{BreakUp}(\mathcal{N})$, then \mathcal{N}_0 has $n = |\mathbf{V}| \geq 1$ one-vertex components and there are no internal edges in \mathcal{N}_0 :

$$\mathcal{N}_0 = \text{BreakUp}(\mathcal{N}) = \{\mathcal{M}_1, \mathcal{M}_2, \dots, \mathcal{M}_n\}.$$

We reassemble the original \mathcal{N} by defining the sequence of networks: $\mathcal{N}_0, \mathcal{N}_1, \dots, \mathcal{N}_m$ where $\mathcal{N}_m = \mathcal{N}$ and $\mathcal{N}_{k+1} = \text{Bind}(e, \mathcal{N}_k)$ for every $0 \leq i < m = |\mathbf{E}_\#|$, where Bind is the operation that splices e^+ and e^- . What we call the *reassembling sequence* Θ is the order in which the internal edges are spliced, *i.e.*, if:

$$\mathcal{N}_1 = \text{Bind}(e_1, \mathcal{N}_0), \quad \mathcal{N}_2 = \text{Bind}(e_2, \mathcal{N}_1), \quad \dots, \quad \mathcal{N}_m = \text{Bind}(e_m, \mathcal{N}_{m-1})$$

then $\Theta = e_1 e_2 \dots e_m$, where $\{e_1, \dots, e_m\} = \mathbf{E}_\#$. The *external dimension* of a network component \mathcal{M} , denoted by $\text{exDim}(\mathcal{M})$, is the number of its input/output edges (dangling edges) of \mathcal{M} . For each of the intermediate networks \mathcal{N}_k with $0 \leq i \leq m$, we define:

$$\text{index}(\mathcal{N}_k) := \max \{ \text{exDim}(\mathcal{M}) \mid \mathcal{M} \text{ is a component of } \mathcal{N}_k \}$$

and also $\text{index}(\Theta) := \max \{ \text{index}(\mathcal{N}_0), \text{index}(\mathcal{N}_1), \dots, \text{index}(\mathcal{N}_n) \}$.

Theorem 11 (Existence of Principal Typings). *For every additive flow network $\mathcal{N} = (\mathbf{V}, \mathbf{E}, U, g)$, there exists a principal typing $T = \{P_1, \dots, P_\ell\}$ in adequate form, as specified in Definitions 4, 5, and 10.*

Proof. We can assume \mathcal{N} is in modified form (Definition 2) by Proposition 3, and moreover that, for every $v \in \mathbf{V}$, if $g(v) \neq 0$ then $\text{degree}(v) = 2$ and if $\text{degree}(v) = 3$ then $g(v) = 0$.

We disassemble \mathcal{N} into $\mathcal{N}_0 = \text{BreakUp}(\mathcal{N})$, consisting of n separate one-vertex components, and choose a particular reassembling sequence $\Theta = e_1 e_2 \dots e_m$ so that if $\mathcal{N}_1 = \text{Bind}(e_1, \mathcal{N}_0), \dots, \mathcal{N}_m = \text{Bind}(e_m, \mathcal{N}_{m-1})$, then for every $k \geq 0$, there is a *main component* in \mathcal{N}_k , denoted $\widetilde{\mathcal{N}}_k$, with $n_k \geq 1$ vertices and $(n - n_k)$ one-vertex components. The latter one-vertex components are always a subset of $\text{BreakUp}(\mathcal{N}) = \{\mathcal{M}_1, \dots, \mathcal{M}_n\}$. At the end of this reassembling, the main component

$\tilde{\mathcal{N}}_m$ is equal to \mathcal{N}_m and there are no one-vertex components in \mathcal{N}_m . We arbitrarily choose the main component $\tilde{\mathcal{N}}_0$ of the initial \mathcal{N}_0 to be any member of $\text{BreakUp}(\mathcal{N})$. We call this method of reassembling, *linear reassembling*⁶.

Let κ_k be the edge-boundary degree of the main component $\tilde{\mathcal{N}}_k$ of \mathcal{N}_k . Initially $\kappa_0 \leq 3$, and the final value is $\kappa_m = |\mathbf{E}_{\text{in,out}}|$. For $0 < k < m$, the value of κ_k can be arbitrarily large and depends on the order of the reassembling sequence.

At every step $k \geq 0$ of the reassembling, we compute a principal typing for the main component $\tilde{\mathcal{N}}_k$ of \mathcal{N}_k . Let v be the single vertex of the initial main component $\tilde{\mathcal{N}}_0$. We determine a principal typing T_0 for $\tilde{\mathcal{N}}_0$ in an adequate form according to two cases:

Case 1: $g(v) \neq 0$ and $\text{degree}(v) = 2$. Example 7 shows how to handle this case. Formal details are omitted and left to the reader.

Case 2: $g(v) = 0$ and $\text{degree}(v) = 3$. Example 8 shows how to handle this case. In that example, there are three vertices, but taking the gains $c_1 = c_2 = 0$ simulates the situation of a one-vertex component with additive gain = 0. Formal details are omitted and left to the reader.

For the induction hypothesis (IH), we assume the existence of a principal typing T_k in adequate form for the main component $\tilde{\mathcal{N}}_k$ of \mathcal{N}_k . The induction step is to derive from T_k a principal typing T_{k+1} in adequate form for the main component $\tilde{\mathcal{N}}_{k+1}$ of \mathcal{N}_{k+1} . Let \mathcal{N}_{k+1} be obtained from \mathcal{N}_i by splicing e^+ and e^- , i.e., $\mathcal{N}_{k+1} = \text{Bind}(e, \mathcal{N}_k)$. There are two cases, depending on whether both of $\{e^+, e^-\}$ are outer edges of $\tilde{\mathcal{N}}_k$, or only one of $\{e^+, e^-\}$ is an outer edge of $\tilde{\mathcal{N}}_k$:

Case 3: Both of $\{e^+, e^-\}$ are outer edges of $\tilde{\mathcal{N}}_k$. After splicing e^+ and e^- , the edge-boundary degree of $\tilde{\mathcal{N}}_{k+1}$ is that of $\tilde{\mathcal{N}}_k$ minus 2. Assume $\tilde{T}_k = \{P_1, \dots, P_\ell\}$, the principal typing for the main component $\tilde{\mathcal{N}}_k$, is in adequate form. Also let $\tilde{\mathbf{E}}_{\text{in,out}}$ be the set of outer edge of $\tilde{\mathcal{N}}_k$. For every $1 \leq j \leq \ell$ there exists $X \subseteq \tilde{\mathbf{E}}_{\text{in,out}}$ such that P_j is X -adequate. We only consider the case where $I_{\text{in}} \neq \emptyset \neq I_{\text{out}}$, i.e., P_j is of the adequate form represented in (\dagger). The analysis of the other case is similar and left to the reader. Hence, for every X adequate NNCC $P_j \in \tilde{\mathcal{N}}_k$,

⁶Linear reassembling and other variations are further studied in Chapter 5)

there are non-negative scalars $\{a_{m,i}, a_{M,i}\}_{i \in I} \cup \{b_j\}_{j \in J} \cup \{c_{in}, c_{out}\} \subseteq \mathbb{R}_+$, such that:

$$P_j = \text{Poly}(\{a_{m,i} < e_i \leq a_{M,i}\}_{i \in I} \cup \{e_j = b_j\}_{j \in J} \cup \{0 < \sum_{i \in I_{in}} e_i \leq c_{in}\} \cup \{\sum_{i \in I_{out}} e_i = c_{out} + \sum_{i \in I_{in}} e_i\} \cup \mathcal{C}_0),$$

and the sets I and J are the indices induced by X as explained in Section 3.3.

Splicing e^+ and e^- is equivalent of enforcing that flow on e^+ is equal to flow on e^- . The principal typing \tilde{T}_{k+1} for $\tilde{\mathcal{N}}_{k+1}$ can be computed by intersecting P_j with hyperplane defined by the equality $e^+ = e^-$ for $1 \leq j \leq \ell$. For every X -adequate NNCC P_j if:

- $e^+, e^- \in X$, then intersecting P_j with hyperplane defined by the equality $e^+ = e^-$ results in:

$$\text{Poly}(\{e^+ = e^-\} \cup \mathcal{C}(P_j)) = \text{Poly}(\{e^+ = e^-\} \cup \{a_{m,i} < e_i \leq a_{M,i}\}_{i \in I} \cup \{e_j = b_j\}_{j \in J} \cup \{0 < \sum_{i \in I_{in}} e_i \leq c_{in}\} \cup \{\sum_{i \in I_{out}} e_i = c_{out} + \sum_{i \in I_{in}} e_i\} \cup \mathcal{C}_0).$$

Since the set of outer edges of $\tilde{\mathcal{N}}_{k+1}$ is $\tilde{\mathbf{E}}_{in,out} - \{e^+, e^-\}$, it is easy to check that the following X' -adequate form is the result of intersecting P_j with hyperplane defined by the equality $e^+ = e^-$, restricted to $\tilde{\mathbf{E}}_{in,out} - \{e^+, e^-\}$, where $X' = X - \{e^+, e^-\}$:

$$\begin{aligned} P'_j = & \text{Poly}(\{a_{m,i} < e_i \leq a_{M,i}\}_{i \in I - \{idx(e^+), idx(e^-)\}} \cup \{e_j = b_j\}_{j \in J} \cup \\ & \{0 < \sum_{i \in I_{in} - \{idx(e^+)\}} e_i \leq \min\{c_{in}, \sum_{i \in I_{in} - \{idx(e^+)\}} a_{M,i}\}\} \cup \\ & \{\sum_{i \in I_{out} - \{idx(e^-)\}} e_i = c_{out} + \sum_{i \in I_{in} - \{idx(e^+)\}} e_i\} \cup \mathcal{C}_0). \end{aligned}$$

- $e^+ \in X$ and $e^- \notin X$, then based on hypothesis assumption, $e^- = b$ for some constant non-negative value b . Using similar reasoning approach as in the previous case, one can check that $P'_j = \text{Poly}(\{e^+ = e^-\} \cup \mathcal{C}(P_j))$, if not empty, can be presented using the following adequate form:

$$P'_j = \text{Poly}(\{a_{m,i} < e_i \leq a_{M,i}\}_{i \in I - \{idx(e^+)\}} \cup \{e_j = b_j\}_{j \in J} \cup$$

$$\left\{ 0 < \sum_{i \in I_{\text{in}} - \{id_x(e^+)\}} e_i \leq \min\{c_{\text{in}}, \sum_{i \in I_{\text{in}} - \{id_x(e^+)\}} a_{M,i}\} \right\} \cup \left\{ \sum_{i \in I_{\text{out}}} e_i = c_{\text{out}} + \sum_{i \in I_{\text{in}} - \{id_x(e^+)\}} e_i \right\} \cup \mathcal{C}_0.$$

- $e^+ \notin X$ and $e^- \in X$, similar to the previous case and left to the reader.
- $e^+ \notin X$ and $e^- \notin X$, if $e^+ = e^- = b$ for some constant b , then P'_j has an exact adequate representation as P_j , otherwise it is empty.

Case 4: One of $\{e^+, e^-\}$ is an outer edge of $\tilde{\mathcal{N}}_k$ and the other is an outer edge of a one-vertex component \mathcal{M} .

Case 4 has two subcases, depending on whether the single vertex v of \mathcal{M} is such that $\text{degree}(v) = 2$ or $\text{degree}(v) = 3$. Similar to the approach used for Case 3, let $\tilde{T}_k = \{\tilde{P}_1, \dots, \tilde{P}_\ell\}$ be the principal typing for $\tilde{\mathcal{N}}_k$ in adequate form. We that all NNCCs members of \tilde{T}_k are of the adequate form represented in (\dagger) , the analysis of the other case is similar and left to the reader. Hence, for some set $X \subseteq \tilde{\mathbf{E}}_{\text{in,out}}$ the X -adequate NNCC $\tilde{P}_j \in \tilde{T}_k$ has the following form:

$$P_j = \text{Poly}\left(\left\{ a_{m,i} < e_i \leq a_{M,i} \right\}_{i \in I} \cup \left\{ e_j = b_j \right\}_{j \in J} \cup \left\{ 0 < \sum_{i \in I_{\text{in}}} e_i \leq c_{\text{in}} \right\} \cup \left\{ \sum_{i \in I_{\text{out}}} e_i = c_{\text{out}} + \sum_{i \in I_{\text{in}}} e_i \right\} \cup \mathcal{C}_0\right),$$

for some (possibly empty) set of constraints \mathcal{C}_0 , where I_{in} , I_{out} and J are the index sets as defined on X as explained in Section 3.3.

Case 4.1: Only one of $\{e^+, e^-\}$ is an outer edge of $\tilde{\mathcal{N}}_k$, and the single vertex v of \mathcal{M} is such that $g(v) \neq 0$ and $\text{degree}(v) = 2$. After splicing e^+ and e^- , the edge-boundary degree of $\tilde{\mathcal{N}}_{k+1}$ is equal to that of $\tilde{\mathcal{N}}_k$.

Case 4.2: Only one of $\{e^+, e^-\}$ is an outer edge of $\tilde{\mathcal{N}}_k$, and the single vertex v of \mathcal{M} is such that $g(v) = 0$ and $\text{degree}(v) = 3$. By splicing e^+ and e^- , the edge-boundary degree of $\tilde{\mathcal{N}}_{k+1}$ is that of $\tilde{\mathcal{N}}_k$ plus 1.

Case 4.1.1 has in turn two subcases, depending on whether e^+ or e^- is an outer edge of $\tilde{\mathcal{N}}_k$.

Case 4.1.1: e^+ is an input edge of $\tilde{\mathcal{N}}_k$ and e^- is an output edge of \mathcal{M} , with the outer edges of \mathcal{M} being $\{e^-, e_1\}$ where e_1 is necessarily an input edge.

We analyze Case 4.1.1 based on the sign of the gain value of vertex v , the only vertex of \mathcal{M} .

Case 4.1.1.1: $g(v) > 0$. As explained in Example 7, an adequate form of the principal typing for \mathcal{M} contains two NNCCs $P_1 = \{(0, 0)\}$ and

$$P_2 = \{ \langle r_1, r_2 \rangle \mid 0 < e_1 \leq a_{M,1}, g(v) < e^- \leq a_{M, \text{id}x(e^-)}, e^- = e_1 + g(v) \}.$$

Principal typing of $\tilde{\mathcal{N}}_{k+1}$ can be computed by intersecting the hyperplane defined by the constraint $e^+ = e^-$ with \tilde{P}_j and exactly one of NNCCs P_1 and P_2 (for every $1 \leq j \leq \ell$). For every X -adequate NNCC \tilde{P}_j if:

- $e^+ \in X$, in the adequate representation of \tilde{P}_j we have $a_{m, \text{id}x(e^+)} < e^+ \leq a_{M, \text{id}x(e^+)}$. Hence, \tilde{P}_j has no intersection with P_1 . If $g(v) \leq a_{M, \text{id}x(e^+)}$ it is not hard to check that the result of intersecting $e^+ = e^-$, \tilde{P}_j and P_2 , restricted to $\tilde{\mathbf{E}}_{\text{in}, \text{out}} - \{e^+\} + \{e_1\}$, can be represented in the X' -adequate form, where $X' = X - \{e^+\} + \{e_1\}$, as following:

$$\begin{aligned} \tilde{P}'_j = & \text{Poly}(\{ a_{m,i} < e_i \leq a_{M,i} \}_{i \in I - \{\text{id}x(e^+)\}} \cup \{ \{0, a_{m, \text{id}x(e^+)} - g(v)\} < e_1 \leq a_{M, \text{id}x(e^+)} - g(v) \}) \\ & \cup \{ e_j = b_j \}_{j \in J} \cup \{ 0 < \sum_{i \in I_{\text{in}} - \{\text{id}x(e^+), \text{id}x(e_1)\}} e_i \leq \min\{c_{\text{in}} - g(v), \sum_{i \in I_{\text{in}} - \{\text{id}x(e^+)\}} a_{M,i}\} \} \cup \\ & \{ \sum_{i \in I_{\text{out}}} e_i = c_{\text{out}} + g(v) + \sum_{i \in I_{\text{in}} - \{\text{id}x(e^+), \text{id}x(e_1)\}} e_i \} \cup \mathcal{C}_0). \end{aligned}$$

- $e^+ \notin X$, then in the X -adequate representation of \tilde{P}_j we have $e^+ = b$ for some constant b (*i.e.* $e^+ \in J$). If $b = 0$, then \tilde{P}_j may have intersecting points with P_1 , otherwise, if $v(g) < b$ it possibly has intersecting points with P_2 . In the former case the X -adequate representation of the result of intersecting the hyperplane corresponding to constraint $e^+ = e^-$ with \tilde{P}_j and P_2 , restricted to $\tilde{\mathbf{E}}_{\text{in}, \text{out}} - \{e^+, e^-\}$, is:

$$\begin{aligned} \tilde{P}'_j = & \text{Poly}(\{ a_{m,i} < e_i \leq a_{M,i} \}_{i \in I} \cup \{ e_j = b_j \}_{j \in J} \cup \{ e_1 = 0 \}_{j \in J} \\ & \cup \{ 0 < \sum_{i \in I_{\text{in}}} e_i \leq c_{\text{in}} \} \cup \{ \sum_{i \in I_{\text{out}}} e_i = c_{\text{out}} + \sum_{i \in I_{\text{in}}} e_i \} \cup \mathcal{C}_0), \end{aligned}$$

and in the later case if $0 < b - g(v) \leq a_{M, \text{id}x(e^-)}$ then:

$$\tilde{P}'_j = \text{Poly}(\{ a_{m,i} < e_i \leq a_{M,i} \}_{i \in I} \cup \{ e_j = b_j \}_{j \in J} \cup \{ e_1 = b - g(v) \}_{j \in J})$$

$$\cup \left\{ 0 < \sum_{i \in I_{\text{in}}} e_i \leq c_{\text{in}} \right\} \cup \left\{ \sum_{i \in I_{\text{out}}} e_i = c_{\text{out}} + \sum_{i \in I_{\text{in}}} e_i \right\} \cup \mathcal{C}_0 \quad .$$

Case 4.1.1.2: $g(v) < 0$. The adequate form of the principal typing T for \mathcal{M} can be written as a set of three disjoint NNCC polyhedra $\{P_1, P_2, P_3\}$, with $P_1 = \{(0, 0)\}$ and

$$P_2 = \left\{ \langle e_1, 0 \rangle \mid 0 < e_1 \leq -g(v) \right\} \text{ and}$$

$$P_3 = \left\{ \langle e_1, e^- \rangle \mid -g(v) < e_1 \leq a_{M, \text{id}x}(e^-), 0 < e^- \leq g(v) + a_{M, \text{id}x}(e^-), e_1 + g(v) = e^- \right\}.$$

For every X -adequate NNCC \tilde{P}_j if:

- $e^+ \in X$, then in the adequate representation of \tilde{P}_j we have $a_{m, \text{id}x}(e^+) < e^+ \leq a_{M, \text{id}x}(e^+)$. Hence \tilde{P}_j has no intersection with P_1 and P_2 . After splicing e^- and e^+ , and therefore intersecting hyperplane $e^- = e^+$ with \tilde{P}_j and P_3 results in the following X' -adequate form:

$$\begin{aligned} \tilde{P}'_j = & \text{Poly} \left(\left\{ \max\{-g(v), a_{m, \text{id}x}(e^+)\} < e_1 \leq \min\{a_{M, \text{id}x}(e^+), a_{M, \text{id}x}(e^-)\} \right\} \cup \right. \\ & \left. \left\{ a_{m, i} < e_i \leq a_{M, i} \right\}_{i \in I'} \cup \left\{ e_j = b_j \right\}_{j \in J} \cup \left\{ 0 < \sum_{i \in I'_{\text{in}}} e_i \leq c_{\text{in}} + g(v) \right\} \cup \right. \\ & \left. \left\{ \sum_{i \in I_{\text{out}}} e_i = c_{\text{out}} - g(v) + \sum_{i \in I'_{\text{in}}} e_i \right\} \cup \mathcal{C}_0 \right), \end{aligned}$$

where $X' = X - \{e^+\} + \{e_1\}$, $I' = I - \{\text{id}x(e^+)\}$ and $I'_{\text{in}} = I_{\text{in}} - \{\text{id}x(e^+), \text{id}x(e_1)\}$.

- $e^+ \notin X$, then in the adequate representation of \tilde{P}_j we have $e^+ = b_+$ for some constant and non-negative value b_+ . The NNCC \tilde{P}_j intersect with P_1 and P_2 only if $b_+ = 0$. After splicing e^- and e^+ and intersecting \tilde{P}_j with P_2 (and also with P_3) results in two NNCCs \tilde{P}'_j and \tilde{P}''_j (as the members of \tilde{T}_{k+1}), where:

$$\begin{aligned} \tilde{P}'_j = & \text{Poly} \left(\left\{ a_{m, i} < e_i \leq a_{M, i} \right\}_{i \in I} \cup \left\{ e_j = b_j \right\}_{j \in J - \text{id}x(e^+)} \cup \left\{ e_1 = 0 \right\} \cup \right. \\ & \left. \left\{ 0 < \sum_{i \in I_{\text{in}}} e_i \leq c_{\text{in}} \right\} \cup \left\{ \sum_{i \in I_{\text{out}}} e_i = c_{\text{out}} + \sum_{i \in I_{\text{in}}} e_i \right\} \cup \mathcal{C}_0 \right), \end{aligned}$$

and

$$\begin{aligned} \tilde{P}_j'' = \text{Poly}(\{ & a_{m,i} < e_i \leq a_{M,i} \}_{i \in I} \cup \{ e_j = b_j \}_{j \in J} \cup \{ 0 < e_1 < g(v) \} \cup \\ & \{ 0 < \sum_{i \in I_{\text{in}}} e_i \leq c_{\text{in}} \} \cup \{ \sum_{i \in I_{\text{out}}} e_i = c_{\text{out}} - g(v) + \sum_{i \in I_{\text{in}}} e_i \} \cup \mathcal{C}_0), \end{aligned}$$

and \tilde{P}_j' is X -adequate, while \tilde{P}_j'' is X' -adequate for $X' = X \cup \{e_1\}$.

Case 4.1.2: e^- is an output edge of $\tilde{\mathcal{N}}_k$ and e^+ is an input edge of \mathcal{M} , with the outer edges of \mathcal{M} being $\{e^+, e_1\}$ where e_1 is necessarily an output edge. The analysis of this case is similar to that of Case 4.1.1 and left to the reader.

Case 4.2 has in turn two subcases, depending on whether e^+ or e^- is an outer edge of $\tilde{\mathcal{N}}_k$.

Case 4.2.1: e^+ is an input edge of $\tilde{\mathcal{N}}_k$ and e^- is an output edge of \mathcal{M} , with the outer edges of \mathcal{M} being $\{e^-, e_1, e_2\}$.

Case 4.2.2: e^- is an output edge of $\tilde{\mathcal{N}}_k$ and e^+ is an input edge of \mathcal{M} , with the outer edges of \mathcal{M} being $\{e^+, e_1, e_2\}$.

Case 4.2.1 (resp., Case 4.2.2) has in turn two subcases, depending on whether only one of $\{e_1, e_2\}$ is an input edge (resp., output edge).

Case 4.2.1.1: e^+ is an input edge of $\tilde{\mathcal{N}}_k$, e^- is an output edge of \mathcal{M} , and the outer edges of \mathcal{M} are $\{e^-, e_1, e_2\}$, where e_1 is an input edge and e_2 is an output edge. The adequate form of the principal typing T for \mathcal{M} can be written as a set of four disjoint NNCCs $\{P_1, P_2, P_3, P_4\}$, with $P_1 = \{(0, 0)\}$ and

$$P_2 = \{ \langle e_1, e^-, 0 \rangle \in \mathbb{R}^3 \mid 0 < e^- \leq a_{M, \text{id}_X(e^-)}, e_1 = e^- \},$$

$$P_3 = \{ \langle e_1, 0, e_2 \rangle \in \mathbb{R}^3 \mid 0 < e_2 \leq a_{M, 2}, e_1 = e_2 \},$$

$$P_4 = \{ \langle e_1, e^-, e_2 \rangle \in \mathbb{R}^3 \mid 0 < e_1 \leq a_{M, 1}, 0 < e^- \leq a_{M, \text{id}_X(e^-)}, 0 < e_2 \leq a_{M, 2}, e_1 = e_2 + e^- \}.$$

For every X -adequate NNCC \tilde{P}_j of \tilde{T}_i if:

- $e^+ \in X$, then in the adequate representation of \tilde{P}_j it is the case that $a_{m,idx(e^+)} < e^+ \leq a_{M,idx(e^+)}$. Hence \tilde{P}_j has no intersection with P_1 and P_3 . After splicing e^- and e^+ and therefore, if not empty, intersecting hyperplane $e^- = e^+$ with \tilde{P}_j and P_2 results in the X' -adequate NNCC \tilde{P}'_j (member of \tilde{T}_{k+1}) such that:

$$\tilde{P}'_j = \text{Poly}\left(\{0 < e_i \leq a'_i\}_{i \in I} \cup \{a_{m,idx(e^+)} < e_1 \leq \min\{a_{M,idx(e^+)}, a_{M,idx(e^-)}\}\} \cup \{e_j = 0\}_{j \in J} \cup \{e_2 = 0\} \cup \left\{0 < \sum_{i \in I'_{\text{in}}} e_i \leq c'_{\text{in}}\right\} \cup \left\{\sum_{i \in I'_{\text{out}}} e_i = c'_{\text{out}} + \sum_{i \in I'_{\text{in}}} e_i\right\} \cup \mathcal{C}_0\right),$$

and $X' = X - \{e^+, e_1\}$ and $I'_{\text{in}} = I_{\text{in}} - \{idx(e^+), idx(e_1)\}$. Similarly, if not empty, intersecting hyperplane $e^- = e^+$ with \tilde{P}_j and P_4 results in the X' -adequate NNCC \tilde{P}''_j (member of \tilde{T}_{k+1}) where:

$$\tilde{P}''_j = \text{Poly}\left(\left\{0 < \sum_{i \in I'_{\text{in}}} e_i \leq c'_{\text{in}}\right\} \cup \{a_{m,idx(e^+)} < e_1 \leq \min\{a_{M,idx(e^+)}, a_{M,idx(e^-)}\}\} \cup \{0 < e_i \leq a'_i\}_{i \in I} \cup \{e_j = 0\}_{j \in J} \cup \{0 < e_2 \leq a_{M,2}\} \cup \left\{\sum_{i \in I'_{\text{out}}} e_i = c'_{\text{out}} + \sum_{i \in I'_{\text{in}}} e_i\right\} \cup \mathcal{C}_0\right),$$

and $X' = (X \cup \{e_1, e_2\}) - \{e^+\}$ and $I'_{\text{in}} = I_{\text{in}} - \{idx(e^+), idx(e_1)\}$.

- $e^+ \notin X$, then in the X -adequate representation of \tilde{P}_j we have $e^+ = b$ for some constant b . If $b = 0$, then \tilde{P}_j may have intersecting points with P_1 and P_3 , and possibly with P_2 and P_4 if $b > 0$. We only analyzed the former case. The analysis of the later case is similar and left to the reader. In the former case the intersecting $e^+ = e^-$, \tilde{P}_j and P_1 restricted to $(\tilde{\mathbf{E}}_{\text{in,out}} \cup \{e_1, e_2\}) - \{e^+, e^-\}$ results in X -adequate NNCC \tilde{P}'_j where:

$$\tilde{P}'_j = \text{Poly}\left(\{0 < e_i \leq a'_i\}_{i \in I} \cup \{e_1 = 0\} \cup \{e_2 = 0\} \cup \{e_j = 0\}_{j \in J} \cup \left\{0 < \sum_{i \in I_{\text{in}}} e_i \leq c'_{\text{in}}\right\} \cup \left\{\sum_{i \in I_{\text{out}}} e_i = c'_{\text{out}} + \sum_{i \in I_{\text{in}}} e_i\right\} \cup \mathcal{C}_0\right),$$

and the X -adequate representation of intersecting $e^+ = e^-$, \tilde{P}_j and P_3 restricted to

$(\tilde{\mathbf{E}}_{\text{in,out}} \cup \{e_1, e_2\}) - \{e^+, e^-\}$ is the NNCC \tilde{P}_j'' , such that:

$$\begin{aligned} \tilde{P}_j' = \text{Poly} & \left(\{0 < e_i \leq a_i'\}_{i \in I} \cup \{0 < e_1 \leq a_{M,2}\} \cup \{e_1 = e_2\} \cup \right. \\ & \left. \{e_j = 0\}_{j \in J} \cup \{0 < \sum_{i \in I_{\text{in}}} e_i \leq c_{\text{in}}'\} \cup \left\{ \sum_{i \in I_{\text{out}}} e_i = c_{\text{out}}' + \sum_{i \in I_{\text{in}}} e_i \right\} \cup \mathcal{C}_0 \right), \end{aligned}$$

and $I_{\text{in}}' = I_{\text{in}} \cup \{idx(e_1)\}$. Note that in the adequate representation of \tilde{P}_j' it is the case that $\mathcal{C}_0 := \mathcal{C}_0 \cup \{e_1 = e_2\}$. In other words, similar to adequate representation of NNCC P_4 in Example 9, this is a case where a non-empty set of constraints \mathcal{C}_0 appears in the adequate representation of an NNCC⁷ (see the adequate representation of an NNCC in Section 3.3).

The analysis of Case 4.2.1.2 to Case 4.2.2.2 are similar to our analysis in Case 4.2.1.1 and left to the reader. In every case, the principal typing T for \mathcal{M} contains up to four NNCCs and for every $1 \leq j \leq \ell$ NNCC \tilde{P}_j of \tilde{T}_i may have intersecting points with at most two of NNCCs of T .

Case 4.2.1.2: e^+ is an input edge of $\tilde{\mathcal{N}}_k$, e^- is an output edge of \mathcal{M} , and the outer edges of \mathcal{M} are $\{e^-, e_1, e_2\}$ where both e_1 and e_2 are input edges.

Case 4.2.2.1: e^- is an output edge of $\tilde{\mathcal{N}}_k$, e^+ is an input edge of \mathcal{M} , and the outer edges of \mathcal{M} are $\{e^+, e_1, e_2\}$ where e_1 is an input edge and e_2 is an output edge.

Case 4.2.2.2: e^- is an output edge of $\tilde{\mathcal{N}}_k$, e^+ is an input edge of \mathcal{M} , and the outer edges of \mathcal{M} are $\{e^+, e_1, e_2\}$ where both e_1 and e_2 are an output edges.

□

Suppose $\mathcal{N} = (\mathbf{V}, \mathbf{E}, U, g)$ is an additive flow network in modified form (Definition 2). We can measure the maximum flow in \mathcal{N} from the producers (source vertices') point of view or from the

⁷Since in this case the value of flow on e^+ and e^- is 0, the NNCC \tilde{P}_j' represent the space of all IO assignment of a flow network which in practice is disconnected (*i.e.*, there is no flow communication between the component containing \mathcal{M} and $\tilde{\mathcal{N}}_i$).

consumers (sink vertices’) point of view, which are generally different in an additive flow network. For the former, we want to maximize the *in-flow* $\sum_{e \in \mathbf{E}_{\text{in}}} f(e)$; for the latter, we want to maximize the *out-flow* $\sum_{e \in \mathbf{E}_{\text{out}}} f(e)$.

Corollary 12. *Let $\mathcal{N} = (\mathbf{V}, \mathbf{E}, U, g)$ be an additive flow network and $T = \{P_1, \dots, P_\ell\}$ a principal typing for \mathcal{N} in adequate form, which exists by Theorem 11. **Conclusion:** We can compute in time $\mathcal{O}(\ell)$ the values of a maximum in-flow and a maximum out-flow in \mathcal{N} .*

Note that the bound $\mathcal{O}(\ell)$ does not include the cost of computing the principal typing T . We take up an analysis of the cost of computing T in the next section.

Proof. Each X -adequate NNCC polyhedron $P \in T$ encodes, and returns in time $\mathcal{O}(1)$, the value of a maximum in-flow and the value of a maximum out-flow. We need to compare these values, 2ℓ of them, which are returned by the members of T , in order to select the largest among ℓ maximum in-flows and the largest among ℓ maximum out-flows. All formal details omitted. \square

3.5 Complexity of Computing Principal Typings

The embedding of a planar graph $G = (\mathbf{V}, \mathbf{E})$ in the plane is *k-outerplanar*, with $k \geq 1$, iff G has k layers of vertices, *i.e.*, after iteratively removing the vertices (and incident edges) on the outer face at most k times, we obtain the empty graph. The *outerplanarity index* of G is the smallest k such that G has a k -outerplanar embedding.⁸

An additive flow network $\mathcal{N} = (\mathbf{V}, \mathbf{E}, U, g)$ has a k -outerplanar embedding (resp., an outerplanarity index k) iff its underlying graph $G = (\mathbf{V}, \mathbf{E})$ has a k -outerplanar embedding (resp., an outerplanarity index k).

Lemma 13. *Let $\mathcal{N} = (\mathbf{V}, \mathbf{E}, U, g)$ be an arbitrary additive network in the original form (Definition 1), and let $\mathcal{N}' = (\mathbf{V}', \mathbf{E}', U', g')$ be the equivalent additive network in modified form (Defini-*

⁸ To compute the outerplanarity index of an arbitrary G , and produce a planar embedding of G of outerplanarity = its outerplanarity index, is not a trivial problem, for which the best known algorithm requires quadratic time $\mathcal{O}(n^2)$ in general [55]. It is worth noting that for a tri-connected planar graph, “outerplanarity” and “outerplanarity index” are the same measure, because the planar embedding of a tri-connected graph is unique ([26] or Section 4.3 in [27]). However, there are very simple examples of bi-connected, but not tri-connected, planar graphs with planar embeddings of arbitrarily large outerplanarity but whose outerplanarity index is as small as 1.

tion 2) constructed from \mathcal{N} according to the proof of Proposition 3. **Conclusion:** If \mathcal{N} is given in a k -outerplanar embedding, then its transformation into \mathcal{N}' produces a k' -outerplanar embedding where $k' \leq 2k$.

Proof. In Appendix 3.6.1. □

The proof of Theorem 11 is an induction on the number of internal edges $m = |\mathbf{E}_\#|$ in an additive network \mathcal{N} . This induction produces a reassembling of \mathcal{N} in no particular order, other than being linear, and does not give a way of estimating the cost of computing a principal typing T for \mathcal{N} .

Lemma 14. Let $\mathcal{N} = (\mathbf{V}, \mathbf{E}, U, g)$ be a network in modified form (Definition 2), given in a k -outerplanar embedding. Let $m = |\mathbf{E}_\#| \geq 1$ and $d = |\mathbf{E}_{in,out}| \geq 2$. **Conclusion:** A principal typing T of \mathcal{N} in adequate form can be computed in time $\mathcal{O}(m)$, where the hidden factors only depend on d and k .

Proof. In Appendix 3.6.2. □

Theorem 15 (Maximum Flow in Fixed-Parameter Linear Time). Let $\mathcal{N} = (\mathbf{V}, \mathbf{E}, U, g)$ be an additive flow network, given in a k -outerplanar embedding. **Conclusion:** We can compute in time $\mathcal{O}(n)$ the values of a maximum in-flow and a maximum out-flow in \mathcal{N} , where $n = |\mathbf{V}|$ and the hidden factors of $\mathcal{O}(n)$ only depend on d and k .

Proof. Based on Lemma 14, a principal typing $T = \{P_1, \dots, P_\ell\}$ for \mathcal{N} can be computed in $\mathcal{O}(m)$, where $m = |\mathbf{E}|$. It is a well-known fact that planar graphs fall into the category of sparse graphs, i.e. $m \in \mathcal{O}(n)$. Precisely, based on Eulers Polyhedral Formula, in the case of planar graphs, it is the case that $m < 3n - 6$ (for more details authors can refer to [3, 27]).

On the other hand, as explained in the proof of Lemma 14 and based on the inductive approach used for finding the principal typing of flow networks, the number of disjoint polyhedron comprising the principal typing T , namely, ℓ is bounded by 2^κ , where $\kappa = \max\{k, d\}$. Accordingly, using the result of Corollary 12, the maximum in/out flow for an additive flow network \mathcal{N} can be computed in time $\mathcal{O}(n) \cdot \mathcal{O}(2^\kappa)$. □

3.6 Supplementary Proofs and Lemmas for Section 3.5

3.6.1 Proof of Lemma 13

For the proof of Lemma 13, the additive network $\mathcal{N} = (\mathbf{V}, \mathbf{E}, U, g)$ in the original form (Definition 1) contains no one-vertex cycles (self-loops), but may contain two-vertex cycles, *i.e.*, \mathcal{N} may contain distinct vertices $v_1, v_2 \in \mathbf{V}$ and distinct edges $e_1, e_2 \in \mathbf{E}$ with $head(e_i) = v_i$ and $tail(e_i) = v_j$ for all $\{i, j\} = \{1, 2\}$. We eliminate such a two-vertex cycle by inserting fresh vertices w_1 and w_2 in the middle of e_1 and e_2 , as well as a fresh edge $\langle w_1, w_2 \rangle$ with $U(\langle w_1, w_2 \rangle) = 0$, *i.e.*, edge $\langle w_1, w_2 \rangle$ is dummy. By eliminating all two-vertex cycles in \mathcal{N} , we add fewer than m new vertices and fewer than $\lceil m/2 \rceil$ new edges. If \mathcal{N} is planar and given with a k -outerplanar embedding, this elimination of all two-vertex cycles does not increase the outerplanarity k if $k \geq 2$, though it may increase it by 1 if $k = 1$, as one can readily verify.

Consider the transformation from the original $\mathcal{N} = (\mathbf{V}, \mathbf{E}, U, g)$ to $\mathcal{N}' = (\mathbf{V}', \mathbf{E}', U', g')$ in modified form (Definition 2) according to the proof of Proposition 3. By the preceding argument, there are no one-vertex and no two-vertex cycles in \mathcal{N} , and therefore there are no one-vertex and no two-vertex cycles in \mathcal{N}' either.

Lemma 13 is about the underlying graph of network \mathcal{N} , with no mention of its capacity function U and gain function g . Hence, if $\mathcal{N} = (\mathbf{V}, \mathbf{E}, U, g)$ and its transformation into modified form is $\mathcal{N}' = (\mathbf{V}', \mathbf{E}', U', g')$, we can focus on the underlying graphs $G = (\mathbf{V}, \mathbf{E})$ and $G' = (\mathbf{V}', \mathbf{E}')$ instead for its proof. Moreover, the conclusion of Lemma 13 is independent of edge directions, and we can thus view G and G' as simple undirected graphs (with no self-loops and no multi-edges because of the absence of two-vertex cycles in \mathcal{N} and \mathcal{N}').

The presence of input and output edges does not affect the outerplanarity index. We can thus further view G and G' as simple undirected graphs without “dangling” edges, having removed all input and output edges.

Based on the preceding argument, the proof of Lemma 13 is a consequence of a simpler graph-theoretic result, Lemma 17 below. We first introduce a useful classification of the edges of a k -outerplanar embedding. We write \overline{vw} to denote the two-element set $\{v, w\}$ representing the

undirected edge between vertices v and w .

Definition 16 (*Peeling Edges and Cross Edges*). Let $G = G_1 = (\mathbf{V}, \mathbf{E})$ be a planar graph, given with a specific planar embedding. We define L_1 as the set of vertices incident to $\text{OuterFace}(G_1)$, and define L_i for $i > 1$ recursively as the set of vertices incident to $\text{OuterFace}(G_i)$, where G_i is the planar embedding obtained after deleting all the vertices in $L_1 \cup \dots \cup L_{i-1}$ and all the edges incident to them.

We call L_i , for $i \geq 1$, the i -th *peeling* of the given planar embedding of G . If the outerplanarity of the planar embedding is k , then there are k non-empty peelings. We pose $G_{k+1} = \emptyset$, the empty graph obtained after deleting the k -th and last non-empty peeling L_k .

We call an edge e which is bounding $\text{OuterFace}(G_i)$ a *level- i peeling edge*. If we ignore the level of e , we simply say e is a *peeling edge*. The two endpoints of e are necessarily two distinct vertices in L_i .

All the edges of G which are not peeling edges are called *cross edges*. If $e = \overline{vw}$ is a cross edge with endpoints v and w , then there are one of two cases:

- *either* there are two consecutive peelings L_i and L_{i+1} , with $1 \leq i < k$, such that $v \in L_i$ and $w \in L_{i+1}$,
- *or* there is a peeling L_i , with $1 \leq i \leq k$, such that both $v, w \in L_i$ and e is not bounding $\text{OuterFace}(G_i)$.

In either case, a cross edge e bounds two adjacent inner faces of G_i and is one of the edges to be deleted when we define G_{i+i} from G_i .

We have thus classified all the edges in a k -outerplanar embedding of an undirected simple graph G into: (1) the peeling edges, which are further partitioned into k disjoint levels, and (2) the cross edges. □

Lemma 17. *Let $G = (\mathbf{V}, \mathbf{E})$ be a simple undirected graph, given in a k -outerplanar embedding. Let $G' = (\mathbf{V}', \mathbf{E}')$ be the simple undirected graph obtained by replacing every vertex $v \in \mathbf{V}$ such that $\text{degree}(v) \geq 4$, with $\{\overline{v_1 v}, \dots, \overline{v_d v}\}$ being its set of incident edges, by an appropriate cycle defined as:*

1. Introduce fresh vertices $\{w_1, \dots, w_d\}$.
2. Introduce fresh edges $\{\overline{w_1 w_2}, \dots, \overline{w_{d-1} w_d}, \overline{w_d w_1}\}$ to create an undirected cycle with d vertices.
3. Replace every edge $\overline{v_i v}$ by a fresh edge $\overline{v_i w_i}$ for every $1 \leq i \leq d$.

Conclusion: The resulting G' is a planar graph, returned in a k' -outerplanar embedding, with $k' \leq 2k$.

Proof. The proof is by induction on the outerplanarity $k \geq 1$. We omit the straightforward proof for the case $k = 1$: The construction of G' produces a planar embedding with outerplanarity ≤ 2 and where every vertex has degree ≤ 3 . To be more specific, if G has an inner face F , a vertex v on the boundary of F with $\text{degree}(v) \geq 4$, and an edge $\overline{v w}$ not contained in $\text{OuterFace}(G)$, then the new G' has outerplanarity 2. Otherwise, if this condition is not satisfied, G' has outerplanarity 1.

Proceeding inductively, the *induction hypothesis* (IH) assumes that, given an arbitrary planar G with a planar embedding of outerplanarity $k \geq 1$, the transformation described in the lemma statement produces a planar G' with a planar embedding of outerplanarity $k' \leq 2k$ and where every vertex has degree ≤ 3 .

We prove the result again for an arbitrary planar graph G with a planar embedding of outerplanarity $k + 1$. For every vertex $v \in \mathbf{V}$ we introduce an additional set of vertices, which we call *hooks*, each denoted $\text{hook}_\ell(v)$ for some $\ell \geq 1$. If $\{w_1, \dots, w_d\}$ are all the vertices incident to v , then the set of hooks associated with v is:

$$\text{hook}_*(v) := \{\text{hook}_1(v), \dots, \text{hook}_d(v)\}$$

For every edge $\overline{v w} \in \mathbf{E}$, we have thus introduced two fresh vertices, $\text{hook}_\ell(v)$ and $\text{hook}_{\ell'}(w)$ for some $\ell, \ell' \geq 1$. Think of the edge $\overline{v w}$ as being cut in two halves, with two new edges being introduced:

$$\overline{v \text{hook}_{\ell'}(w)} \quad \text{and} \quad \overline{\text{hook}_\ell(v) w}.$$

Let P and Q be the first and second peelings of G , respectively. P and Q are disjoint subsets of

vertices. We define two graphs $G_1 = (\mathbf{V}_1, \mathbf{E}_1)$ and $G_2 = (\mathbf{V}_2, \mathbf{E}_2)$ from the $(k + 1)$ -outerplanar G :

$$\mathbf{V}_1 := P \cup \{ \text{hook}_\ell(w) \mid w \in Q, \ell \geq 1 \text{ and there is } v \in P \text{ such that } \overline{vw} \in \mathbf{E} \}$$

$$\mathbf{E}_1 := \{ \overline{vw} \mid v, w \in P \text{ and } \overline{vw} \in \mathbf{E} \} \cup \{ \overline{v \text{ hook}_\ell(w)} \mid v \in P, w \in Q, \ell \geq 1 \text{ and } \overline{vw} \in \mathbf{E} \}$$

$$\mathbf{V}_2 := (\mathbf{V} - P) \cup \{ \text{hook}_\ell(v) \mid v \in P, \ell \geq 1 \text{ and there is } w \in Q \text{ such that } \overline{vw} \in \mathbf{E} \}$$

$$\mathbf{E}_2 := \{ \overline{vw} \mid v, w \in (\mathbf{V} - P) \text{ and } \overline{vw} \in \mathbf{E} \} \cup \{ \overline{\text{hook}_\ell(v) w} \mid v \in P, w \in Q, \ell \geq 1 \text{ and } \overline{vw} \in \mathbf{E} \}$$

Observe that every hook, *i.e.*, a vertex of the form $\text{hook}_\ell(v)$ has degree = 1, and that every edge of the form $\overline{v \text{ hook}_\ell(w)}$ or $\overline{\text{hook}_\ell(v) w}$ is entirely contained in both $\text{OuterFace}(G_1)$ and $\text{OuterFace}(G_2)$. We need to distinguish between *open edges* and *closed edges* of G_1 and G_2 . For G_1 first:

$$\mathbf{E}_{1,\text{open}} := \{ \overline{v \text{ hook}_\ell(w)} \mid v \in P, w \in Q, \ell \geq 1 \text{ and } \overline{vw} \in \mathbf{E} \} \quad \text{the open edges of } G_1$$

$$\mathbf{E}_{1,\text{closed}} := \mathbf{E}_1 - \mathbf{E}_{1,\text{open}} \quad \text{the closed edges of } G_1$$

And similarly for G_2 :

$$\mathbf{E}_{2,\text{open}} := \{ \overline{\text{hook}_\ell(v) w} \mid v \in P, w \in Q, \ell \geq 1 \text{ and } \overline{vw} \in \mathbf{E} \} \quad \text{the open edges of } G_2$$

$$\mathbf{E}_{2,\text{closed}} := \mathbf{E}_2 - \mathbf{E}_{2,\text{open}} \quad \text{the closed edges of } G_2$$

An *open edge* is therefore an edge with a hook as one of its two endpoints, which is always of degree = 1. The graphs G_1 and G_2 are planar, and their definitions are such that they produce a planar embedding of G_1 with outerplanarity = 2 and a planar embedding of G_2 with outerplanarity = k . These assertions follow from the two facts below, together with the fact that the presence of open edges drawn inward (as in G_1) increases outerplanarity by 1, and drawn outward (as in G_2) does not increase outerplanarity:

- If we delete every open edge in G_1 , we obtain the 1-outerplanar subgraph of G induced by P .
- If we delete every open edge in G_2 , we obtain the k -outerplanar subgraph of G induced by $(\mathbf{V} - P)$.

An example of how G is broken up into two graphs G_1 and G_2 is shown in Figure 3.5.

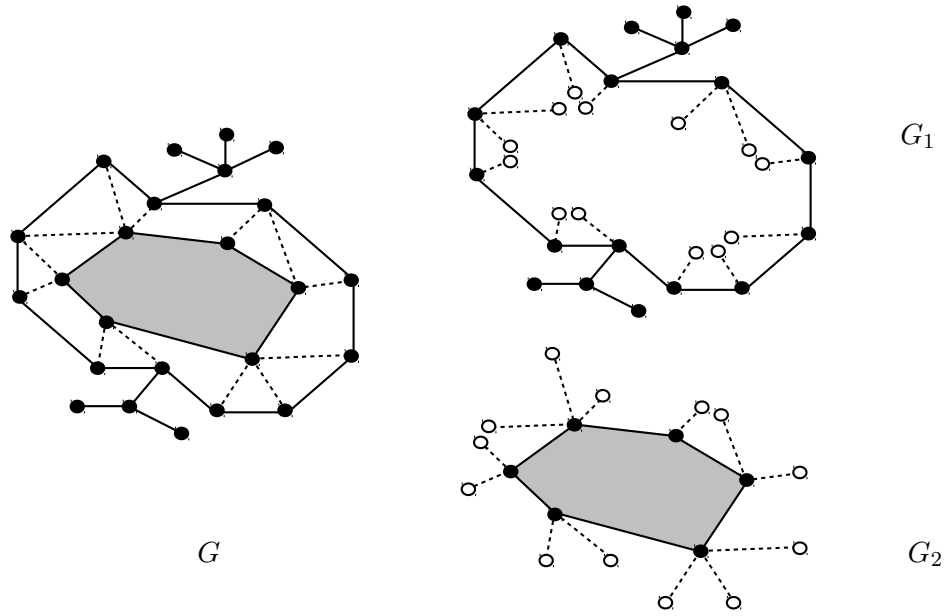


Figure 3.5: Example for the proof of Lemma 17. The outermost edges in G form its first peeling; the edges enclosing the shaded area form its second peeling; the shaded area, including the nodes of the second peeling, is a subgraph of G of outerplanarity k . G_1 and G_2 are shown in planar embeddings of outerplanarity 2 and k , respectively. An *open edge* (dashed line) in G_1 and G_2 has a *hook* (white node) as one of its two endpoints.

We can “re-build” G from G_1 and G_2 as follows:

$$\mathbf{V} = (\mathbf{V}_1 \cup \mathbf{V}_2) - (\text{hook}_*(G_1) \cup \text{hook}_*(G_2))$$

$$\mathbf{E} = \mathbf{E}_{1,\text{closed}} \cup \mathbf{E}_{2,\text{closed}} \cup \left\{ \overline{vw} \mid \overline{v \text{ hook}_\ell(w)} \in \mathbf{E}_{1,\text{open}} \text{ and } \overline{\text{hook}_\ell(v) w} \in \mathbf{E}_{2,\text{open}} \right\}$$

where $\text{hook}_*(G_i) := \{ \text{hook}_\ell(v) \mid \text{there is } w \in \mathbf{V} \text{ and } \ell \geq 1 \text{ such that } \overline{vw} \in \mathbf{E} \}$ with $i = 1, 2$. The

two preceding equalities are easily checked and make explicit the way in which we use hooks and open edges to connect G_1 and G_2 .

Let G'_1 be the graph obtained from G_1 according to the transformation defined in the lemma statement, which produces a planar embedding of G'_1 with outerplanarity ≤ 2 . And let G'_2 be the graph obtained from G_2 according to the transformation defined in the lemma statement, which, by the IH, produces a planar embedding of G'_2 with outerplanarity $\leq 2k$.

We note carefully how open edges in G_1 may get transformed into open edges in G'_1 . Consider a vertex $v \in P$ with $\text{degree}(v) = d \geq 4$ and let the open edges of G_1 that have v as one of their two endpoints be:

$$\overline{v \text{ hook}_{\ell_1}(w_1)}, \dots, \overline{v \text{ hook}_{\ell_t}(w_t)}$$

where $1 \leq t \leq d$. The number t of open edges with endpoint v is not necessarily d . The corresponding open edges in G'_1 are:

$$\overline{v_{i_1} \text{ hook}_{\ell_1}(w_1)}, \dots, \overline{v_{i_t} \text{ hook}_{\ell_t}(w_t)}$$

where $\{v_{i_1}, \dots, v_{i_t}\} \subseteq \{v_1, \dots, v_d\}$, and $\{v_1, \dots, v_d\}$ is the set of fresh vertices in the simple cycle that replaces v in G'_1 . Note that the transformation from G_1 to G'_1 does not affect the second endpoints (the hooks) of these open edges, because the degree of a hook is always = 1. Similar observations apply to the way in which open edges in G_2 get transformed into open edges in G'_2 .

We are ready to define the desired graph G' by connecting G'_1 and G'_2 via their hooks and open edges, in the same way in which we can re-connect G from G_1 and G_2 :

$$\mathbf{V}' = (\mathbf{V}'_1 \cup \mathbf{V}'_2) - (\text{hook}_*(G'_1) \cup \text{hook}_*(G'_2))$$

$$\mathbf{E}' = \mathbf{E}'_{1,\text{closed}} \cup \mathbf{E}'_{2,\text{closed}} \cup \left\{ \overline{v_i w_j} \mid \overline{v_i \text{ hook}_{\ell}(w)} \in \mathbf{E}_{1,\text{open}} \text{ and } \overline{\text{hook}_{\ell}(v) w_j} \in \mathbf{E}_{2,\text{open}}, \right. \\ \left. \text{with } 1 \leq i \leq \text{degree}(v) \text{ and } 1 \leq j \leq \text{degree}(w) \right\}$$

This produces a planar graph G' together with a planar embedding. To conclude the induction and the proof, it suffices to note that the outerplanarity of G' is “the outerplanarity of G'_1 ” + “the

outerplanarity of G'_2 ” which is therefore $\leq 2(k+1)$. \square

3.6.2 Proof of Lemma 14

We need another lemma before proving Lemma 14. We defined *graph reassembling* in the opening paragraphs of Section 3.5 in relation to networks and their underlying directed graphs. This notion applies equally well to undirected graphs. The reassembling sequence in Theorem 11 and again in Lemma 14 is said to be *linear*, because a single *main component* is reassembled to which *one-vertex components* are added one by one, for a total of $(n-1)$ steps.

Lemma 18. *Let $G = (\mathbf{V}, \mathbf{E})$ be a simple 3-regular undirected graph, given in a k -outerplanar embedding, with $|\mathbf{V}| = n \geq 1$ and $k \geq 1$. **Conclusion:** A linear reassembling of G can be carried out in time $\mathcal{O}(n)$ such that the edge-boundary degree of the main component at any step during reassembling is $\leq 2(k+1)$.*

The next definition, and lemma based on it, are not essential, but, together with the preceding assumption, they considerably simplify Algorithm 1 and proving its correctness as the base for the proof of Lemma 18.

Assumption 19. From now on, there is no loss of generality if we assume that:

1. Networks are connected.
2. Networks are 3-regular.
3. There are no two-vertex cycles in networks.

The second and third conditions follow from the construction in the proof of Lemma 13. \square

Definition 20 (Good Planar Embeddings). Let $\mathcal{N} = (\mathbf{V}, \mathbf{E})$ be a network satisfying Assumption 19 and given in a fixed k -outerplanar embedding, for some $k \geq 1$. From the peelings L_1, \dots, L_k specified in Definition 16, we define the sets of vertices L'_1, \dots, L'_k , respectively, as follows. For every $1 \leq i \leq k$:

$$L'_i := L_i - \{ \nu \in L_i \mid \nu \text{ is incident to at most one peeling edge} \}.$$

In other words, L'_i is a subset of L_i which is proper whenever L_i contains a vertex ν such that:

1. ν is incident to three cross edges.
2. ν is incident to two cross edges and one input/output edge.
3. ν is incident to one cross edge and one input/output edge.

Thus, L'_i is defined to exclude all the vertices of L_i that are of degree ≤ 1 in the network \mathcal{N}_i (see Definition 16).

We say the planar embedding of \mathcal{N} is *good* if for every $1 \leq i \leq k$, the vertices in L'_i form a single (undirected) simple cycle, namely, the outermost one, in the network \mathcal{N}_i . \square

Example 21. For an example of how L'_i may be different from L_i , consider the 3-outerplanar embedding in Figure 3.7 (originally constructed from the graph depicted on the left side of Figure 3.6): $L_1 = L'_1$ and $L_3 = L'_3$, but $L_2 \neq L'_2$. The latter inequality is caused by one of the vertices on the periphery of the south-east face, which is incident to one cross edge and one input/output edge. This vertex is shown by a dashed circle around it.

In a good planar embedding, the sets L'_1, \dots, L'_k can be viewed as forming k concentric simple cycles. All the vertices in $(L_i - L'_i)$ lay between the level- i concentric cycle and the one immediately enclosing it (the level- $(i - 1)$ concentric cycle). This implies that, if \mathcal{N} is 3-regular and no two distinct input/output edges are incident to the same vertex of the outer layer L_1 (as required by Assumption 19), *i.e.* $L_1 = L'_1$, but we may have $L_i \neq L'_i$ for $i \geq 2$. The 3-outerplanar embedding in Figure 3.7 is good. \square

Lemma 22 (From Planar Embeddings to Good Planar Embeddings). *Let $\mathcal{N} = (\mathbf{V}, \mathbf{E})$ be a network satisfying Assumption 19 and given in a specific planar embedding. In time $\mathcal{O}(n)$, where $n = |\mathbf{V}|$, we can transform the given planar embedding of \mathcal{N} into a planar embedding of an equivalent $\mathcal{N}' = (\mathbf{V}', \mathbf{E}')$ such that:*

1. *The planar embedding of \mathcal{N}' is good (and, in particular, \mathcal{N}' satisfies Assumption 19).*
2. *$|\mathbf{V}'| \leq 2 \cdot |\mathbf{V}|$ and $|\mathbf{E}'| \leq 2 \cdot |\mathbf{E}|$.*
3. *\mathcal{N} and \mathcal{N}' have the same outerplanarity.*

Proof. Straightforward, by appropriately inserting extra vertices and *dummy edges*, also making sure not to violate 3-regularity and not to increase outerplanarity. An edge e is dummy edge if $U(e) = 0$, i.e., e cannot carry any flow and therefore cannot affect the overall flow properties of the network. \square

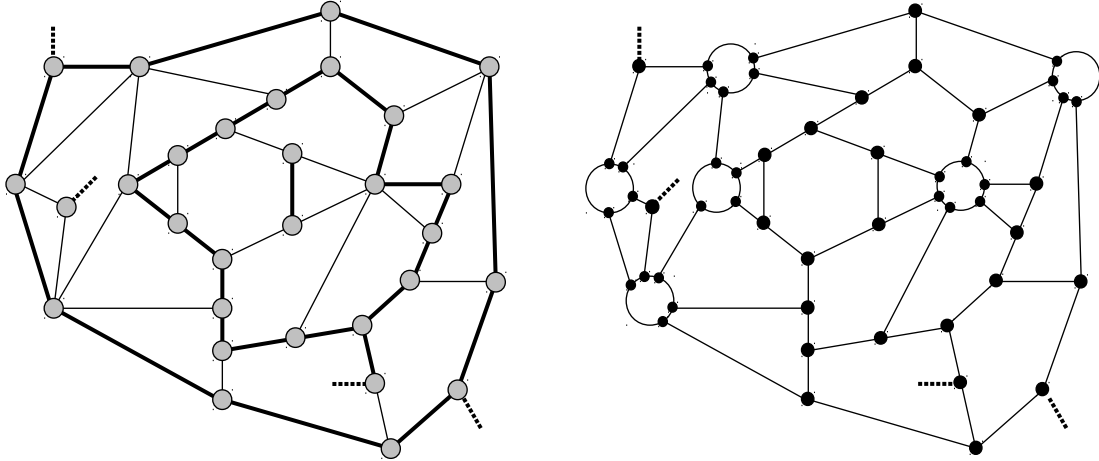


Figure 3.6: Example of a planar network \mathcal{N} (with all edge directions ignored) on the left, its transformation into a 3-regular network \mathcal{N}' according to Lemma 17 on the right. The dashed edges are input/output edges, 4 of them.

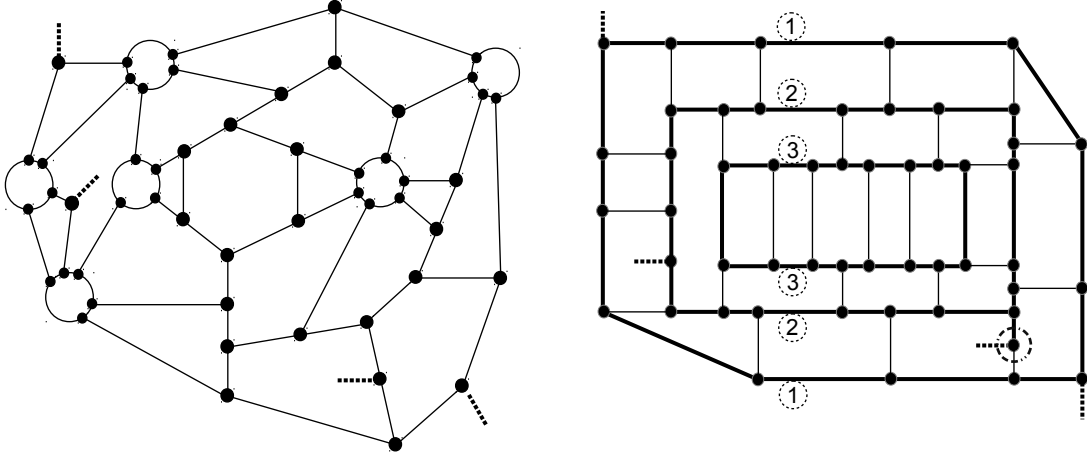


Figure 3.7: The planar network \mathcal{N}' on the right in Figure 3.6 is reproduced without change on the left in this figure, and re-drawn on a rectangular grid on the right in this figure – except for two edges because of the missing north-east corner and south-west corner. The dashed edges are input/output edges, 4 of them.

Definition 23 (*Neighbor Subnetworks and their Merge*). Let $\mathcal{N} = (\mathbf{V}, \mathbf{E})$ be a network, and consider two non-empty disjoint subsets of vertices: $X, X' \subseteq \mathbf{V}$ with $X \cap X' = \emptyset$. Let \mathcal{M} and \mathcal{M}' be the subnetworks of \mathcal{N} induced by X and X' , respectively. Let $\mathbf{B}_{\text{in,out}}$ and $\mathbf{B}'_{\text{in,out}}$ be the input/output edges of \mathcal{M} and \mathcal{M}' . We say \mathcal{M} and \mathcal{M}' are *neighbor subnetworks*, or just *neighbors*, iff $|\mathbf{B}_{\text{in,out}} \cap \mathbf{B}'_{\text{in,out}}| \geq 1$, i.e., \mathcal{M} and \mathcal{M}' have one input/output edge or more in common. We refer to the sequence of edges in $\mathbf{B}_{\text{in,out}} \cap \mathbf{B}'_{\text{in,out}}$ listed according to some fixed (but otherwise arbitrary) ordering scheme as the *sequence of joint edges* of \mathcal{M} and \mathcal{M}' :

$$\text{joint-arcs}(\mathcal{M}, \mathcal{M}') := \text{a fixed ordering of the edges in } \mathbf{B}_{\text{in,out}} \cap \mathbf{B}'_{\text{in,out}}.$$

Observe that we restrict the notion of “neighbors” to two subnetworks \mathcal{M} and \mathcal{M}' induced by disjoint subsets of vertices X and X' , but which share some input/output edges.

To *merge* \mathcal{M} and \mathcal{M}' means to produce the subnetwork of \mathcal{N} induced by $X \cup X'$, which we denote $(\mathcal{M} \otimes \mathcal{M}')$. If \mathcal{M} and \mathcal{M}' are not neighbors, then $(\mathcal{M} \otimes \mathcal{M}')$ is undefined.⁹ \square

⁹ $(\mathcal{M} \otimes \mathcal{M}')$ can be written in terms of the Bind operation defined in Section 3.4, by applying it as many times as

Definition 24 (*Strong Neighbors*). Let $\mathcal{N} = (\mathbf{V}, \mathbf{E})$ be a network, and \mathcal{M}' and \mathcal{M}'' be neighbors in \mathcal{N} induced by the disjoint subsets of vertices X' and X'' , as in Definition 23. We define the *binding strength* of the neighbors \mathcal{M}' and \mathcal{M}'' as follows:

$$\text{binding-strength}(\mathcal{M}', \mathcal{M}'') := |\text{joint-arcs}(\mathcal{M}', \mathcal{M}'')|.$$

Because \mathcal{M}' and \mathcal{M}'' are neighbors, $\text{binding-strength}(\mathcal{M}', \mathcal{M}'') \geq 1$. The external dimension of $\mathcal{M}' \oplus \mathcal{M}''$ is:

$$\text{exDim}(\mathcal{M}' \oplus \mathcal{M}'') = \text{exDim}(\mathcal{M}') + \text{exDim}(\mathcal{M}'') - 2 \cdot \text{binding-strength}(\mathcal{M}', \mathcal{M}'').$$

We say \mathcal{M}' and \mathcal{M}'' are *strong neighbors* if the following inequality is satisfied:

$$\begin{aligned} \text{exDim}(\mathcal{M}' \oplus \mathcal{M}'') &\leq \\ &\min \left(\{ \text{exDim}(\mathcal{M}' \oplus \mathcal{M}) \mid \mathcal{M} \text{ is a neighbor of } \mathcal{M}' \} \right. \\ &\quad \left. \cup \{ \text{exDim}(\mathcal{M}'' \oplus \mathcal{M}) \mid \mathcal{M} \text{ is a neighbor of } \mathcal{M}'' \} \right). \end{aligned}$$

Equivalently, \mathcal{M}' and \mathcal{M}'' are *strong neighbors* if:

$$\begin{aligned} \text{binding-strength}(\mathcal{M}', \mathcal{M}'') &\geq \\ &\max \left(\{ \text{binding-strength}(\mathcal{M}', \mathcal{M}) \mid \mathcal{M} \text{ is a neighbor of } \mathcal{M}' \} \right. \\ &\quad \left. \cup \{ \text{binding-strength}(\mathcal{M}'', \mathcal{M}) \mid \mathcal{M} \text{ is a neighbor of } \mathcal{M}'' \} \right). \end{aligned}$$

In other words, \mathcal{M}' and \mathcal{M}'' are *strong neighbors* if they have at least as many external edges in common as each has in common with another neighbor \mathcal{M} . □

In Algorithm 1, we use repeatedly the same group of instructions, which we here collect together as a single “macro” instruction called Merge. Let $X_1 \uplus \dots \uplus X_p = \mathbf{V}$ be a partition of the vertices of the given network \mathcal{N} . Let $\mathcal{C} = \{\mathcal{M}_1, \dots, \mathcal{M}_p\}$ be the subnetworks of \mathcal{N} induced

there are edges in $\text{joint-arcs}(\mathcal{M}, \mathcal{M}')$.

by X_1, \dots, X_p , respectively. Let $\mathbf{B}_\#^1, \dots, \mathbf{B}_\#^p$ be the (necessarily disjoint) sets of *internal* edges of $\mathcal{M}_1, \dots, \mathcal{M}_p$, respectively. With \mathcal{N} thus disassembled, if we select two distinct subnetworks $\mathcal{M}, \mathcal{M}' \in \mathcal{C}$ that are neighbors, we write Merge with 5 arguments as:

$$\text{Merge}(\mathcal{M}, \mathcal{M}', \Theta, \partial, \mathcal{C})$$

where the last 3 are the following quantities:

- Θ = an ordering of the edges in $\mathbf{B}_\#^1 \cup \dots \cup \mathbf{B}_\#^p$ (the binding schedule computed by the algorithm)
- $\partial \geq 3$ (a tight upper bound on $\text{index}(\Theta)$, *i.e.* the maximum edge boundary degree of components in \mathcal{C})
- $\mathcal{C} = \{\mathcal{M}_1, \dots, \mathcal{M}_p\}$

The notions of a “binding schedule” and its “index” were defined in Section 3.4. The macro expansion of $\text{Merge}(\mathcal{M}, \mathcal{M}', \Theta, \partial, \mathcal{C})$ is shown in Figure 3.8.

1. $\Theta := \Theta \text{ joint-arcs}(\mathcal{M}, \mathcal{M}')$	//append $\text{joint-arcs}(\mathcal{M}, \mathcal{M}')$ to Θ
2. $\partial := \max\{\partial, \text{exDim}(\mathcal{M}) + \text{exDim}(\mathcal{M}') - 2\}$	//new tight upper bound on $\text{index}(\Theta)$
3. $\mathcal{C} := (\mathcal{C} - \{\mathcal{M}, \mathcal{M}'\}) \cup \{\mathcal{M} \oplus \mathcal{M}'\}$	//exclude \mathcal{M} and \mathcal{M}' , include their merge $\mathcal{M} \oplus \mathcal{M}'$

Figure 3.8: Macro expansion of $\text{Merge}(\mathcal{M}, \mathcal{M}', \Theta, \partial, \mathcal{C})$.

Instead of the three instructions shown in Figure 3.8, we can now write a single macro instruction:

$$(\Theta, \partial, \mathcal{C}) := \text{Merge}(\mathcal{M}, \mathcal{M}', \Theta, \partial, \mathcal{C})$$

We need one more classification of edges before we define Algorithm 1. Let network $\mathcal{N} = (\mathbf{V}, \mathbf{E})$ be given in a good k -outerplanar embedding, with $\mathbf{E} = \mathbf{E}_{\text{in,out}} \uplus \mathbf{E}_\#$. Using Lemma ??, we partition the internal edges of \mathcal{N} into two parts, $\mathbf{E}_\# = \mathbf{E}_{\#,1} \uplus \mathbf{E}_{\#,2}$, where:

$$\mathbf{E}_{\#,1} := \{a \in \mathbf{E}_\# \mid a \text{ is a cross edge}\} \cup$$

$$\{a \in \mathbf{E}_\# \mid \text{there is } 1 \leq i \leq k \text{ such that } \{head(a), tail(a)\} \cap (L_i - L'_i) \neq \emptyset\},$$

$$\mathbf{E}_{\#,2} := \mathbf{E}_\# - \mathbf{E}_{\#,1}.$$

In words, $\mathbf{E}_{\#,1}$ is the set of: (1) all cross edges, and (2) all peeling edges on a path connecting two consecutive concentric cycles of the good embedding of \mathcal{N} . See the statement of Lemma ?? for further explanation.

Example 25. This is a continuation of the network considered in Example 21. It refer to the good 3-outerplanar embedding on the right in Figure 3.6, and again in Figure 3.7, which we use to illustrate the operation of Algorithm 1. The progress of Algorithm 1 is shown in Figure 3.9, for the **first iteration** and the **second iteration**, and in Figure 3.10 for the **main iteration**. \square

Proof 26 (for Lemma 18). Let \mathcal{N}_0 be the network in the statement of Lemma 18, to distinguish it from the “ \mathcal{N} ” introduced below. Let $\mathcal{N}_0 = (\mathbf{V}_0, \mathbf{E}_0)$ be given in a k_0 -outerplanar embedding, for some $k_0 \geq 1$. Let $p = |\mathbf{E}_{0,\text{in}}| \geq 1$, $q = |\mathbf{E}_{0,\text{out}}| \geq 1$, $m_0 = |\mathbf{E}_{0,\#}| \geq 1$ and $n_0 = |\mathbf{V}_0| \geq 1$. Because \mathcal{N}_0 is planar, \mathcal{N}_0 is sparse; more specifically, $m_0 \leq 3n_0 - 6$. Hence, the complexity bound $\mathcal{O}(m_0 + n_0)$ is the same as $\mathcal{O}(n_0)$.

By Lemma 22, we can transform the k_0 -outerplanar embedding of \mathcal{N}_0 into a *good* k -outerplanar embedding of an equivalent \mathcal{N} , with $k = k_0$. The transformation is such that $\text{exDim}(\mathcal{N}_0) = \text{exDim}(\mathcal{N}) = p + q$. Let $m = |\mathbf{E}_\#|$ and $n = |\mathbf{V}|$. By Lemma 22, $m \leq 2m_0$ and $n \leq 2n_0$.

We next run Algorithm 1 on the good k -outerplanar embedding of \mathcal{N} . We first consider the correctness of the algorithm, and then its run-time complexity. The initialization consists in breaking up \mathcal{N} into n one-vertex subnetworks, each of external dimension 3.

The **first iteration** assembles new subnetworks \mathcal{M} of external dimension 4, if we ignore the presence of all input/output edges of \mathcal{N} .¹⁰ See Figure 3.9 for an illustration. Such a subnetwork \mathcal{M} of external dimension 4 has two vertices – say $\{\nu_1, \nu_2\}$ – that are *either* on the same peeling level *or* on two consecutive peeling levels. Specifically, there is $1 \leq i \leq k$, such that *either* both $\nu_1, \nu_2 \in L'_i$ *or* $\nu_1 \in L'_i$ and $\nu_2 \in L'_{i+1}$.

¹⁰By “ignoring an input/output edge a ”, we mean that we omit a but not the input/output vertex ν to which a is incident. The vertex ν is thus temporarily made to have degree 2.

Algorithm 1 BindSchedule: Define Optimal Binding Schedule

input: good planar embedding of network $\mathcal{N} = (\mathbf{V}, \mathbf{E})$, with $\mathbf{E} = \mathbf{E}_{\text{in,out}} \uplus \mathbf{E}_{\#,1} \uplus \mathbf{E}_{\#,2}$
output: $\Theta = b_1 b_2 \dots b_m$, an ordering of internal edges of \mathcal{N} (a “binding schedule”),
 where $\mathbf{E}_{\#} = \{b_1, b_2, \dots, b_m\}$, together with a tight upper bound ∂ on $\text{index}(\Theta)$.

initialization

- 1: $k :=$ outerplanarity of \mathcal{N}
 - 2: $\Theta := \varepsilon$ // Θ is initially the empty “binding schedule”
 - 3: $\mathcal{C} := \{ \mathcal{M} \mid \mathcal{M} \text{ subnetwork of } \mathcal{N} \text{ induced by } \{\nu\} \text{ with } \nu \in \mathbf{V} \}$
 // \mathcal{N} is disassembled into $|\mathcal{N}|$ one-vertex subnetworks, each of external dimension 3
 - first iteration** // pre-processing
 - 4: **for** every edge $a \in \mathbf{E}_{\#,1}$ **do**
 - 5: $(\Theta, \partial, \mathcal{C}) := \text{Merge}(\mathcal{M}, \mathcal{M}', \Theta, \partial, \mathcal{C})$
 where $\mathcal{M}, \mathcal{M}' \in \mathcal{C}$ are the two subnetworks such that $a \in \text{joint-arcs}(\mathcal{M}, \mathcal{M}')$
 - 6: **end for** // every edge $a \in \mathbf{E}_{\#,1}$ is now included in Θ ,
 // for every $\mathcal{M} \in \mathcal{C}$ such that $|\mathcal{M}| = 1$, the single vertex of \mathcal{M} is an input/output vertex,
 // for every $\mathcal{M} \in \mathcal{C}$ such that $|\mathcal{M}| \geq 2$, ignoring input/output edges of \mathcal{N} , $\text{exDim}(\mathcal{M}) = 4$
 - second iteration** // pre-processing
 - 7: **while** there are neighbors $\mathcal{M}, \mathcal{M}' \in \mathcal{C}$ such that: $|\mathcal{M}| = 1$ **or** $\text{binding-strength}(\mathcal{M}, \mathcal{M}') = 2$ **do**
 - 8: $(\Theta, \partial, \mathcal{C}) := \text{Merge}(\mathcal{M}, \mathcal{M}', \Theta, \partial, \mathcal{C})$
 - 9: **end while** // for every $\mathcal{M} \in \mathcal{C}$, ignoring input/output edges of \mathcal{N} , $\text{exDim}(\mathcal{M}) = 4$
 - main iteration** // re-assemble \mathcal{N} from the subnetworks in \mathcal{C} and store it in $\tilde{\mathcal{N}}$
 - 10: $\tilde{\mathcal{N}} := \mathcal{M}$ where \mathcal{M} is any “outermost” subnetwork in \mathcal{C}
 - 11: $\mathcal{C} := \mathcal{C} - \{\tilde{\mathcal{N}}\}$
 - 12: **while** $\mathcal{C} \neq \emptyset$ **do**
 - 13: select $\mathcal{M} \in \mathcal{C}$ which is a strong neighbor of $\tilde{\mathcal{N}}$
 - 14: $\Theta := \Theta \text{ joint-arcs}(\tilde{\mathcal{N}}, \mathcal{M})$
 - 15: $\partial := \max\{\partial, \text{exDim}(\tilde{\mathcal{N}}) + \text{exDim}(\mathcal{M}) - 2\}$
 - 16: $\tilde{\mathcal{N}} := \tilde{\mathcal{N}} \oplus \mathcal{M}$
 - 17: $\mathcal{C} := \mathcal{C} - \{\mathcal{M}\}$
 - 18: **end while** // \mathcal{N} is now re-assembled and stored in $\tilde{\mathcal{N}}$
 - 19: **return** Θ and ∂
-

A similar conclusion applies to the **second iteration**: It assembles new subnetworks \mathcal{M} of external dimension 4, again ignoring the presence of all input/output edges of \mathcal{N} . See Figure 3.9 for an illustration. Such a subnetwork has external dimension 4, with four input/output vertices (these are *not* the same as the input/output vertices of \mathcal{N}) – say $\{\nu_1, \nu_2, \nu_3, \nu_4\}$ – that are *either* all on the same peeling level *or* on two consecutive peeling levels with two vertices on each. Specifically, there is $1 \leq i \leq k$, such that *either* both $\{\nu_1, \nu_2, \nu_3, \nu_4\} \subseteq L'_i$ *or* $\{\nu_1, \nu_2\} \subseteq L'_i$ and $\{\nu_3, \nu_4\} \subseteq L'_{i+1}$.

At the end of the **second iteration**, if we ignore all input/output edges of \mathcal{N} , every subnetwork \mathcal{M} in \mathcal{C} has external dimension 4 and is one of two kinds:

- \mathcal{M} is assembled in the **first iteration** and not affected by the **second iteration**. In this case, \mathcal{M} straddles *either* two opposite vertices of the same level L'_i *or* two vertices of two consecutive levels L'_i and L'_{i+1} .
- \mathcal{M} is assembled in the **second iteration** from two or more networks of the previous kind. In this case, \mathcal{M} straddles *either* two opposite peeling edges on the same level *or* two peeling edges on two consecutive levels.

At the end of the **second iteration**, for any two subnetworks $\mathcal{M}_1, \mathcal{M}_2 \in \mathcal{C}$, if \mathcal{M}_1 and \mathcal{M}_2 are neighbors, then $\text{binding-strength}(\mathcal{M}_1, \mathcal{M}_2) = 1$.

The task of the **main iteration** in Algorithm 1 is to re-assemble the original \mathcal{N} from the subnetworks in \mathcal{C} at the end of the **second iteration** in such a way as to minimize the external dimension of the intermediate subnetwork $\tilde{\mathcal{N}}$. We initialize $\tilde{\mathcal{N}}$ by selecting for it an “outermost” \mathcal{M} in \mathcal{C} (**line 10** of Algorithm 1), *i.e.*, we choose \mathcal{M} so that all its vertices are *either* all on level L'_1 *or* on two consecutive levels L'_1 and L'_2 .

The selection of the initial \mathcal{M} in the **main iteration** is totally arbitrary. For example, in the third assembly on the right of Figure 3.9, we choose for this initial \mathcal{M} the subnetwork containing the north-west corner of \mathcal{N} , and the corresponding progress of Algorithm 1 during the **main iteration** is shown in Figure 3.10.

To minimize the external dimension of $\tilde{\mathcal{N}}$ at every turn of the **main iteration**, it suffices to select any $\mathcal{M} \in \mathcal{C}$ which is a strong neighbor of $\tilde{\mathcal{N}}$ (**line 13** of Algorithm 1). For every $\mathcal{M} \in$

\mathcal{C} which is a neighbor of $\tilde{\mathcal{N}}$, we have $\text{binding-strength}(\tilde{\mathcal{N}}, \mathcal{M}) \geq 1$. Initially, $\text{exDim}(\tilde{\mathcal{N}}) = 4$ (ignoring all input/output edges of \mathcal{N}), and the maximum number of strong neighbors $\mathcal{M} \in \mathcal{C}$ such that $\text{binding-strength}(\tilde{\mathcal{N}}, \mathcal{M}) = 1$ in consecutive turns of the **main iteration** is $(k - 1)$. It is now easy to see that $\text{exDim}(\tilde{\mathcal{N}}) \leq 2k + 2$ is an invariant of the **main iteration**. Figure 3.10 shows how $\tilde{\mathcal{N}}$ may be assembled during the **main iteration**.

Consider now a subnetwork \mathcal{M} which is obtained by merging subnetworks \mathcal{M}' and \mathcal{M}'' , *i.e.*, $\mathcal{M} = \mathcal{M}' \oplus \mathcal{M}''$, where:

- $\text{exDim}(\mathcal{M}') = \ell' \geq 2$ and $\text{exDim}(\mathcal{M}'') = \ell'' \geq 2$,
- $\text{joint-arcs}(\mathcal{M}', \mathcal{M}'') = \{a_1, \dots, a_j\}$ where $j \leq \min\{\ell', \ell''\}$.

The edges in $\{a_1, \dots, a_j\}$ are re-connected one at a time, so that, starting from $\{\mathcal{M}', \mathcal{M}''\}$ and ending with \mathcal{M} , the merge operation produces j intermediate subnetworks (including \mathcal{M}) with external dimensions:

$$(\ell' + \ell'' - 2), (\ell' + \ell'' - 4), \dots, (\ell' + \ell'' - 2j),$$

respectively. Hence, while $\text{exDim}(\tilde{\mathcal{N}}) \leq 2k + 2$, the maximum external dimension encountered in the course of the operation of Algorithm 1 is – again ignoring all input/output edges of \mathcal{N} :

$$\begin{aligned} & \text{“maximum external dimension of } \tilde{\mathcal{N}}\text{”} + \text{“external dimension of all subnetworks in } \mathcal{C}\text{”} - 2 \\ & \leq (2k + 2) + 4 - 2 = 2k + 4, \end{aligned}$$

Hence, if we include the presence of the $p + q$ input/output edges of \mathcal{N} , the maximum external dimension of subnetworks produced during the entire operation of Algorithm 1 cannot exceed $2k + 4 + p + q$, which is precisely the final value assigned to ∂ by Algorithm 1, which is also $\leq 4k_0 + 4 + p + q$ where k_0 is the outerplanarity of the original network \mathcal{N}_0 .

To conclude the proof of Theorem 22, we need to show that the run-time complexity is $\mathcal{O}(n_0) = \mathcal{O}(n)$. This is a straightforward consequence of the fact that:

1. The initial transformation from \mathcal{N}_0 to \mathcal{N} is carried out in time $\mathcal{O}(n_0)$, according to Lemma 22.

2. Each of the four stages in Algorithm 1 (**initialization**, **first iteration**, **second iteration**, and **main iteration**) runs in time $\mathcal{O}(m)$, which is the same as $\mathcal{O}(m_0) = \mathcal{O}(n_0)$.

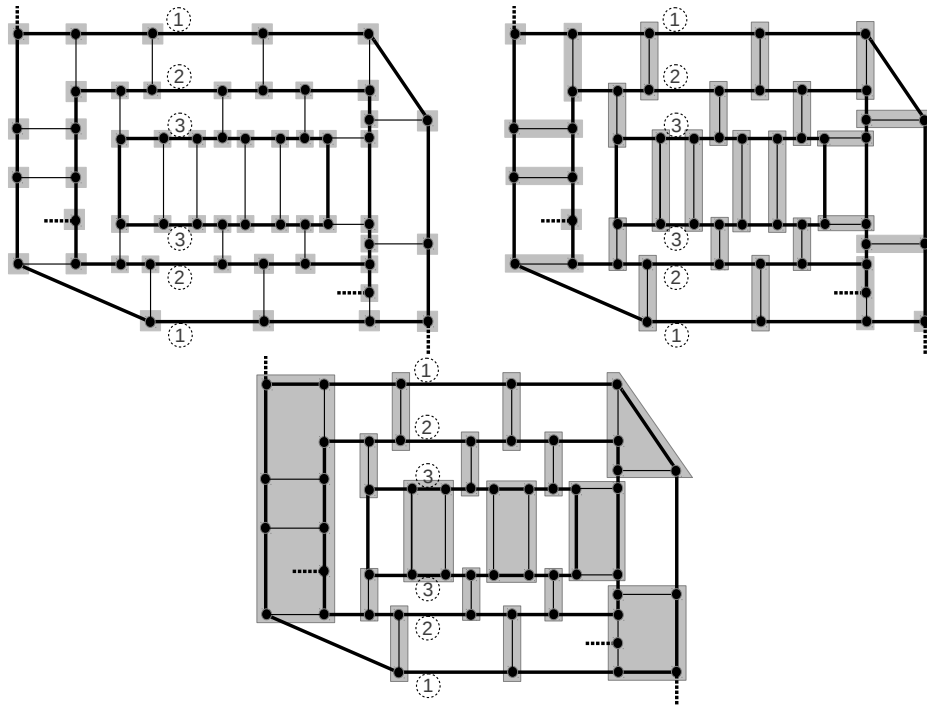


Figure 3.9: Progress of Algorithm 1 on a good 3-outerplanar embedding (same as in Figure 3.7). The shaded areas demarcate the subnetworks already assembled. Left assembly: after **initialization**, Middle assembly: after **first iteration**, Right assembly: after **second iteration**.

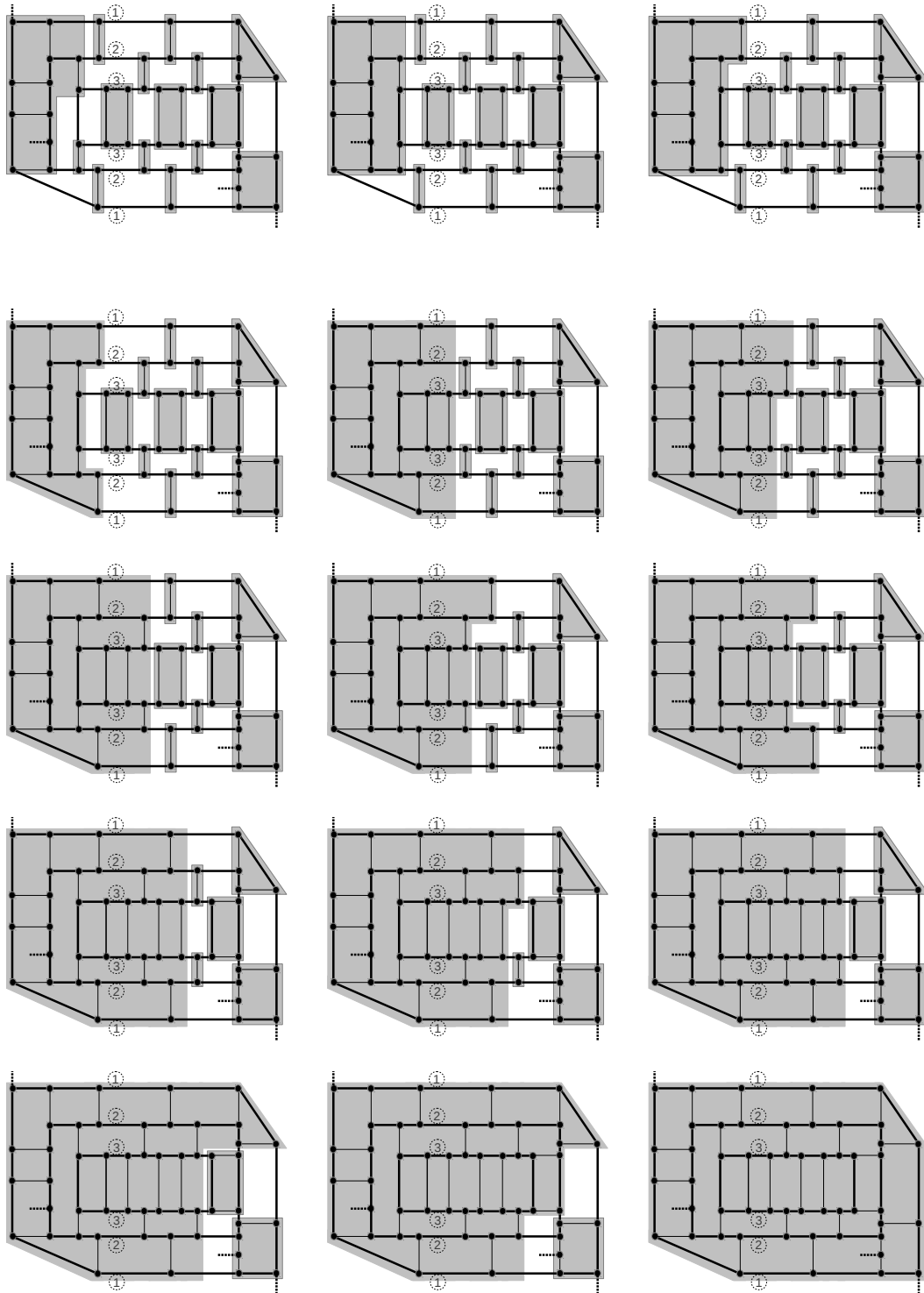


Figure 3.10: Progress of Algorithm 1 on a good 3-outerplanar embedding (same as in Figure 3.7) during the **main iteration**.

Proof 27 (for Lemma 14). We have all the pieces for showing the proof of Lemma 14. To do so, given a planar flow network $\mathcal{N} = (\mathbf{V}, \mathbf{E})$ with a fixed k -outerplanar embedding, we take following steps before applying the approach in Theorem 11 for finding a principal typing for \mathcal{N} :

- Using Lemma 22, in $\mathcal{O}(n)$ steps, we find a good planar embeddings (see definition 20) for \mathcal{N} . This new embedding has the same outerplanarity k .
- Using Lemma 18, in $\mathcal{O}(n)$ steps, we find a linear reassembling sequence Θ for \mathcal{N} where the edge-boundary degree of the main component at every step during reassembling is $\kappa \leq 2(k+1)$.

Using Θ in the approach if Theorem 11 we compute a principal for \mathcal{N} , and in particular the value of the maximum in/out feasible flow in \mathcal{N} .

The proof of Theorem 11 utilizes an induction on the size the main component $\tilde{\mathcal{N}}$ appearing in the reassembling sequence. The time complexity of this approach directly depends on: (1) the time complexity of splicing an edge in every step, *i.e.* the time complexity of the binding procedure and (2) the size of the principal typing of the main component, *i.e.* $|\tilde{T}|$, where \tilde{T} is the principal typing of the main component $\tilde{\mathcal{N}}$. The analysis of the time complexity of the method used in Theorem 11 can be broken down as following:

1. Let $\mathcal{C}(\tilde{P})$ be the set of constraints representing the adequate form of an NNCC P in \tilde{T} . Based on the adequate representation of P , every constraints of $\mathcal{C}(\tilde{P})$ is linear and the coefficient of the variables contributing are in set $\{0, \pm 1\}$. Accordingly, if the edge-boundary degree of the main component at every step is κ , every NNCC of the \tilde{T} is defined by at most 2^κ constraints. In a binding step (splicing e^+ and e^-) assume one of the edges $\{e^+, e^-\}$ belongs to $\tilde{\mathcal{N}}$ and the other is an outer edge of a one-vertex component \mathcal{M} . Also let T be the principal typing for \mathcal{M} . Hence, as explained in the proof of Theorem 11, the time complexity of finding a principal typing for the new main component $\tilde{\mathcal{N}}'$ is $\mathcal{O}(|T| \cdot |\tilde{T}| \cdot \mathcal{P}(2^\kappa))$. The polynomial \mathcal{P} represents the cost of computing the intersection of $e^+ = e^-$ with \tilde{P} and P , for every NNCC \tilde{P} of \tilde{T} and every NNCC P of T . Since (besides other efficient method) invoking Linear

Programming methods is always an option for this purpose, then this cost is polynomial with respect to the size of $\mathcal{C}(\tilde{P})$ and $\mathcal{C}(P)$.

2. The base case of the inductive approach in the proof of Theorem 11 starts with a one-vertex main component $\tilde{\mathcal{N}}_0 = (\{v\}, \tilde{\mathbf{E}}_0)$. It is left to reader and easy to check that $|\tilde{T}_0| \leq 2^{\text{degree}(v)}$, where \tilde{T}_0 is a principal typing for $\tilde{\mathcal{N}}_0$. Now consider an inductive step of the proof of Theorem 11. Let \tilde{T}_i be the principal typing for the main component $\tilde{\mathcal{N}}_i$ and $\tilde{\mathbf{E}}_{i,\text{in},\text{out}}$ be the set of outer edges of $\tilde{\mathcal{N}}_i$. For the induction hypothesis let assume $|\tilde{T}_i| \leq 2^{\tilde{\mathbf{E}}_{i,\text{in},\text{out}}}$. It is easy to check that subcases of Case 4.1.1 are the only cases where after splicing e^+ and e^- , we have $|\tilde{T}_{i+1}| \geq |\tilde{T}_i|$. Precisely, $|\tilde{T}_i| \leq |\tilde{T}_{i+1}| \leq 2|\tilde{T}_i|$. On the other hand, $\tilde{\mathbf{E}}_{i+1,\text{in},\text{out}} = \tilde{\mathbf{E}}_{i,\text{in},\text{out}} + 2$. Hence, after every inductive step, the induction hypothesis assumption $|\tilde{T}_{i+1}| \leq 2^{\tilde{\mathbf{E}}_{i+1,\text{in},\text{out}}}$ holds.

Since based on Θ , for every main component $\tilde{\mathcal{N}}_i$ we have $|\tilde{T}_i| \in \mathcal{O}(2^k)$, and also the fact that for every one-vertex component $|T|$ is a small constant, then the time complexity of every edge splicing is in $\mathcal{O}(2^k \cdot \mathcal{P}(2^k))$ (*i.e.* it is independent of the size of \mathcal{N}).

Finally, since there are exactly m binding steps where each one's time complexity is in $\mathcal{O}(2^k \cdot \mathcal{P}(2^k))$, then the overall cost of finding a principal typing for a k -outerplanar network is in $\mathcal{O}(m \cdot 2^k \cdot \mathcal{P}(2^k)) = \mathcal{O}(n \cdot 2^k \cdot \mathcal{P}(2^k))$.

3.7 Conclusion and Future work

In this chapter we studied the network typings framework as a solution for a compositional, algebraic and polyhedral analysis of a more general class of flow networks, *i.e.* additive flow network. In addition to the presenting necessary notions, definitions and theorems for this framework for additive flow networks, we also formally presented an efficient approach for finding a principal typing for an arbitrary flow network \mathcal{N} when the underlying graph of \mathcal{N} is k -outerplanar (where k is a small and constant integer value relative to the size of \mathcal{N}).

There are possible and natural generalizations of this framework that can be investigated in future. Among such generalizations:

1. Adjust the formal framework to handle the commonly-considered cases of:

- *multicommodity flows* (formal definitions in [2])
- *minimum-cost flows, minimum-cost max flows*, and variations

These cases introduce new kinds of linear constraints, often more general than *flow-conservation* equations and *capacity-constraint* inequalities.

2. Other *network topologies* that are amenable to the kind of examination we already applied to *planar networks*, which can be efficiently analyzed.

Beyond network generalizations and topologies, there are a number of questions more directly related to the fine-tuning of our framework. For instance the existence of customized and efficient algebraic operators for additive flow networks which are the counter parts of efficient operators (e.g. Bind operator) that we studied in [63] for standard networks.

Chapter 4

Max-Flow and Shortest Path Problems in Additive Flow Networks

In previous chapter we studied a compositional framework for analysis of flow networks, with additive gains and losses where to every edge e is assigned a gain factor $g(e)$ which represents the loss or gain of the flow while using edge e . Hence, if a flow $f(e)$ enters the edge e and $f(e)$ is less than the designated capacity of e , then $f(e) + g(e) \geq 0$ units of flow reach the end point of e , provided e is used, *i.e.*, provided $f(e) \neq 0$. In this chapter we study the maximum flow problem in additive flow networks, which we prove to be NP-hard even when the underlying graphs of additive flow networks are planar. Additionally, we also investigate the shortest path problem, when to every edge e is assigned a cost value for every unit flow entering edge e , which we show to be NP-hard in the strong sense even when the additive flow networks are planar.

4.1 Background and Motivation

In standard definition of flow network problems, such as the max-flow problem, it is assumed that if $f(e)$ units of flow enter the edge $e = \langle v, w \rangle$ at its tail v , exactly $f(e)$ units will reach its head u . In practice this assumption in many flow models does not hold. For instance, in the well-known *generalized flow networks*, if $f(e)$ units of flow enter v , and a gain factor $g(e)$ is assigned to e , then $g(e) \times f(e)$ units reach u . Depending on the application, the gain factor can represent the loss or gain due to evaporation, energy dissipation, interest, leakage, toll or *etc.*

The *generalized maximum flow* problem has been widely studied. Similar to the standard max-flow problem, generalized max-flow problem can be formulated as a linear programming,

and therefore it can be polynomially solved using different approaches such as modified simplex method, or the ellipsoid method, or the interior-point methods. Taking advantage of the structure of the problem, different general purpose linear programming algorithms have been tailored to speed up the calculation of max-flow in generalized flow networks [54, 57, 77].

The strong relationship between generalized max-flow problem and *minimum cost flow* problem was first recognized and established by Truemper in [94]. Exploiting this relationship and more importantly the discrete structure of the underlying graph, the generalized max-flow problem can also be solved in polynomial time by *combinatorial methods* [42, 46, 47, 78, 91, 95].

In contrast to the well-studied generalized flow networks, recently in [20], the authors introduced and investigated flow networks where an additive fixed gain factor is assigned to every edge *if used*. Flow networks with additive gains and losses (*additive flow networks* for short) have several applications in practice. In communication networks, a fixed-size load is added to every package being sent out by routing nodes in the network. In transportation of goods or commodities, a fixed amount may be lost in the transportation process or a flat-rate amount of other commodities or cost may be added to the commodity passing every toll station. In financial systems there are fixed costs or losses for every transaction.

The max-flow problem in additive networks can be views as the problem of finding a feasible flow which either maximizes the amount of flows departing the source vertices or maximizes the amount of flow reaching the sink vertices (respectively called *maximum in-flow* and *maximum out-flow* problems). Similarly, assuming that a *unit flow* is departing a source vertex, the shortest path problem is the problem of finding a feasible flow along a path from the source vertex to the sink vertex with minimum accumulated cost.

In Chapter 3 we studied the compositional analysis of additive flow network using so called typing framework. As we saw in this chapter and discussed in [20], flow networks with additive gains and losses, are different in many aspects from standard and generalized flow networks due to some properties such as *flow discontinuity*, lack of max-flow/min-cut *duality*, and unsuitability of *augmented path* methods. Similarly, some basic properties of the shortest path in standard flow network do not hold in the additive case. For instance, it is not anymore the case that the sub-

path, the prefix or the suffix of a shortest path must themselves be shortest. For more details on the properties of additive flow networks, we refer the reader to [20]. These differences make both the max-flow problem and the shortest path problem hard to solve for additive flow networks. In Chapter 3 is shown that an efficient algorithm for finding the value of maximum in/out flow exists, when the underlying graph of the network is k -outerplanar and k is relatively a small integer value. The authors of [20] show that the shortest path problem and the maximum in/out-flow problems are NP-hard for general graphs. In this paper we extend this results to the case where the underlying graph of the flow network is planar.

4.2 Definitions and Preliminaries

In this chapter we augment the definition of additive flow networks by adding flow cost values to every edge. This definition complies with the Definition 1 in Chapter 3 while for simplicity we assume that the flow networks do not have dangling edges and instead there are two designated set of source and sink vertices. Hence, a flow network with additive losses and gains (*additive flow network* for short) is defined by a tuple $\mathcal{N} = (V, E, S, T, U, c, g)$, where $G = (V, E)$ is the directed underlying graph with $n = |V|$ vertices and $m = |E|$ edges. The two sets $S, T \subset V$ are respectively the designated sets of source and sink vertices. $U : E \rightarrow \mathbb{R}^+$ is the edge capacity function, while $U : E \rightarrow \mathbb{R}$ is the cost function and $g : E \rightarrow \mathbb{R}$ assigns a gain or loss value to every edge *if used*. Precisely speaking, the cost $U(c)$ per units of flow and the gain factor $g(e)$ are applied, only if a positive flow enters the edge e .

As in standard flow networks, it is assumed that for every source vertex $s \in S$, $deg^+(s) = 0$ and for every sink vertex $t \in T$, $deg^-(t) = 0$. Flow $f : E \rightarrow \mathbb{R}_+$ in a network \mathcal{N} is feasible if it satisfies edge capacity $0 \leq f(e) \leq U(e)$ for every edge $e \in E$ and flow conservation constraint at every vertex in V . Formally speaking, f satisfies the flow conservation constraints if for every $v \in V - (S \cup T)$:

$$\sum_{e \in \text{in}(v), f(e) > 0} \max(0, f(e) + g(e)) = \sum_{e \in \text{out}(v)} f(e).$$

Given edge $e = \langle v, u \rangle$, if flow $f(e)$ exits vertex v , then $\max(0, f(e) + g(e))$ units reach u . Therefore edge $e = \langle u, v \rangle \in E$ is lossy if the entering edge $f(e)$ is positive and $g(e) < 0$. Edge e consumes the entering flow if $f(e) + g(e) < 0$, in which case no flow reaches the vertex v .

Definition 28 (*Out-flow and in-flow*). Given flow f for network \mathcal{N} , the *out-flow* is the summation of the amount of flow exiting source vertices, i.e., $f_{out} = \sum_{s \in S, e \in \text{out}(s)} f(e)$. Similarly, *in-flow* is the summation of flow values entering all sink vertices, namely $f_{in} = \sum_{t \in T, e \in \text{in}(t), f(e) > 0, \max(0, f(e) + g(e))}$. \square

In additive flow networks, the max-flow problem can be studied from the producers' (source vertices) point of view or from the consumers' (sink vertices) point of view. Hence, the maximum out-flow problem is the problem of finding a feasible flow f maximizing f_{out} , and the maximum in-flow problem is defined similarly. Regarding these two problems, while trying to maximize the amount of outgoing/incoming flows, we are not concerned with the cost of flow.

On the other hand, in additive flow networks, the shortest path problem can be generalized in several ways. For instance, given a producer vertex $s \in S$ (similarly consumer vertex $t \in T$), the problem can be defined as finding a consumer vertex $t \in T$ (producer vertex $s \in S$) with the shortest distance among the others, with respect to the cost of a unit flow departing s towards t . A more generalized variation of the problem can be defined, where neither of the source or destination vertices are fixed. Therefore, the generalized shortest path is the problem of finding a source vertex $s \in S$, a destination vertex $t \in T$, and a path Π from s to t with minimum cost, if a unit of flow departs s . Without loss of generality, in the rest of this chapter, it is assumed that the sets of source and sink vertices each has one member (namely $|S| = |T| = 1$). Hence, every negative result shown for this special case is immediately applicable to the generalized version of the shortest path problem. Section 4.3 concerns the shortest path problem and the related definitions with more details.

Definition 29 (*Cost of flow*). Given a flow function f in network \mathcal{N} , the *cost of flow* on every edge e is $f(e) \times U(e)$. Similarly the *accumulated cost* of f is the summation of cost of flow entering all edges, i.e., $cost(f) = \sum_{e \in E} f(e) \times U(e)$. \square

Example 30. In Figure 4.1 an additive flow network and a feasible flow from vertex s to vertex t are depicted. The cost of unit flow for every edge is 1 and the capacity of every edge is $B + 1$ for $B > 1$. For a compact illustration of figures in this chapter, we adopt the following conventions:

- If we label an edge e with $g = r$ or $c = r$ or $u = r$, for some $r \in \mathbb{R}$, then we mean that $g(e) = r$ or $U(c) = r$ or $U(e) = r$, respectively.
- If we omit such a label on an edge e , then we mean that the value of the corresponding function for e is the default value (as stated in the description of that figure).

For instance in Figure 4.1, $g(\langle s, v_1 \rangle) = B + 1$ for edge $\langle s, v_1 \rangle$, is represented by $g = B + 1$ by that edge. The missing gain values are the default value 0. In this example, the initial unit flow leaves vertex s while $B + 2$ units reach v_1 , due to gain value $B + 1$ assigned to edge $\langle s, v_1 \rangle$. The flow entering edge v_1, v_5 is fully absorbed by the gain factor $-B$ assigned to it. Hence, the accumulated cost of the flow along path¹ $\Pi = (s, v_1, v_2, v_3, t)$ is $B + 4$ while the accumulated cost of the flow f is $B + 5$.

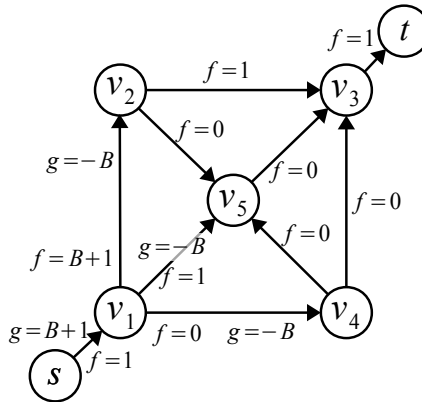


Figure 4.1: An example of an additive flow network \mathcal{N} and the flow function f assigning feasible flow values to every edge of \mathcal{N} . Missing gain values are 0, the cost function c is 1 on every edge, and the capacity function u is a "large number" on every edge (for example $B + 1$).

□

¹A simple path can be interchangeably represented both by the set of vertices or by the set of edges taking part in it.

4.3 Shortest Path Problem in Additive Flow Network

Let \mathcal{N} be an additive flow network and let $\Pi = (\langle s, v_1 \rangle, \langle v_1, v_2 \rangle, \dots, \langle v_{k-1}, t \rangle)$ be a (simple) path in \mathcal{N} from vertex s to t . Let $e_i = \langle v_{i-1}, v_i \rangle$ for $2 \leq i \leq k-1$ while $e_1 = \langle s, v_1 \rangle$ and $e_k = \langle v_{k-1}, t \rangle$. Given the flow f along path Π with the initial flow f_1 entering e_1 (*seed flow* for short), the accumulated flow entering edge e_i for every $1 \leq i \leq k$, is represented by:

$$\gamma(\Pi, f_1, i) = f_1 + \sum_{j < i} g(e_j).$$

Definition 31 (*Feasible and dead-end flows along a path*). Flow f with the seed flow f_1 , is *feasible* along path $\Pi = (e_1, \dots, e_k)$ if the flow on every edge is positive and a positive amount of flow reaches the destination t . *i.e.* for $1 \leq i \leq k+1$:

$$\gamma(\Pi, f_1, i) > 0.$$

Flow f is *infeasible* or *dead-end* if an edge e_i for $1 \leq i \leq k$ absorbs all the entering flow, due to the loss value $g(e_i)$ assigned to that edge. Hence, no flow reaches the destination vertex. \square

For every vertex t reachable from s via some path Π , there exists a threshold value Υ such that a flow along Π is feasible only if its seed flow $f_1 > \Upsilon$. Finding the reachability threshold (if exists) for every pair of source and sink vertices is a polynomial task, as formally stated in Lemma 32.

Lemma 32. *Consider a flow network $\mathcal{N} = (V, E, \{s\}, \{t\}, U, c, g)$. There exists a polynomial time algorithm that decides the reachability of t from s in network \mathcal{N} and finds the reachability threshold in $\mathcal{O}(nm)$, if such threshold exists.*

Proof. In order to find the threshold value Υ in \mathcal{N} such that at least one path from source s to destination t is feasible (*i.e.*, t is reachable from s for seed flow $f_1 > \Upsilon$), one can use a variation of well-known shortest path algorithms (such as Bellman-Ford algorithm) in the reversed graph of \mathcal{N} . The reversed graph is constructed from the underlying graph of \mathcal{N} by reversing the direction of every edge e and assigning $-g(e)$ as the weight of the reversed edge. Hence, the reachability

threshold problem reduces to a modified variation of the shortest path problem from t to s . This problem is a modified variation of the standard shortest path problem with negative costs, in the sense that the distance of no two vertices can be negative. Hence, in the modified variation of the Bellman-Ford algorithm in the process of updating the distance matrix for every vertex v from s , the distance of v from s is set to $\max\{0, d\}$, where d is the newly updated distance of v from s . Note that due to flow conservation constraints at every vertex, there is no feasible flow with positive gain cycles involved in it. Also in this modified variation of Bellman-Ford algorithm the distance of no two vertices can be negative. Assuming that there is no positive gain cycle in \mathcal{N} , means that there is no negative cycle in the reversed graph, which in turn implies that the modified variation of Bellman-Ford algorithm always returns the threshold, if t is reachable from s . The correctness proof of this approach is straightforward and similar to the proof of the correctness of the standard Bellman-Ford algorithm for the shortest path problem. The time complexity of this algorithm is the same as the standard Bellman-Ford algorithm, which has $\mathcal{O}(|V| \times |E|)$ worst case time bound. \square

The accumulated cost of a flow along the path $\Pi = (e_1, \dots, e_k)$ (feasible or not), is the summation of the cost of the flows entering every edge times the value of cost function assigned to that edge, namely:

$$\sum_{1 \leq i \leq k} U(e_i) \gamma(\Pi, f_1, i).$$

In [20] regarding the shortest path problem, the authors simplify the problem by assuming that the seed value $f_1 = 1$. On the other hand, given the source and destination vertices s and t in flow network \mathcal{N} , it may be the case that for every path Π from s to t , no feasible flow along Π with the seed value $f_1 = 1$ exists. Accordingly the definition of shortest path problem in [20] can be generalized as in following.

Definition 33 (*Shortest path in additive flow networks*). The *shortest path* problem in additive flow network \mathcal{N} for a given pair of source and sink vertices $\{s, t\}$, is the problem of finding a min-cost feasible flow along some path Π from s to t , when the seed flow $f_1 = \min\{1, \Upsilon\}$. Where Υ is the reachability threshold for the source/destination pair $\{s, t\}$. \square

In contrast to the shortest path problem in standard flow networks, this problem is hard in the case of additive flow networks. From Theorem 1 in [20] it can be inferred that when the underlying graph of the flow network is planar, this problem is weakly NP-hard. In other words, the problem may be polynomially solvable if the cost and capacity values assigned to the edges are bounded from above by some polynomial function in the size of the graph. In the same paper it is shown that this result holds if the cost and gain values are all *nonnegative integers*, even for the case where the underlying graph is not necessary planar.

In this section we show that the shortest path problem is NP-hard *in the strong sense* when the underlying graph of the additive flow network is planar (*planar additive flow network* for short). We show this result using a polynomial reduction from a problem called Path Avoiding Forbidden Transitions (PAFT for short) [56]. PAFT is a special case of the problem of finding a path from source vertex s to destination vertex t while avoiding a set of forbidden paths (initially introduced in [96]). Before presenting the main result of this section (as stated in Theorem 39), we briefly define PAFT and some results on this problem (for more details, the reader is referred to [56]).

Given undirected multi-graph $G = (V, E)$, a *transition* in G is an unordered set of two distinct edges of E which are incident to the same vertex of V . If \mathcal{T} denotes the set of all possible transitions in the graph G , the set \mathcal{F} of *forbidden transitions* is a subset of \mathcal{T} . Then $\mathcal{A} = \mathcal{T} - \mathcal{F}$ denotes the set of *allowed transitions*. A simple path $\Pi = (e_1, e_2, \dots, e_k)$, where $e_1 = \{s, v_1\}$ and $e_k = \{v_{k-1}, t\}$, is *\mathcal{F} -valid* if for every $1 \leq i < k$, $\{e_i, e_{i+1}\} \notin \mathcal{F}$; namely, no transition in Π is forbidden. A vertex v is *involved* in a forbidden transition $\{e, e'\}$ if two edges e and e' share v and $\{e, e'\} \in \mathcal{F}$.

Definition 34 (PAFT). Consider a multi-graph $G = (V, E)$, a set of forbidden transitions \mathcal{F} and designated source and destination vertices $s, t \in V$. PAFT is the problem of find an \mathcal{F} -valid path from s to t , if exists. □

Lemma 35. *PAFT is NP-complete for planar graphs where the degree of every vertex $v \in V - \{s, t\}$ is 3 or 4, where s and t are the source and destination vertices, respectively.*

Proof. In [56], the authors show that PAFT is NP-complete in planar graphs where the degree of every vertex is at most 4. Consider graph $G = (V, E)$ with maximum vertex degree 4 and source

and destination vertices s and t , and let \mathcal{F} be the set of forbidden transitions. Graph $G' = (V', E')$ and the set of forbidden transitions \mathcal{F}' are constructed as explained in what follows.

Initially $V' = V$, $E' = E$ and $\mathcal{T}' = \mathcal{T}$. Until there is no vertex of degree 1 or 2, for every vertex $v \in V' - \{s, t\}$:

1. **If $\text{degree}(v) = 1$:** (i) $V' = V' - v$, (ii) $E' = E' - e$, where e is the edge incident to v , and (iii) $\mathcal{T}' = \mathcal{T}' - \tau$, for every forbidden transition τ that v is involved in.
2. **If $\text{degree}(v) = 2$ and $\tau = \{e_1, e_2\} \in \mathcal{F}$, where e_1 and e_2 share v :** (i) $V' = V' - v$, (ii) $E' = E' - \{e_1, e_2\}$, and (iii) $\mathcal{T}' = \mathcal{T}' - \tau$
3. **If $\text{degree}(v) = 2$ and $\tau = \{e_1, e_2\} \notin \mathcal{F}$, where e_1 and e_2 share v :** (i) v is smoothed out by removing v and the two incident edge $e_1 = \{w, v\}$ and $e_2 = \{v, u\}$ and introducing a new edge $e' = \{w, u\}$. (ii) For $i \in \{1, 2\}$ if $\{e_i, e\} \in \mathcal{F}$ for some $e \in E$; $F' = F' \uplus \{\{e', e\}\} - \{e_i, e\}$.

It is straightforward and left to reader to verify that there is \mathcal{F} -valid path from s to t in G iff there is a \mathcal{F}' -valid path from s to t in G' . Accordingly, the NP-hardness of PAFT for planar graphs with maximum degree 4 results in the NP-hardness of PAFT for the class of planar graphs where the degree of every vertex is 3 or 4. In [56], using a similar approach, this result is extended to grid graphs. \square

This lemma helps us to draw the main result of this section (as stated at the end of this section in Theorem 39). Before that, in an intermediate step, Procedure 36 represents an approach to transforming an instance of PAFT into an additive flow network. Without loss of generality, we assume that the source and the sink vertices are not involved in any forbidden transition².

Procedure 36 (*PAFT's instance into additive flow network*). Consider a planar undirected multi-graph³ $G = (V, E)$, a set of forbidden transitions \mathcal{F} , and source and destination vertices s and t as an instance of PAFT, where for every vertex $v \in V$, $\text{degree}(v) \in \{3, 4\}$. We transform such instance of PAFT into an additive flow network \mathcal{N} in several steps:

²If the source vertex (or sink vertex) is involved in any forbidden transition, a new source vertex (or sink vertex) is introduced and is connected to the old one.

³Assume that the planar embedding is given.

1. Every undirected edge $e = \{v, u\}$ is replaced by a pair of parallel incoming/outgoing directed edges $e = \langle v, u \rangle$ and $e' = \langle u, v \rangle$.
2. Two new vertices s' and t' are introduced. s' is connected to s via edge $e_s = \langle s', s \rangle$ and t is connected to t' via $e_t = \langle t, t' \rangle$. e_t has 0 gain while e_s has gain value B assigned to it for some $B > 1$. To both edges e_s and e_t is assigned cost value 0.
3. For every vertex v , not involved in any forbidden transition, every outgoing edge $\langle v, v' \rangle$ is subdivided into two edges $\langle v, w \rangle$ and $\langle w, v' \rangle$ by introducing a new vertex w . Two edges $\langle v, w \rangle$ and $\langle w, v' \rangle$ respectively have gain value $-B$ and $+B$ and cost $c = +B$ and $c = -B$. Figure 4.2 represents the replacement gadget for a vertex v with degree 4.

The gain value $-B$ assigned to every outgoing edge $\langle v, w \rangle$ guarantees that if $B + 1$ units of flow enter the gadget of v , the exiting flow reaches the gadget of at most one of the neighbors of v in G . The gain value $+B$ assigned to the edge $\langle w, v' \rangle$ (connected to outgoing edge $\langle v, w \rangle$) compensates for the lost flow that enters $\langle v, w \rangle$, if some flow reach w . Hence, $B + 1$ units of flow entering the gadget of v reach the gadget of exactly one of the neighbors of v in G *with no loss*, if only one of the outgoing edges is chosen.

Moreover based on the same reasoning, the cost values in this gadget assure us that the $B + 1$ units of flow entering this gadget reaches the gadget of the neighboring vertex *with no cost*, if only one of the outgoing edges is chosen.

In summary, if $B + 1$ units of flow enter such gadget of a vertex v , in order to have some flow reaching the neighboring gadget, one of the outgoing edges must have $B + 1 - x$ units entering flow for $0 \leq x < 1$. In this case $B + 1 - x$ units reach the neighboring gadget and the x units of flow is consumed with total cost xB .

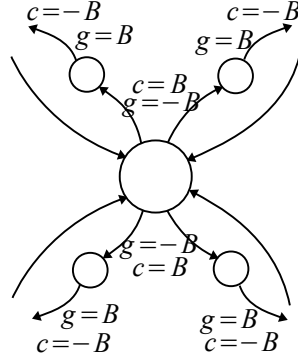


Figure 4.2: The gadget that replaces a vertex of degree 4 which is not involved in any forbidden transition.

4. Consider vertex v involved in some forbidden transitions with degree $d = \text{degree}(v) \leq 4$ incident to d pairs of incoming/outgoing parallel edges $\{e_1, e'_1\}, \dots, \{e_d, e'_d\}$. Originally in graph G vertex v is incident to e_1, \dots, e_d . The planar embedding of v in G is represented in Figure 4.3 on the left, when $d = 4$.
 - (a) **If $d = 3$:** vertex v is replaced by 3 new vertices v_1, v_2, v_3 where v_i is incident to the pair $\{e_i, e'_i\}$ for $1 \leq i \leq 3$, based on the planar embedding of edges e_1, e_2, e_3 in G . For $1 \leq i, j \leq 3$, two vertices v_i and v_j are directly connected with a pair of parallel incoming/outgoing edges, if $\{e_i, e_j\} \notin \mathcal{F}$. The cost of every edge is 0 and the gain factor for every introduced edge $\langle v_i, v_j \rangle$ is $-B$ for $1 \leq i, j \leq 3$. Finally, every outgoing edge connecting v_i (for $1 \leq i \leq 3$) to another gadget (corresponding to one of the neighbors of v in G) has cost 0 and gain factor $+B$.
 - (b) **If $d = 4$ and $\{\{e_1, e_4\}, \{e_2, e_3\}\} \notin \mathcal{A}$:** A gadget of four vertices replaces v , following the same procedure as explained in the previous case. Figure 4.3 depicts an example of transforming a vertex of degree 4 involved in some forbidden transitions.
 - (c) **If $d = 4$ and $\{\{e_1, e_4\}, \{e_2, e_3\}\} \subseteq \mathcal{A}$:** Using the same approach as in the previous case spoils the planarity of the resulting network \mathcal{N} . To solve this problem, vertex v is replaced by a gadget of 5 vertices. In addition to 4 vertices v_1, \dots, v_4 , where for $1 \leq i \leq 4$, v_i is connected to the pair of incoming/outgoing edges $\{e_i, e'_i\}$, a central vertex w is introduced as well. As depicted in Figure 4.4, in the replacing gadget,

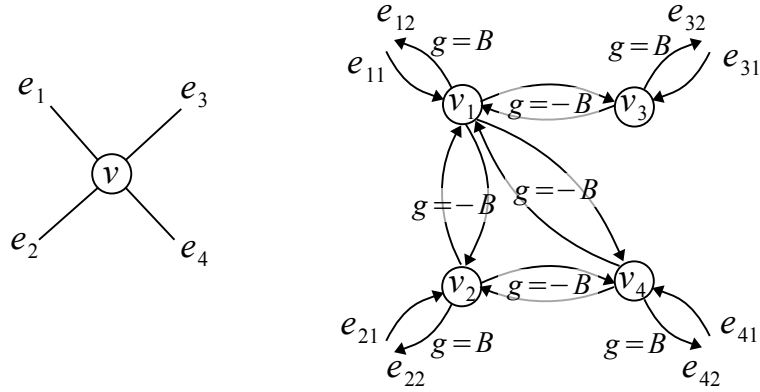


Figure 4.3: On the left is the planar embedding of a vertex $v \in V$ incident to 4 edges where $\{e_2, e_3\}, \{e_3, e_4\} \in \mathcal{F}$. The right image shows the gadget replacing vertex v . Every edge connecting two vertices v_i and v_j for $1 \leq i, j \leq 4$ has cost value 0 and gain value $-B$ assigned to it. Vertices v_2 and v_3 are not directly connected since their corresponding edges e_2 and e_3 are involved in a forbidden transition. Hence, a flow from v_2 to v_3 (and vice versa) loses $-2B$ units. Same situation holds for v_3 and v_4 .

vertices v_1 and v_4 (also vertices v_2 and v_3) are connected via the central vertex w . Any other two vertices v_i and v_j for $1 \leq i, j \leq 4$ are directly connected via a pair of parallel incoming/outgoing edges, if $\{e_i, e_j\} \notin \mathcal{F}$ (with gain and cost values $-B$ and 0 respectively). The cost and gain factors for every edge can be found by that edge and the missing values are the default value 0.

Assume $B + 1$ units of flow reach any of the four vertices v_1, \dots, v_4 . The gain and cost values assigned to the edges incident to w guarantee that a flow can go through the central vertex w with no cost and reach the destination with $-B$ units loss, only if it is from v_1 to v_4 and vice versa or if it is from v_2 to v_3 and vice versa. Any other flow, with the initial value $B + 1$ units, that uses w is either costly (costs $2B(B + 1)$) or gets fully consumed (*i.e.*, does not reach the destination).

5. Every edge has capacity $B + 1$.

Note that if the undirected multi-graph G is planar, then so is the underlying graph of the flow network \mathcal{N} as a result of the preceding transformation. Also since the capacity of every edge is $B + 1$, it is guaranteed that maximum amount of flow that enters a gadget is $B + 1$ units and no more than $B + 1$ units of flow departs any gadget. \square

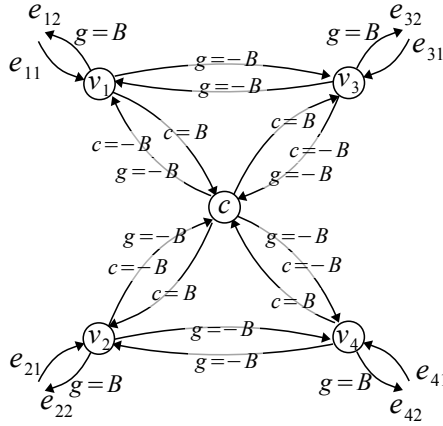


Figure 4.4: The gadget replacing vertex v where $\text{degree}(v) = 4$ and $\{e_1, e_4\}, \{e_2, e_3\} \in \mathcal{A}$. Edges e_1, \dots, e_4 are the edges connected to v as represented in a planar embedding of G in Figure 4.3 (on the left). This gadget replaces v where $\{e_1, e_2\}, \{e_3, e_4\} \in \mathcal{F}$.

Example 37. Figure 4.5a denotes a graph G and its designated source and destination vertices. The set of forbidden transitions is $\mathcal{F} = \{\{e_2, e_3\}, \{e_3, e_4\}\}$. Figure 4.5b shows the additive flow network \mathcal{N} constructed based on G and the set of forbidden transitions. The set of four vertices shown in the dashed circle represents the gadget replacing vertex v which is the only vertex involved in the forbidden transitions. The missing gains of every edge in this gadget is $-B$.

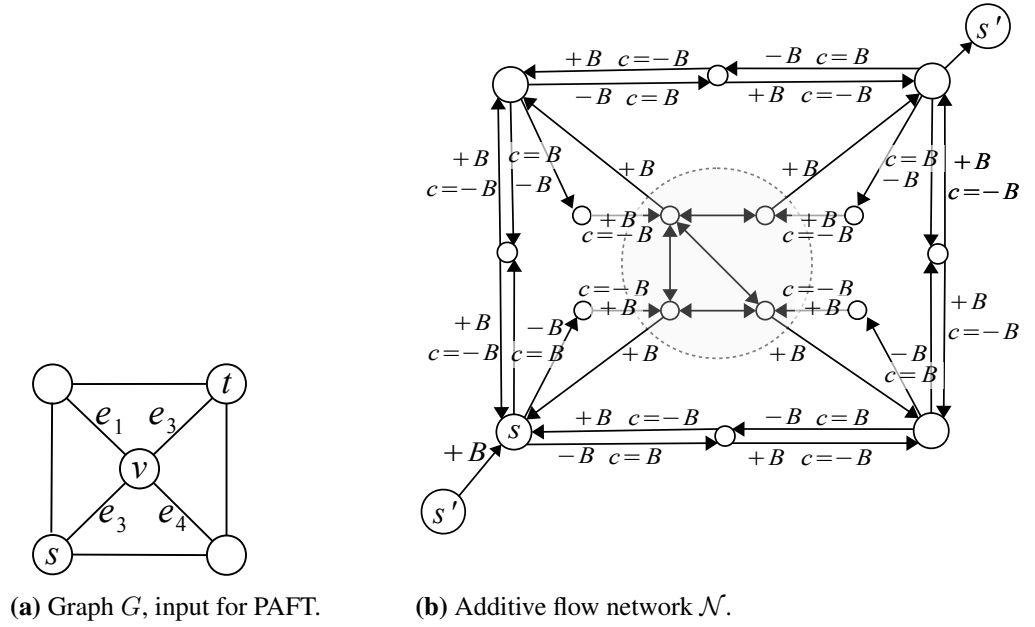


Figure 4.5: Additive flow network \mathcal{N} in 4.5b, constructed based on the graph G in 4.5a and the set of forbidden transitions $\mathcal{F} = \{\{e_2, e_3\}, \{e_3, e_4\}\}$. The set of four vertices shown in the dashed circle represent the gadget replacing middle vertex v . The missing gain values for every edge in this gadget are $-B$. The gain of every other edge is shown by $+B$ or $-B$ by that edge. The default values for missing cost and capacity functions are respectively 0 and $B + 1$.

□

Lemma 38. Consider an undirected multi-graph G , a set of forbidden transitions \mathcal{F} , and source and destination vertices s and t as an instance of PAFT, where the degree of every vertex $v \in V$ is 3 or 4. Let \mathcal{N} be the additive flow network constructed from G and the forbidden transitions set \mathcal{F} using Procedure 36. There is a (simple) \mathcal{F} -valid path from s to t iff there exists a feasible flow in \mathcal{N} along a (simple) path from s' to t' with no cost, when seed flow $f_1 = 1$.

Proof. (\Rightarrow) Assume there is a \mathcal{F} -valid (simple) path $\Pi = (s, v_1, \dots, v_k, t)$ in G . Based on Π we suggest a flow in \mathcal{N} where a unit of flow departing s' reaches the gadget of $v_1 = s$ while gaining B units of flow. The accumulated cost so far is 0. Based on the construction of every gadget, following the path Π , the $B + 1$ units of flow can go through the corresponding gadget of every vertex v_i and reach the gadget of t with no loss and no cost. Therefore, there is a feasible flow in \mathcal{N} from s' to t' with 0 cost.

(\Leftarrow) Let f be a feasible flow in \mathcal{N} along a (simple) path $\Pi = (s', s, \dots, t, t')$ with no cost, where

seed flow $f_1 = 1$. Based on the construction of \mathcal{N} from G , in a coarser view, a feasible flow goes from one gadget to another gadget. Hence, path Π can be viewed as $\Pi = (s', \underline{s}, \underline{v}_1, \dots, \underline{v}_k, \underline{t}, t')$, where \underline{v}_i represent the gadget corresponding to vertex v_i that some of its edges are used in path Π .

The unit flow departs s' and $B + 1$ units reach the gadget \underline{s} with cost 0. When $B + 1$ units reach the gadget \underline{v} of v :

- **If v is not involved in any forbidden transition:** As explained in the third step of Procedure 36, $B + 1 - x$ units reach the gadget of one of the neighbors of v with cost xB for $0 \leq x < 1$.
- **If v is involved in some forbidden transitions:**
 - If v is an instance of 4-(a) or 4-(b) in Procedure 36, flow can reach the gadget of exactly one of neighbors of v only if no forbidden transition is used.
 - If v is an instance of 4-(c) in Procedure 36, as explained in this case, the flow is either fully consumed or suffers cost $2B(B + 1)$, if any forbidden transition is used. The entering flow to this gadget reaches the neighboring gadget with no loss and no cost, only if no forbidden transition is used by the flow.
 - For every gadget involved in some forbidden transitions, it is always the case that the entering $B + 1$ units of flow either is fully consumed or suffers no loss upon reaching the neighboring gadget.

Every feasible flow f originated from s' with seed flow $f_1 = 1$ as reaches t' has gained $B - x$ units, with accumulated cost xB for $0 \leq x < 1$, if no forbidden transition is used. On the other hand, if any forbidden transition is used, a feasible flow along a path from s' to t' costs at least $2B(B + 1)$, as explained in the sub-cases of case 4 in Procedure 36. Accordingly, a feasible flow with $f_1 = 1$ along some path from s' to t' has minimum cost 0, only if no forbidden transition is used and $x = 0$ (*i.e.*, $B + 1$ units reach t' with no loss). \square

The following theorem concludes this section.

Theorem 39. *The simple shortest path problem is NP-hard in the strong sense for additive flow networks where the underlying graph is planar.*

Proof. Proof is immediate based on Lemmas 35 and 38 and the fact that the procedure 36 can be carried out in polynomial time (with respect to the size of the input graph G and the set of forbidden transitions \mathcal{F}). \square

4.4 Maximum Flow Problem in Additive Flow Network

In [20], the authors study the problem of maximum flow with additive gains and losses for general graphs. They show that finding maximum in-flow and out-flow are NP-hard tasks for general graphs. In this section we show the same result for planar additive flow networks. In order to show this result, we facilitate a special variation of planar satisfiability problem (planar SAT), which is briefly defined in the following.

Definition 40 (*Strongly planar CNF*). Conjunctive normal form (CNF for short) formula $\varphi = (\mathcal{X}, \mathcal{C})$ with the set of variable \mathcal{X} and the set of clauses \mathcal{C} is *strongly planar* if graph $G_\varphi = (V, E)$ constructed as follows is planar:

1. V contains a vertex for every literal and one vertex for every clause.
2. $\{x, \tilde{x}\} \in E$ for every $x \in \mathcal{X}$, where \tilde{x} denotes the negation of boolean variable x .
3. If a clause C contains literal \bar{x} (which can be x or \tilde{x}), there is an edge $\{\bar{x}, C\}$ connecting vertex \bar{x} and the vertex corresponding to C .
4. No other edge exists other than those introduced in Part 2 and Part 3.

\square

Example 41. Consider CNF formula $\varphi = (\tilde{x} \vee y \vee \tilde{z}) \wedge (x \vee \tilde{y} \vee z) \wedge (x \vee w \vee z) \wedge (\tilde{x} \vee \tilde{w} \vee \tilde{z})$. Figure 4.6 represents a planar embedding of the graph G_φ . \square

Definition 42 (*1-in-3SAT*). Given 3CNF formula φ , 1-in-3SAT is the problem of finding a satisfying assignment such that in each clause, exactly one of the three literals is assigned to 1. \square

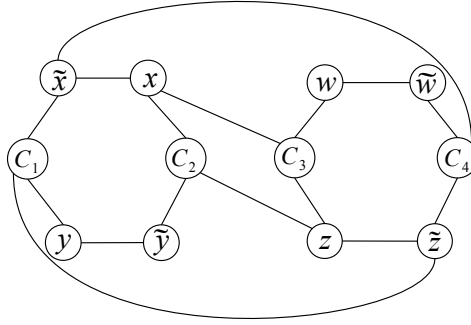


Figure 4.6: Underlying graph of the 3CNF formula $\varphi = (\tilde{x} \vee y \vee \tilde{z}) \wedge (x \vee \tilde{y} \vee z) \wedge (x \vee w \vee z) \wedge (\tilde{x} \vee \tilde{w} \vee \tilde{z})$.

Lemma 43. *Strongly Planar 1-in-3SAT is NP-complete.*

For the proof of Lemma 43, we refer the reader to [98]. We use Lemma 43 to present the main contribution of this section (stated in Theorems 47 and 49). Initially, Procedure 44 shows three steps in order to transform a CNF formula φ into an additive flow network \mathcal{N}_φ .

Procedure 44 (*CNF into additive flow network*). Consider graph $G_\varphi = (V, E)$ corresponding to a 3CNF φ as an instance of 1-in-3SAT. Starting from the undirected graph G_φ , additive flow network \mathcal{N}_φ is constructed using the following steps:

1. Every undirected edge $\{\bar{x}, C\}$ connecting literal \bar{x} (which can be x or \tilde{x}) to clause C is replaced by a directed edge $e = \langle x, C \rangle$ from x to C where $U(e) = 1$ and $g(e) = 0$.
2. For every clause-vertex C_i , a sink vertex t_i and an edge $e = \langle C_i, t_i \rangle$ are introduced, where $U(e) = 1$ and $g(e) = 0$.
3. Every pair of literal-vertices $\{x, \tilde{x}\}$ is replaced by a gadget as shown in Figure 4.7.

□

Example 45. The flow network corresponding to the CNF formula φ of example 41 is depicted in figure 4.8. The missing capacity values and gain factors are 1 and 0, respectively. For simplicity some source vertices (also some sink vertices) are combined. It is easy to see that the graph of the constructed network \mathcal{N}_φ is planar iff G_φ is planar.

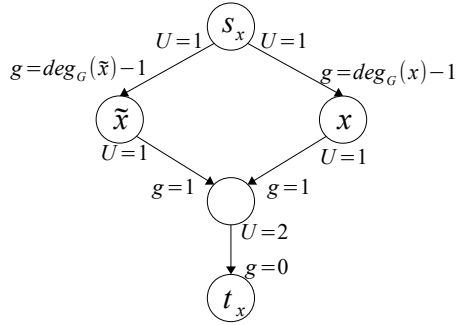


Figure 4.7: The gadget that replaces every pair of literal-vertices $\{x, \tilde{x}\}$. The upper bound and the gain or loss of every edge can be found by that edge, where $deg_G(x)$ is the degree of literal-vertex x in the graph G_φ . The gadget of every pair $\{x, \tilde{x}\}$ introduces one source s_x and one sink vertex t_x .

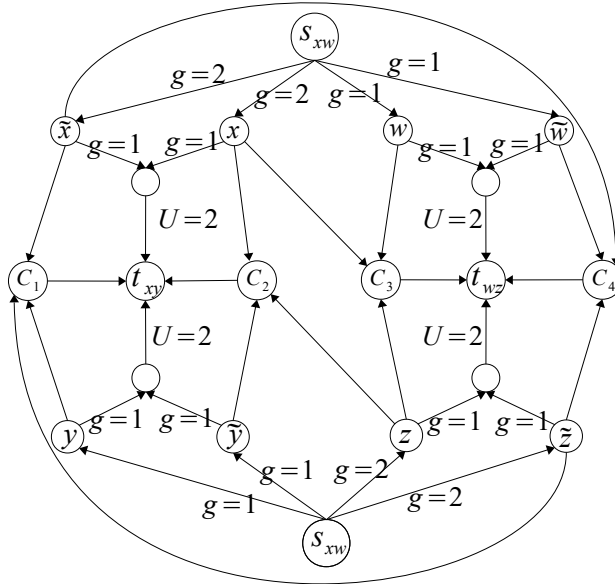


Figure 4.8: The flow network constructed from the graph G_φ of CNF formula $\varphi = (\tilde{x} \vee y \vee \tilde{z}) \wedge (x \vee \tilde{y} \vee z) \wedge (x \vee w \vee z) \wedge (\tilde{x} \vee \tilde{w} \vee \tilde{z})$.

□

Lemma 46. *CNF formula $\varphi = (\mathcal{X}, \mathcal{C})$ is a positive instance of strongly planar 1-in-3SAT iff the maximum in-flow in additive flow network \mathcal{N}_φ is $2|\mathcal{X}| + |\mathcal{C}|$, where \mathcal{N}_φ is constructed according to procedure 44.*

Proof. (\Rightarrow) Consider a satisfying assignment of variables that every clause has exactly one literal with value 1. For every literal \bar{x} assigned to 1 we push 1 unit flow through the edge connecting a

source vertex to the related literal-vertex of \bar{x} . The amount of flow that reaches \bar{x} is exactly one unit more than the number of clauses containing \bar{x} . This flow saturates all the edge connecting \bar{x} to the vertices of the clauses that contain the literal \bar{x} . The remaining one unit flow reaches the sink vertex in the gadget containing the vertex of \bar{x} (plus an extra unit flow gained). Every edge connecting a source to the vertices corresponding to the literals that are assigned 0, will not be used. It is easy to check that the suggested flow is feasible and all the edges connected to sink vertices are saturated.

(\Leftarrow) We produce a variable assignment for φ based on a given feasible f for \mathcal{N}_φ . According to the construction of the gadgets for every pair of literals $\{x, \tilde{x}\}$ in Figure 4.7, in every feasible flow at most one of the two edges $\langle s_x, x \rangle$ and $\langle s_x, \tilde{x} \rangle$ can be used (*i.e.* in every feasible flow f , for every $x \in X$, $f(x) = 0$ or $f(\tilde{x}) = 0$). A literal \bar{x} is set to 1 if $f(\langle s_x, \bar{x} \rangle) > 0$, and is set to 0 otherwise.

Hence, given \mathcal{N}_φ where $f_{in} = 2|\mathcal{X}| + |\mathcal{C}|$, all the edges reaching a sink vertex are saturated. Namely: (i) every edge connecting a clause-vertex to a sink vertex is saturated, which make up $|\mathcal{C}|$ units of flow (*i.e.* corresponding clause is satisfied) and (ii) the remaining $2|\mathcal{X}|$ units of flow is supplied by the sink vertices of all the gadgets (*i.e.* $f(\langle s_x, x \rangle) = 1$ or $f(\langle s_x, \tilde{x} \rangle) = 1$ for every pair $\{x, \tilde{x}\}$). Accordingly, if a feasible flow has maximum in-flow $2|\mathcal{X}| + |\mathcal{C}|$, it is the case that for every variable x only one of the literal-vertices x or \tilde{x} has entering flow, hence in the suggested assignment that literal is set to 1. Also for every clause C_i one unit flow reaches its corresponding vertex, which means that clause is satisfied and only one of its literal is set to 1. \square

It is not hard to check that Procedure 44 can be done in polynomial time in the size of the input CNF formula. Hence, based on Lemmas 43 and 46, the following theorem can be deduced immediately.

Theorem 47. *computing a feasible flow f with maximum in-flow f_{in} is an NP-hard problem in the strong for the class planar additive flow networks.*

In the context of max-flow problem,⁴ for every additive flow network $\mathcal{N} = (V, E, S, T, U, g)$ there exists a reversed flow network $\mathcal{N}' = (V', E', S', T', U', g')$, constructed by reversing the direction of every edge and swapping source vertices with sink vertices (*i.e.* $S' = T$ and $T' = S$).

⁴Note that in this context cost functions are irrelevant.

Given edge $e = \langle v, u \rangle$ in E , we have $e' = \langle u, v \rangle \in E'$ where $u'(e') = U(e) + g(e)$ and $g(e') = -g(e)$. Hence, if edge e is gainy (lossy) in \mathcal{N} , e' is lossy (gainy) in \mathcal{N}' .

Based on the definition of reversed flow networks, the following lemma is straightforward and the details can be found in [20].

Lemma 48. *Consider the feasible flow f for an additive flow network \mathcal{N} , where for every edge e , $g(e) \geq 0$ (in other words, there is no lossy edge in \mathcal{N}). Then exists a flow f' for the reversed network \mathcal{N}' , where $f'_{out} = f_{in}$ (and similarly $f'_{in} = f_{out}$).*

In the construction of \mathcal{N}_φ from G_φ based on Procedure 44, there is no lossy edge. Hence the following theorem, as an immediate result of lemma 48, concludes this section. The proof is straightforward and left to the reader.

Theorem 49. *In planar additive flow networks, finding feasible flow f with maximum out-flow f_{out} is an NP-hard problem in the strong sense.*

4.5 Conclusion and Future Work

In this chapter we investigated the max-flow and shortest path problems for flow networks with additive gains and losses when the underlying graph is planar. In Sections 4.3 and 4.4, we show that both problems are NP-hard in the strong sense for planar additive flow networks, *i.e.* even when all the values of cost, gain and capacity functions assigned to every edge are bounded by polynomials in the size of the input network.

Hence, there is the question of existence of approximation algorithms for any of the two problems. To our best knowledge, no approximation algorithm has yet been suggested for any of those problems for additive flow networks (with or without any restriction on the structure of the underlying graph).

Chapter 3 proposes a compositional framework for the analysis of additive flow networks. As shown in Section 3.4 and Section 3.5 there exists a principal typing for every arbitrary additive flow network \mathcal{N} . An immediate result of this principal typing is efficient calculation of the value of maximum in/out flow for \mathcal{N} . As defined currently, this framework does not account for the

presence of cost functions on the flow. Hence, another problem left for future investigation is the problem of incorporating cost functions in that framework thereby allowing a compositional analysis of the shortest path problem in additive flow networks.

Chapter 5

Efficient Reassembling of Graphs

The *reassembling of a connected graph* $G = (\mathbf{V}, \mathbf{E})$ is the problem arising in the study of flow network analysis. The graph reassembling problem has a simple representation which is relative to a rooted binary tree \mathcal{B} – its so-called *reassembling tree*, with root node at the top and $n = |\mathbf{V}|$ leaf nodes at the bottom – where every cross-section corresponds to a partition of \mathbf{V} (every node of the cross-section is a representative of a block in the partition) such that:

- leaf nodes or bottom cross-section is the finest partition of \mathbf{V} with n one-vertex blocks,
- the top cross-section (*i.e.*, the root) is the coarsest partition with a single block, the entire set \mathbf{V} ,
- an internal node in an intermediate cross-section (or partition) is the result of merging its two children nodes (or blocks) in the cross-section (or partition) below it.

A component A is a subset of \mathbf{V} and its edge-boundary degree is the number of edges in G with one endpoint in A and one endpoint in $\mathbf{V} - A$ (which is the same as the number of half edges attached to A after all edges with both endpoints in A have been spliced together). The **maximum** edge-boundary degree encountered during the reassembling process is what we call the **α -measure** of the reassembling, and the **sum** of all edge-boundary degrees is its **β -measure**. The α -optimization (resp. β -optimization) of the reassembling of G is to determine an order Θ for splicing the edges that minimizes its α -measure (resp. β -measure).

There are different forms of reassembling, depending on restrictions and variations on the ordering Θ of the edges. To be consistent with our approach for compositional analysis of flow networks in Chapter 3, we considered the cases satisfying the condition that if the an edge between disjoint

components A and B is spliced, then all the edges between A and B are spliced at the same time. A particular case restricting the reassembling sequence is called *linear reassembling*, which requires that the next edge to be spliced must be adjacent to an already spliced edge. Subsequently, in a linear reassembling the height of reassembling tree \mathcal{B} is $(n - 1)$. A reassembling is *balanced*, when the \mathcal{B} has height $\lceil \log n \rceil$. Finally, it is the *general reassembling*, which is the case when the height of \mathcal{B} can be any number between $(n - 1)$ and $\lceil \log n \rceil$, namely there is no restriction on the reassembling sequence Θ .

5.1 Background and Motivation

Besides the questions of optimization and different reassembling variations which are naturally suggested, graph *reassembling* is part of the execution by programs in flow network typing framework as explained in Chapter 3. In network *reassembling*, the network is taken apart and reassembled in an order determined by the designer; in network *assembling*, the order in which components are put together is pre-determined, which is the order in which components become available to the designer for the sake of the compositional analysis of the network.

A flow network is a directed graph where vertices and edges are assigned various attributes that regulate flow through the network.¹ Programs for flow-network design are meant to connect network components in such a way that *typings at their interfaces*, *i.e.*, formally specified properties at their common boundaries, are satisfied. Network typings guarantee there are no conflicting data types when different components are connected, and insure that desirable properties of safe and secure operation are not violated by these connections, *i.e.*, they are *invariant properties* of the whole network construction.

A typing T for a *network component* \mathcal{M} formally expresses a constraining relationship between the flow variables denoting the input/output edges of \mathcal{M} . The smaller the set of input/output edges of \mathcal{M} (shown by $\partial(V(\mathcal{M}))$ where $V(\mathcal{M})$ is the set of vertices of component \mathcal{M}) is, the easier and more efficient it is to formulate and analyze the typing T and to test whether it is compatible

¹Such networks are typically more complex than the capacited directed graphs that algorithms for max-flow (and other related quantities) and its generalizations (*e.g.*, multicommodity max-flow) operate on.

with the typing T' of another network component \mathcal{M}' . Although every input/output edge of \mathcal{M} is directed, as input edge or output edge, the complexity of the formulation of T depends only on the number of input/output edges (*i.e.* $|\partial(V(\mathcal{M}))|$), not on their directions.

If k is a uniform upper bound on the number of input/output edges of all network components, as discussed in Chapter 3, the time complexity of reassembling the network without violating any component typing T can be made linear in the size n of the completed network and exponential in the bound k – not counting the preprocessing time $f(n)$ to determine an appropriate reassembling order. Hence, the smaller are k and $f(n)$, the more efficient is the construction of the entire network. From this follows the importance of minimizing the preprocessing time $f(n)$ for finding a reassembling strategy that also minimizes the bound k .

5.2 Problem Statement

We start with a gentle presentation of our graph reassembling problem.

Let $G = (\mathbf{V}, \mathbf{E})$ be a connected, undirected graph, with $|\mathbf{V}| = n \geq 1$ vertices and $|\mathbf{E}| = m$ edges. One version of the *reassembling* of G is edge-directed and can be defined by a *total order* Θ of the m edges of G . Informally and very simply, a total order Θ of the edges gives rise to a reassembling of G as follows:

- (1) In the disassembling step every edge is cut into two dangling halves, thus obtaining a collection of n one-vertex components, s.t. every the one-vertex component comprising vertex $v \in \mathbf{V}$ has $|\text{degree}(v)|$ half edges attached to it.
- (2) In the reassembling step, the two halves of every edge are connected in the order specified by Θ , obtaining larger and larger components in every splicing of half edges, until the original G is fully reassembled.

The reassembling of a graph G can also be defined relative to a binary tree \mathcal{B} – one root node at the top, n leaf nodes at the bottom – where every cross-section corresponds to a partition of \mathbf{V} (a block in the partition is a node in the cross-section) such that:

- the bottom (or first) cross-section (*i.e.*, all the leaves) is the finest partition of \mathbf{V} with n

one-vertex blocks,

- the top (or last) cross-section (*i.e.*, the root) is the coarsest partition with a single block, the entire set \mathbf{V} ,
- a node (or block) in an intermediate cross-section (or partition) is the result of merging its two children nodes (or blocks) in the cross-section (or partition) below it.

For convenience, we say *vertices* to refer to the vertices of G and *nodes* to refer to those of the tree \mathcal{B} . We call G the *input graph* and \mathcal{B} the *reassembling tree*.

We distinguish this definition of graph reassembling of G when it is in accordance with an order Θ (called *sequential reassembling*) from a later definition which is more suitable for parallel computation, called *binary reassembling*.

A *bridge* is defined as a yet-to-be-spliced edge between two disjoint components A and B ; we call such components *clusters*.² The set of bridges between A and B is denoted by $\partial(A, B)$. For the purpose of this report and the approach used in Chapter 3, when we reconnect one of the bridges in $\partial(A, B)$, we also reconnect all the other bridges in $\partial(A, B)$ and remove them from further consideration in Θ . Hence, in the reassembling of G according to Θ , there are at most m steps, rather than exactly m steps.

In the case when $B = \mathbf{V} - A$, the set $\partial(A, B)$ is the same as the set of input/output edges of A (equivalently the set of input/output edges component B). For the sake of brevity, instead of $\partial(A, \mathbf{V} - A)$, we write $\partial(A)$. The *edge-boundary degree* of a cluster A is $|\partial(A)|$ which in Chapter 3 is denoted by $\text{exDim}(A)$.

Various optimization problems can be associated with graph reassembling. Two such optimizations are the following, identified by the letters α and β throughout. We want to determine a reassembling tree \mathcal{B} for which:

- (α) the **maximum** edge-boundary degree encountered during reassembling is minimized, or
- (β) the **sum** of all edge-boundary degrees encountered during reassembling is minimized.

²These terms (*bridge*, *cluster*, and others, later in this report) are overloaded in graph-theoretical problems. We make our own use of these terms, and state it explicitly when our meaning is somewhat at variance with that elsewhere in the literature.

Initially, before we start reassembling, we always set the α -measure M_α to the **maximum** of all the vertex degrees, *i.e.*, $\max \{degree(v) | v \in \mathbf{V}\}$, and we set the β -measure M_β to the **sum** of the vertex degrees, *i.e.*, $\sum \{degree(v) | v \in \mathbf{V}\}$, regardless of which strategy, *i.e.*, the order Θ of edges, is selected for the reassembling. During reassembling, after we merge disjoint nonempty clusters A and B , we update the α -measure M_α to: $\max \{M_\alpha, |\partial(A \cup B)|\}$, and the β -measure M_β to: $(M_\beta + |\partial(A \cup B)|)$. The reassembling process terminates when only one cluster remains, which is also the set \mathbf{V} of all the vertices.

In what we call the *binary reassembling* of G , we reconnect bridges in several non-overlapping pairs of clusters simultaneously. That is, at every step – which we may call a *parallel step* for emphasis – we choose $k \geq 1$ and choose k cluster pairs $(A_1, B_1), \dots, (A_k, B_k)$, where $A_1, B_1, \dots, A_k, B_k$ are $2k$ pairwise disjoint clusters (*i.e.*, with pairwise disjoint subsets of vertices), and simultaneously reconnect all the bridges in $\sum_{1 \leq i \leq k} \partial(A_i, B_i)$. The subsets $A_1, B_1, \dots, A_k, B_k$ may or may not include all of the vertices, *i.e.*, in general $\sum_{1 \leq i \leq k} (A_i \uplus B_i) \subseteq \mathbf{V}$ rather than $= \mathbf{V}$. (We write “ \uplus ” to denote *disjoint union*.)

A binary reassembling is naturally viewed as vertex-directed and described by a binary tree \mathcal{B} – root at the top, leaves at the bottom – with n leaf nodes, one for each of the initial one-vertex clusters. Each non-leaf node in \mathcal{B} is a cluster $(A \uplus B)$ obtained by reconnecting all the bridges in $\partial(A, B)$ between the sibling clusters A and B . The first parallel step, $i = 0$, starts at the bottom in the reassembling process, by considering the n leaf nodes of \mathcal{B} and calculating the max (for α optimization) or the sum (for β optimization) of all vertex degrees. If h is the height of \mathcal{B} , the last parallel step is $i = h$, which corresponds to the root node of \mathcal{B} (the entire set \mathbf{V} of vertices) and produces the final α -measure and β -measure. Clearly, $\lceil \log n \rceil \leq h \leq n - 1$.

Every sequential reassembling of G can be viewed as a binary reassembling of G where, at every step, only one cluster pair (A, B) is selected and one nonempty set of bridges $\partial(A, B)$ is reconnected. Conversely, by serializing (or sequentializing) parallel steps, every binary reassembling which we call *strict* can be re-defined as a sequential reassembling. Details of the correspondence between sequential and binary reassemblings are in Appendix 1 of [64].

A binary reassembling is *strict* if the merging of a cluster pair (A, B) is restricted to the case

$\partial(A, B) \neq \emptyset$. If an α -optimal (resp. β -optimal) binary reassembling is strict, then its serialization is an α -optimal (resp. β -optimal) sequential reassembling.

The Linear Case. A possible and natural variation (or restriction) of graph reassembling is one which we call *linear*. If, at every step of the reassembling process, we require that the cluster pair (A, B) to be merged is such that one of the two clusters, A or B (or both at the first step), is a singleton set, then the resulting reassembling is *linear*. The binary tree \mathcal{B} describing a linear reassembling of a graph G with n vertices is therefore a degenerate tree of height $h = n - 1$.

Clearly, there can be no non-trivial parallel step which merges two (or more) cluster pairs in linear reassembling, *i.e.*, no step which simultaneously merges disjoint cluster pairs (A_1, B_1) and (A_2, B_2) to form the non-singleton clusters $(A_1 \uplus B_1)$ and $(A_2 \uplus B_2)$, before they are merged to form the cluster $((A_1 \uplus B_1) \uplus (A_2 \uplus B_2))$.

Another useful way of understanding a linear reassembling of graph G is in its sequential formulation (when the reassembling is strict): The order Θ for reconnecting the edges is such that the next edge to be reconnected is always adjacent to an already re-reconnected edge, which enforces the requirement that the next cluster pair (A, B) to be merged is always such that A or B is a singleton set.

The Balanced Case. There are other natural variations of graph reassembling, such as *balanced reassembling*, whose binary tree \mathcal{B} maximizes the merging of cluster pairs at every parallel step. In other words, in balanced case, at every level of the reassembling, there is a maximum number of pairs of blocks which are each merged into a block at the next level up. The height h of \mathcal{B} is therefore $\lceil \log n \rceil$.

Main Results. In this report, we first prove that α -optimization and β -optimization of linear and balanced reassembling are both NP-hard problems. We obtain these results by showing that:

- α -optimization of linear reassembling and *minimum-cutwidth linear arrangement* (CutWidth) are polynomially-reducible to each other,

- β -optimization of linear reassembling and *minimum-cost linear arrangement* (MinArr) are polynomially-reducible to each other.
- there is a sequence of several polynomial-time reductions from *minimum bisection of graphs* (MinBisection) to α -optimization of balanced reassembling of graphs,
- there is a sequence of several polynomial-time reductions from *clique cover of graphs* (called CliqueCover) to β -optimization of balanced reassembling of graphs.

CutWidth, MinArr, MinBisection and CliqueCover have been extensively studied: They are all NP-hard in general, but, in a few non-trivial special cases, amenable to low-degree polynomial-time solutions [79]. By our polynomial-reducibility results, these polynomial-time solutions are transferable to the α -optimization and β -optimization of linear reassembling. This leaves open the problem of identifying classes of graphs, whether of practical or theoretical significance, for which there are low-degree polynomial-time solutions for our two optimization problems.

Later on in Chapter 6, we study the general reassembling problem by showing that the β -optimization of general reassembling of weighted/multi-graphs is an NP-hard problem. We show this result by translating this problem into the well-known *Tree Layout* problem, as a special case of graph embedding problems.

5.3 Formal Definitions and Preliminary Lemmas

Let $G = (\mathbf{V}, \mathbf{E})$ be a simple undirected graph, with $|\mathbf{V}| = n \geq 1$ and $|\mathbf{E}| = m \geq 1$. Without loss of generality, throughout this chapter, it is assumed that G is connected.

As pointed out in Section 5.2, there are two distinct definitions of the reassembling of G . The first and easier to state informally is *sequential reassembling*. The second is *binary reassembling*, which is more convenient for the optimization problems we seek to study and more suitable for our main purpose, *i.e.* providing the possibility of parallel computation for the sake of the efficiency of the compositional framework represented in Section 3. Our analyses in this chapter are based on the binary reassembling and its variations; Hence, in this Chapter we only consider the binary case. In what follows, we briefly represent some basic definitions and notations that are used in the

future sections.

We write $\text{GRAPHS}(2^*)$ to denote the class of all simple graphs $G = (\mathbf{V}, \mathbf{E})$ where $|\mathbf{V}|$ is a power-of-2.

Definition 50 (*Minimum Bisection Problem*). A *bisection* of the graph $G = (\mathbf{V}, \mathbf{E})$ is a partition $\{A, B\}$ of $\mathbf{V}(G)$ with two blocks of equal size:

$$|A| = |B|, \quad A \cap B = \emptyset, \quad \text{and} \quad A \cup B = \mathbf{V}.$$

(Our running assumption is that \mathbf{V} has an even number of vertices and can therefore be partitioned into two equal-size blocks.) We say the bisection $\{A, B\}$ is of *type* (X, Y) iff $X, Y \subseteq \mathbf{V}(G)$ such that:

- either $X \cap Y = \emptyset$, $X \subseteq A$, and $Y \subseteq B$,
- or $X \cap Y = \emptyset$, $X \subseteq B$, and $Y \subseteq A$.

A *minimum bisection* $\{A, B\}$ of G is one that minimizes the set of bridges $\partial(A, B)$ between A and B , i.e.,

$$|\partial(A, B)| \leq \min \{ |\partial(A', B')| \mid \{A', B'\} \text{ is a bisection of } G \}.$$

The *minimum bisection problem*, MinBisection , asks for finding a minimum bisection of a given input graph G . MinBisection is known to be NP-hard [37].

We write $\text{MinBisection}(2^*)$ for the restriction of MinBisection to the class $\text{GRAPHS}(2^*)$. Lemma 95 asserts that $\text{MinBisection}(2^*)$ is also NP-hard. □

Definition 51 (*Clique Cover Problem*). Given an integer $k \geq 1$, a *k-clique cover* of the graph $G = (\mathbf{V}, \mathbf{E})$ is a partition of $\mathbf{V}(G)$ into k disjoint subsets $\{\mathbf{V}_1, \dots, \mathbf{V}_k\}$ such that each of the induced subgraphs $G[\mathbf{V}_1], \dots, G[\mathbf{V}_k]$ is a complete graph (or a clique in G).

The *k-clique cover problem*, $k\text{-CliqueCover}$ in shorthand, is a (yes,no) question that asks whether a graph G has a k -clique cover. $k\text{-CliqueCover}$ is polynomial-time solvable for $k = 1, 2$, and is known to be NP-complete for every $k \geq 3$ [58]. □

5.3.1 Binary Graph-Reassembling

The notion of *binary reassembling* of graph $G = (\mathbf{V}, \mathbf{E})$ is directly related to the notion of a *binary tree* defined over a set of *leaf nodes* $\mathbf{V} = \{v_1, \dots, v_n\}$ and internal nodes which correspond to the subsets of \mathbf{V} . Our definition of *binary trees* is not standard, but is more convenient for our purposes.³

Definition 52 (*Binary trees*). An (*unordered*) *binary tree* \mathcal{B} over $\mathbf{V} = \{v_1, \dots, v_n\}$ is a collection of non-empty subsets of \mathbf{V} satisfying three conditions:

1. For every $v \in \mathbf{V}$, the singleton set $\{v\}$ is in \mathcal{B} .
2. The full set \mathbf{V} is in \mathcal{B} .
3. For every $X \in \mathcal{B}$, there is a unique $Y \in \mathcal{B}$ such that: $X \cap Y = \emptyset$ and $(X \cup Y) \in \mathcal{B}$.

The *leaf nodes* of \mathcal{B} are the singleton sets $\{v\}$ for some $v \in \mathbf{V}$, and the *root node* of \mathcal{B} is the full set \mathbf{V} . Depending on the context, we may refer to the members of \mathcal{B} as its *nodes* or as its *clusters*. Some properties of \mathcal{B} are stated in the next two propositions. □

Our definition of Binary tree slightly differs from the standard definition in the sense that we are not concerned regarding the order of the nodes with the same parent. Formally speaking given binary tree \mathcal{B} and $X, Y, Z \in \mathcal{B}$ where X and Y are direct children of Z (*i.e.* $X \cap Y = \emptyset$ and $X \cup Y = Z$), as apposed to the standard definition, in our definition of binary tree no ordering is assigned to the nodes X and Y .

Proposition 53 (Properties of binary trees). *Let \mathcal{B} be a binary tree as in Definition 52, let $v \in \mathbf{V}$, and let:*

$$\{v\} = X_0 \not\subseteq X_1 \not\subseteq X_2 \not\subseteq \dots \not\subseteq X_p = \mathbf{V} \quad (\dagger)$$

be a maximal sequence of nested clusters from \mathcal{B} . We then have:

³A standard definition of a binary tree T makes T a subset of the set of finite binary strings $\{0, 1\}^*$ such that:

- T is prefix-closed, *i.e.*, if $t \in T$ and u is a prefix of t , then $u \in T$.
- Every node has two children, *i.e.*, $t0 \in T$ iff $t1 \in T$ for every $t \in \{0, 1\}^*$.

The *root node* of T is the empty string ε , and a *leaf node* of T is a string $t \in T$ without children, *i.e.*, both $t0 \notin T$ and $t1 \notin T$.

1. The sequence in (†) is uniquely defined, i.e., every maximal nested sequence starting from $\{v\}$ is the same.
2. For every cluster $Y \in \mathcal{B}$, if $\{v\} \subseteq Y$, then $Y \in \{X_0, \dots, X_p\}$.
3. There are pairwise disjoint clusters $\{Y_0, \dots, Y_{p-1}\} \subseteq \mathcal{B}$ such that, for every $0 \leq i < p$:

$$X_i \cap Y_i = \emptyset \quad \text{and} \quad X_i \cup Y_i = X_{i+1}.$$

Based on this proposition, we use the following terminology:

- We call the sequence in (†), which is unique by part 1, the *path* from the leaf node $\{v\}$ to the root node \mathbf{V} .
- Every cluster containing v is one of the nodes along this unique path, according to part 2.
- In part 3, we call Y_i the unique *sibling* of X_i , whose unique common *parent* is X_{i+1} , for every $0 \leq i < p$.

Proof. Part 1 is a consequence of the third condition in Definition 52, which prevents any cluster/node in \mathcal{B} from having two distinct parents.

For part 2, consider the nested sequence $\{v\} \subseteq Y \subseteq \mathbf{V}$, and extend it to a maximal nested sequence, which is uniquely defined by part 1. This implies Y is one of the nodes along the path from $\{v\}$ to the root \mathbf{V} .

Part 3 is another consequence of the third condition in Definition 52: For every $0 \leq i < p$, Y_i is the unique cluster in \mathcal{B} such that $X_i \cap Y_i = \emptyset$ and $X_i \cup Y_i \in \mathcal{B}$. Remaining details omitted. \square

Proposition 54 (Properties of binary trees). *Let \mathcal{B} be a binary tree as in Definition 52. We then have:*

1. For all clusters $X, Y \in \mathcal{B}$, if $X \cap Y \neq \emptyset$ then $X \subseteq Y$ or $Y \subseteq X$.
2. For every cluster $X \in \mathcal{B}$, the sub-collection of clusters $\mathcal{B}_X := \{Y \in \mathcal{B} \mid Y \subseteq X\}$ is a binary tree over X , with root node X .
3. \mathcal{B} is a collection of $(2n - 1)$ clusters.

Proof. For part 1, let $Z = X \cap Y \neq \emptyset$ and consider the maximal sequence of nested clusters which extends $Z \subseteq X \subseteq \mathbf{V}$ or $Z \subseteq Y \subseteq \mathbf{V}$. By part 1 of Proposition 53, both X and Y must occur along this nested sequence.

For part 2, we directly check that \mathcal{B}_X satisfies the three defining properties of a binary tree. Straightforward details omitted.

Part 3 is proved by induction on $n \geq 1$. For the induction hypothesis, we assume the statement is true for every binary tree with less than n leaf nodes, and we then prove that the statement is true for an arbitrary binary tree over \mathbf{V} with $|\mathbf{V}| = n$. Consider a largest cluster $X \in \mathcal{B}$ such that $X \neq \mathbf{V}$. There is a unique $Y \in \mathcal{B}$ such that $X \cap Y = \emptyset$ and $X \cup Y \in \mathcal{B}$. Because X is largest in size but smaller than \mathbf{V} , it must be that $X \cup Y = \mathbf{V}$, which implies that $\{X, Y\}$ is a two-block partition of \mathbf{V} . Consider now the subtrees \mathcal{B}_X and \mathcal{B}_Y , apply the induction hypothesis to each, and draw the desired conclusion. \square

A common measure for a standard definition of binary trees is *height*, which becomes for our notion of binary trees in Definition 52:

$$\text{height}(\mathcal{B}) := \max \left\{ p \mid \text{there is } v \in \mathbf{V} \text{ such that} \right. \\ \left. \{v\} = X_0 \subsetneq X_1 \subsetneq \dots \subsetneq X_p = \mathbf{V} \text{ is a maximal sequence of nested clusters} \right\}.$$

For a particular node/cluster $X \in \mathcal{B}$, the subtree of \mathcal{B} rooted at X is \mathcal{B}_X , by part 2 in Proposition 54. The height of X in \mathcal{B} is therefore $\text{height}_{\mathcal{B}}(X) := \text{height}(\mathcal{B}_X)$.

If $X, Y \subseteq \mathbf{V}$ are disjoint sets of vertices, $\partial_G(X, Y)$ is the subset of edges of G with one endpoint in each of X and Y . If $Y = \mathbf{V} - X$, we write $\partial_G(X)$ instead of $\partial_G(X, \mathbf{V} - X)$.

Definition 55 (*Binary reassembling*). A *binary reassembling of the graph* $G = (\mathbf{V}, \mathbf{E})$ is simply defined by a pair (G, \mathcal{B}) where \mathcal{B} is a binary tree over \mathbf{V} , as in Definition 52.

Given a binary reassembling (G, \mathcal{B}) of G , two measures are of particular interest for our later analysis, namely, for every cluster $X \in \mathcal{B}$, the *degree* of X and the *height* of X :

$$\text{degree}_{G, \mathcal{B}}(X) := |\partial_G(X)| \quad \text{and} \quad \text{height}_{G, \mathcal{B}}(X) := \text{height}_{\mathcal{B}}(X).$$

If the context makes clear the binary reassembling (G, \mathcal{B}) – respectively, the binary tree \mathcal{B} – relative to which these measures are defined, we write $degree(X)$ and $height(X)$ – respectively, $degree_G(X)$ and $height_G(X)$ – instead of $degree_{G,\mathcal{B}}(X)$ and $height_{G,\mathcal{B}}(X)$.⁴ \square

Definition 56 (*Strict binary reassembling*). We say the binary reassembling (G, \mathcal{B}) in Definition 55 is *strict* if it satisfies the following condition: For all clusters $X, Y \in \mathcal{B}$, if X and Y are sibling nodes, then $\partial_G(X, Y) \neq \emptyset$. (In Definition 55, when we merge sibling clusters $X, Y \in \mathcal{B}$, we do not require that $\partial_G(X, Y) \neq \emptyset$.) \square

5.3.2 Optimization Problems

The following definition repeats a definition in Section 5.2 more formally.

Definition 57 (*Measures on the reassembling of a graph*). Let (G, \mathcal{B}) be a binary reassembling of G . We define the measures α and β on (G, \mathcal{B}) as follows:

$$\alpha(G, \mathcal{B}) := \max \{ degree_{G,\mathcal{B}}(X) \mid X \in \mathcal{B} \},$$

$$\beta(G, \mathcal{B}) := \sum \{ degree_{G,\mathcal{B}}(X) \mid X \in \mathcal{B} \}.$$

An optimization problem arises with the minimization of each of these measures. For example, the optimal α -measure of graph G is:

$$\alpha(G) = \min \{ \alpha(G, \mathcal{B}) \mid (G, \mathcal{B}) \text{ is a binary reassembling of } G \}.$$

We say the binary reassembling (G, \mathcal{B}) is α -*optimal* iff $\alpha(G, \mathcal{B}) = \alpha(G)$. We leave to the reader the obvious similar definition of what it means for the binary reassembling (G, \mathcal{B}) to be β -*optimal*.

\square

⁴ Our binary reassembling of G can be viewed as the “hierarchical clustering” of G , similar to a method of analysis in data mining, though used for a different purpose. Our binary reassembling mimicks what is called “agglomerative, or bottom-up, hierarchical clustering” in data mining.

5.3.3 Linear Graph-Reassembling

Let $G = (\mathbf{V}, \mathbf{E})$ be a simple undirected graph with $|\mathbf{V}| = n$. We say the binary reassembling (G, \mathcal{B}) is *linear* if \mathcal{B} is a *linear binary tree* over \mathbf{V} , i.e., all the clusters of size ≥ 2 forms a single nested chain of length $(n - 1)$:

$$X_1 \subsetneq X_2 \subsetneq \cdots \subsetneq X_{n-1} = \mathbf{V} \quad (\text{A})$$

This implies the height of \mathcal{B} is $(n - 1)$. By part 3 in Proposition 53, there are n leaf nodes/singleton clusters $\{Y_0, \dots, Y_{n-1}\} \subseteq \mathcal{B}$ such that $X_1 = Y_0 \cup Y_1$ and for every $1 \leq i \leq n - 2$:

$$X_i \cap Y_{i+1} = \emptyset \quad \text{and} \quad X_i \cup Y_{i+1} = X_{i+1}. \quad (\text{B})$$

We use the letter \mathcal{L} to denote a linear binary tree, and write (G, \mathcal{L}) to denote a linear reassembling of G .

In Definition 58, we mostly use the notation and conventions of [79] and the references therein. We write \overline{vw} to denote the edge connecting vertex v and vertex w .

Definition 58 (*Linear arrangements and cutwidths*). A *linear arrangement* φ of the graph $G = (\mathbf{V}, \mathbf{E})$, where $|\mathbf{V}| = n$, is a bijection $\varphi : \mathbf{V} \rightarrow \{1, \dots, n\}$. We refer to this linear arrangement by writing (G, φ) .

Following [79], given linear arrangement (G, φ) and $i \in \{1, \dots, n\}$, we define a two-block partition of the vertices, $\mathbf{V} = L(G, \varphi, i) \uplus R(G, \varphi, i)$, where:

$$L(G, \varphi, i) := \{v \in \mathbf{V} \mid \varphi(v) \leq i\} \quad \text{and} \quad R(G, \varphi, i) := \{w \in \mathbf{V} \mid \varphi(w) > i\}.$$

The *edge cut at position i* , denoted $\zeta(G, \varphi, i)$, is the number of edges connecting $L(G, \varphi, i)$ and $R(G, \varphi, i)$:

$$\zeta(G, \varphi, i) := \left| \{ \overline{vw} \in \mathbf{E} \mid v \in L(G, \varphi, i) \text{ and } w \in R(G, \varphi, i) \} \right|.$$

In our notation in Definition 55, we have:

$$\zeta(G, \varphi, i) = |\partial(L(G, \varphi, i), R(G, \varphi, i))| = \text{degree}(L(G, \varphi, i)).$$

The length of \overline{vw} in the linear arrangement (G, φ) , denoted $\xi(G, \varphi, \overline{vw})$, is “1 + the number of vertices between v and w ”:

$$\xi(G, \varphi, \overline{vw}) := |\varphi(v) - \varphi(w)|.$$

The α -measure and β -measure of the linear arrangement (G, φ) are defined by:

$$\begin{aligned} \alpha(G, \varphi) &:= \max \{ \zeta(G, \varphi, i) \mid 1 \leq i \leq n \} = \max \{ \text{degree}(L(G, \varphi, i)) \mid 1 \leq i \leq n \}, \\ \beta(G, \varphi) &:= \sum \{ \zeta(G, \varphi, i) \mid 1 \leq i \leq n \} = \sum \{ \text{degree}(L(G, \varphi, i)) \mid 1 \leq i \leq n \}. \end{aligned}$$

In the literature, $\alpha(G, \varphi)$ is called the *cutwidth* of the linear arrangement (G, φ) . The *cost* of the linear arrangement (G, φ) is usually defined as the total length of all the edges relative to (G, φ) , i.e., the cost is the measure $\gamma(G, \varphi)$ given by:

$$\gamma(G, \varphi) := \sum \{ \xi(G, \varphi, \overline{vw}) \mid \overline{vw} \in \mathbf{E} \}.$$

However, by Lemma 59 below, $\beta(G, \varphi)$ is equal to $\gamma(G, \varphi)$. □

Lemma 59. *For every linear arrangement (G, φ) , we have $\beta(G, \varphi) = \gamma(G, \varphi)$.*

Proof. We have to prove that

$$\sum \{ \zeta(G, \varphi, i) \mid 1 \leq i \leq n \} = \sum \{ \xi(G, \varphi, \overline{vw}) \mid \overline{vw} \in \mathbf{E} \}.$$

This equality holds whether G is connected or not. So, a formal proof (omitted) can be written by induction on the number $m \geq 0$ of edges in G . But informally, for every edge $\overline{vw} \in \mathbf{E}$, if $\varphi(v) = i$ and $\varphi(w) = j$ with $i < j$, then its length $\xi(G, \varphi, \overline{vw}) = j - i$. In this case, the length of edge \overline{vw} contributes one unit to each of $(j - i)$ consecutive edge cuts: $\zeta(G, \varphi, i), \dots, \zeta(G, \varphi, j - 1)$. Hence,

if we delete edge \overline{vw} from the graph, we decrease the two quantities:

$$\sum \{ \zeta(G, \varphi, i) \mid 1 \leq i \leq n \} \quad \text{and} \quad \sum \{ \xi(G, \varphi, \overline{vw}) \mid \overline{vw} \in \mathbf{E} \}.$$

by exactly the same amount $(j - i)$. The desired conclusion follows. \square

Definition 60 (*Optimal linear arrangements*). Let $G = (\mathbf{V}, \mathbf{E})$ be a simple undirected graph. We say the linear arrangement (G, φ) is α -optimal if:

$$\alpha(G, \varphi) = \min \{ \alpha(G, \varphi') \mid (G, \varphi') \text{ is a linear arrangement} \}.$$

The α -optimal linear arrangement problem is the problem of defining a bijection $\varphi : \mathbf{V} \rightarrow \{1, \dots, n\}$ such that (G, φ) is α -optimal. We define similarly the β -optimal linear arrangement problem. \square

Definition 61 (*Linear arrangement induced by linear reassembling*). Let $G = (\mathbf{V}, \mathbf{E})$ be a simple undirected graph and (G, \mathcal{L}) be a linear reassembling of G . Using the notation in (A) and (B) in the opening paragraph of Section 5.3.3, the n leaf nodes (or singleton clusters) of \mathcal{L} are: $Y_0, Y_1, Y_2, \dots, Y_{n-1}$. Observe that the order of the singletons Y_2, \dots, Y_{n-1} and, therefore, the order of the $(n - 2)$ vertices in $Y_2 \cup \dots \cup Y_{n-1}$ is uniquely determined by the chain in (A), but this is not the case for the order in which we write Y_0 and Y_1 , i.e., the first cluster X_1 in (A) is equal to both $Y_0 \cup Y_1$ and $Y_1 \cup Y_0$.

We want to extract a linear arrangement (G, φ) from the linear reassembling (G, \mathcal{L}) . This is achieved by defining $\varphi : \mathbf{V} \rightarrow \{1, \dots, n\}$ as follows:

- Let $Y_0 = \{v\}$ and $Y_1 = \{v'\}$.
 - If $\text{degree}(v) \leq \text{degree}(v')$ then set $\varphi(v) := 1$ and $\varphi(v') := 2$.
 - If $\text{degree}(v) \geq \text{degree}(v')$ then set $\varphi(v') := 1$ and $\varphi(v) := 2$.
- For every $2 \leq i \leq n - 1$, if $Y_i = \{v\}$ then set $\varphi(v) := i + 1$.

It is possible that $\text{degree}(v) = \text{degree}(v')$, in which case φ may place v first and v' second, or v' first and v second. This ambiguity is harmless for our analysis, in that it does not affect the

α -measure and the β -measure of the linear arrangement (G, φ) .

Whether φ places v first and v' second, or v' first and v second, we call (G, φ) a linear arrangement of G induced by the linear reassembling (G, \mathcal{L}) .

Note that, for every $1 \leq i \leq n - 1$, we have the equality $X_i = L(G, \varphi, i)$, where $L(G, \varphi, i)$ is the set of vertices at position i and to the left of it, as in Definition 58. \square

Definition 62 (*Linear reassembling induced by linear arrangement*). Let $G = (\mathbf{V}, \mathbf{E})$ be a simple undirected graph and let (G, φ) be a linear arrangement of G . We extract a linear reassembling (G, \mathcal{L}) from (G, φ) . The n leaf nodes/singleton clusters of \mathcal{L} are:

$$\{\{v\} \mid v \in \mathbf{V}\} = \{\{\varphi^{-1}(i)\} \mid 1 \leq i \leq n\}.$$

The $(n - 1)$ non-leaf nodes/clusters of \mathcal{L} are:

$$\begin{aligned} X_1 &:= \{\varphi^{-1}(1), \varphi^{-1}(2)\}, \\ X_2 &:= \{\varphi^{-1}(1), \varphi^{-1}(2), \varphi^{-1}(3)\}, \\ &\dots \\ X_{n-1} &:= \{\varphi^{-1}(1), \varphi^{-1}(2), \varphi^{-1}(3), \dots, \varphi^{-1}(n)\} = \mathbf{V}. \end{aligned}$$

We call (G, \mathcal{L}) the linear reassembling of G induced by the linear arrangement (G, φ) . \square

5.4 Examples

We present several simple examples to illustrate notions mentioned in Sections 5.2 and 5.3. We first introduce a convenient notation for specifying binary trees over a set \mathbf{V} of vertices. Let $2^{\mathbf{V}}$ denote the power set of \mathbf{V} . We define a binary operation $\overline{\mathcal{X} \mathcal{Y}}$ for all $\mathcal{X} \subseteq 2^{\mathbf{V}}$ and $\mathcal{Y} \subseteq 2^{\mathbf{V}}$ by:

$$\overline{\mathcal{X} \mathcal{Y}} := (\mathcal{X} \cup \mathcal{Y}) \cup ((\bigcup \mathcal{X}) \cup (\bigcup \mathcal{Y}))$$

The examples below illustrate how we use this operation “ $\overline{\quad}$ ”.

We overload the overline notation “ $\overline{}$ ” to denote a nonempty subset of \mathbf{V} , i.e., $\overline{v_1 \cdots v_k}$ means $\{v_1, \dots, v_k\}$. In particular, the edge connecting v and w is denoted by the two-vertex set $\overline{vw} = \{v, w\}$.

The context will make clear whether we use “ $\overline{}$ ” to refer to a set of subsets of \mathbf{V} (in the case of binary trees) or to just a subset of \mathbf{V} .

Example 63. The hypercube graph Q_3 is shown on the left in Figure 5.1, and three of its reassemblings are shown on the right in Figure 5.1. The top reassembling is neither balanced nor linear; the middle one is *balanced*; and the bottom one is *linear*.

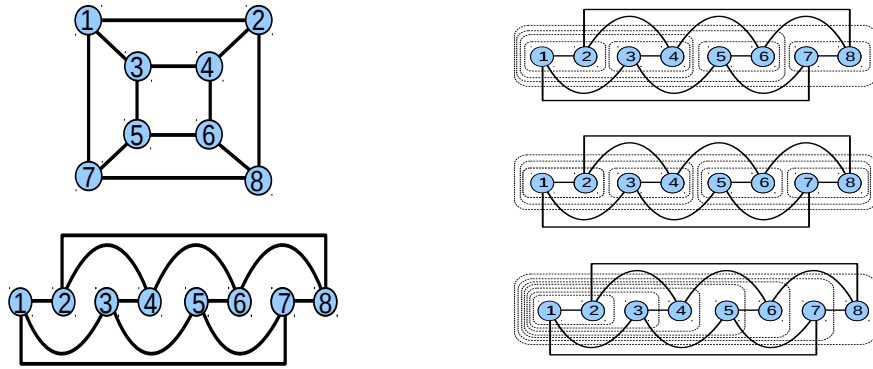


Figure 5.1: Two different drawings of the hypercube graph Q_3 (on the left) and three of its reassemblings (on the right).

For each of the three reassemblings on the right in Figure 5.1, from top to bottom, we list the unique binary tree \mathcal{B} over the vertices $\{1, \dots, 8\}$ underlying it (in its *binary reassembling* formulation) and one of the orderings Θ of the edges $\{\overline{12}, \dots, \overline{78}\}$ inducing it (in its *sequential reassembling* formulation):

$$\begin{aligned}
 \mathcal{B}_1 &= \left\{ \begin{array}{c} \overline{\overline{\overline{12}34}56}78 \\ \overline{12} \overline{34} \overline{56} \overline{78} \end{array} \right. & \Theta_1^{Q_3} &= \overline{12} \overline{34} \overline{56} \overline{78} \overline{13} \overline{35} \overline{57} \dots \\
 \mathcal{B}_2 &= \left\{ \begin{array}{c} \overline{\overline{\overline{12}34}56}78 \\ \overline{12} \overline{34} \overline{56} \overline{78} \end{array} \right. & \Theta_2^{Q_3} &= \overline{12} \overline{34} \overline{56} \overline{78} \overline{13} \overline{57} \overline{35} \dots \text{(balanced)} \\
 \mathcal{B}_3 &= \left\{ \begin{array}{c} \overline{\overline{\overline{12}34}56}78 \\ \overline{12} \overline{34} \overline{56} \overline{78} \end{array} \right. & \Theta_3^{Q_3} &= \overline{12} \overline{13} \overline{34} \overline{35} \overline{56} \overline{57} \overline{78} \dots \text{(linear)}
 \end{aligned}$$

where, for simplicity, we write just “ v ” instead of the cumbersome $\{\{v\}\} = \overline{\overline{v}}$. Thus, for example, two of the subsets of \mathcal{B}_1 above appear as “ $\overline{1\ 2}$ ” and “ $\overline{\overline{1\ 2\ 3\ 4}}$ ”, and if we expand them in full, we obtain:

$$\overline{1\ 2} = \{\{1\}, \{2\}, \{1, 2\}\} \quad \text{and} \quad \overline{\overline{1\ 2\ 3\ 4}} = \{\{1\}, \{2\}, \{3\}, \{4\}, \{1, 2\}, \{3, 4\}, \{1, 2, 3, 4\}\}.$$

The ellipsis “ \dots ” in the definition of $\Theta_i^{Q_3}$ above are the remaining edges of Q_3 , which can be listed in any order without changing the reassembling.⁵ A simple calculation of the α -measure and β -measure of these three reassemblings of Q_3 produces:

$$\begin{aligned} \alpha(Q_3, \mathcal{B}_1) &= \alpha(Q_3, \mathcal{B}_2) = 4 & \text{and} & \quad \alpha(Q_3, \mathcal{B}_3) = 5, \\ \beta(Q_3, \mathcal{B}_1) &= \beta(Q_3, \mathcal{B}_2) = 48 & \text{and} & \quad \beta(Q_3, \mathcal{B}_3) = 49. \end{aligned}$$

By exhaustive inspection (details omitted), (Q_3, \mathcal{B}_1) is both α -optimal and β -optimal for the class of all binary reassemblings of Q_3 . Because the α -measure and β -measure of (Q_3, \mathcal{B}_1) and those of (Q_3, \mathcal{B}_2) are equal, (Q_3, \mathcal{B}_2) is also both α -optimal and β -optimal for the class of all binary reassemblings and, therefore, for the smaller class of all *balanced* reassemblings of which it is a member. By exhaustive inspection again, (Q_3, \mathcal{B}_3) is both α -optimal and β -optimal for the subclass of all *linear* reassemblings, but not for the full class of all binary reassemblings of Q_3 . \square

Example 64. The complete graph K_8 on 8 vertices is shown in Figure 5.2. We can carry out three reassemblings of K_8 by using the binary trees \mathcal{B}_1 , \mathcal{B}_2 , and \mathcal{B}_3 , from Example 63 again.

⁵ We qualify $\Theta_i^{Q_3}$ with the superscript “ Q_3 ” because it depends on the graph Q_3 . The same ordering of the edges may not be valid for a sequential reassembling of another 8-vertex graph with a set of edges different from that of Q_3 . This is not the case for the binary tree \mathcal{B} underlying the binary reassembling (G, \mathcal{B}) of a graph $G = (\mathbf{V}, \mathbf{E})$; that is, regardless of the placement of edges in G , the tree \mathcal{B} over \mathbf{V} is valid for the binary reassembling (G, \mathcal{B}) and again for the binary reassembling (G', \mathcal{B}) of every graph $G' = (\mathbf{V}, \mathbf{E}')$ over the same set \mathbf{V} of vertices.

The resulting β -measure is $\beta(S_7, \mathcal{B}_5) = 29$. Note that \mathcal{B}_5 is also linear. Hence, (S_7, \mathcal{B}_5) is also β -optimal for the class of linear reassemblings of S_7 . \square

Example 66. The binary tree \mathcal{B}_3 in Examples 63, 64, and 65, is a linear binary tree. Written in full, using the notation in (A) at the beginning of Section 5.3.3, the non-singleton sets of \mathcal{B}_3 are:

$$\begin{aligned} X_1 &= \overline{1\ 2}, & X_2 &= \overline{1\ 2\ 3}, & X_3 &= \overline{1\ 2\ 3\ 4}, & X_4 &= \overline{1\ 2\ 3\ 4\ 5}, \\ X_5 &= \overline{1\ 2\ 3\ 4\ 5\ 6}, & X_6 &= \overline{1\ 2\ 3\ 4\ 5\ 6\ 7}, & X_7 &= \overline{1\ 2\ 3\ 4\ 5\ 6\ 7\ 8}. \end{aligned}$$

The singleton sets of \mathcal{B}_3 are:

$$Y_0 = \overline{1}, Y_1 = \overline{2}, Y_2 = \overline{3}, Y_3 = \overline{4}, Y_4 = \overline{5}, Y_5 = \overline{6}, Y_6 = \overline{7}, Y_7 = \overline{8}.$$

The linear arrangement φ_3 induced by the linear reassembling (S_7, \mathcal{B}_3) in Example 65 is (see Definition 61):

$$\varphi_3(2) = 1, \varphi_3(1) = 2, \varphi_3(3) = 3, \varphi_3(4) = 4, \varphi_3(5) = 5, \varphi_3(6) = 6, \varphi_3(7) = 7, \varphi_3(8) = 8,$$

rather than the linear arrangement φ'_3 :

$$\varphi'_3(1) = 1, \varphi'_3(2) = 2, \varphi'_3(3) = 3, \varphi'_3(4) = 4, \varphi'_3(5) = 5, \varphi'_3(6) = 6, \varphi'_3(7) = 7, \varphi'_3(8) = 8,$$

because $\text{degree}_{S_7}(2) = 1 < 7 = \text{degree}_{S_7}(1)$. The difference between φ_3 and φ'_3 is in the placement of the two first vertices: vertex “1” and vertex “2”. \square

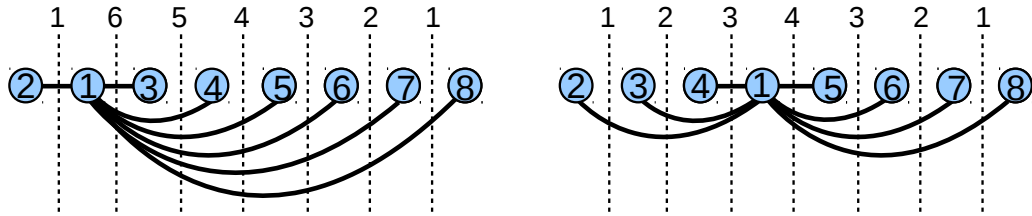
Example 67. The binary tree \mathcal{B}_5 in Example 65 is a linear binary tree. As in Example 66, it is straightforward to specify the singleton and non-singleton sets of \mathcal{B}_5 (omitted here) to fit the notation of (A) and (B) at the beginning of Section 5.3.3. There are two possible linear arrangements, φ_5 and φ'_5 , which are induced by the linear reassembling (S_7, \mathcal{B}_5) , because $\text{degree}_{S_7}(2) =$

$degree_{S_7}(3) = 1$ (see Definition 61), namely:

$$\varphi_5(2) = 1, \varphi_5(3) = 2, \varphi_5(4) = 3, \varphi_5(1) = 4, \varphi_5(5) = 5, \varphi_5(6) = 6, \varphi_5(7) = 7, \varphi_5(8) = 8,$$

$$\varphi'_5(3) = 1, \varphi'_5(2) = 2, \varphi'_5(4) = 3, \varphi'_5(1) = 4, \varphi'_5(5) = 5, \varphi'_5(6) = 6, \varphi'_5(7) = 7, \varphi'_5(8) = 8.$$

The difference between φ_5 and φ'_5 is in the placement of the two first vertices: vertex “2” and vertex “3”. In contrast to φ_3 and φ'_3 in Example 66, both φ_5 and φ'_5 are valid as linear arrangements induced by the linear reassembling (S_7, \mathcal{B}_5) . A comparison between φ_3 and φ_5 is shown in Figure 5.4. □



$$\begin{aligned} \alpha(S_7, \varphi_3) &= \max \{1, 6, 5, 4, 3, 2, 1\} = 6 & \alpha(S_7, \varphi_5) &= \max \{1, 2, 3, 4, 3, 2, 1\} = 4 \\ \beta(S_7, \varphi_3) &= \sum \{1, 6, 5, 4, 3, 2, 1\} = 22 & \beta(S_7, \varphi_5) &= \sum \{1, 2, 3, 4, 3, 2, 1\} = 16 \end{aligned}$$

Figure 5.4: Comparison of linear arrangements (S_7, φ_3) in Example 66 and (S_7, φ_5) in Example 67.

Example 68. This is a continuation of Example 66. The linear reassembling induced by the linear arrangement (S_7, φ_3) is precisely (S_7, \mathcal{B}_3) , but so is the linear reassembling induced by the linear arrangement (S_7, φ'_3) again the same (S_7, \mathcal{B}_3) , according to Definition 62. This means: *linear arrangements make distinction that linear reassemblings do not make*. This difference is in the placement of the two first vertices, specifically:

- The α -measure and β -measure of a *linear arrangement* generally depend on which vertex is placed first and which is placed second.
- The α -measure and β -measure of a *linear reassembling* do *not* distinguish between a first and second vertex and do *not* depend on which is placed first and which is placed second.

As an example, consider the linear arrangements (S_7, φ_3) and (S_7, φ'_3) . Their α -measure and β -measure are:

$$\begin{aligned}\alpha(S_7, \varphi_3) &= 6, & \beta(S_7, \varphi_3) &= 22, \\ \alpha(S_7, \varphi'_3) &= 7, & \beta(S_7, \varphi'_3) &= 28.\end{aligned}$$

For the linear reassembling (S_7, \mathcal{B}_3) induced by both (S_7, φ_3) and (S_7, φ'_3) , we have: $\alpha(S_7, \mathcal{B}_3) = 7$ and $\beta(S_7, \mathcal{B}_3) = 35$, as noted in Example 65. Moreover, while both (S_7, φ_3) and (S_7, φ'_3) are neither α -optimal nor β -optimal, (S_7, \mathcal{B}_3) is α -optimal (though not β -optimal). \square

5.5 α -Optimization of Linear Reassembling Is NP-Hard

We prove that the α -optimality of linear arrangements (in the literature: the *minimum-cutwidth linear arrangement* problem) and the α -optimality of linear reassemblings are reducible to each other in polynomial time.

Definition 69 (*Chordal graph, triangulation, clique number, treewidth*). Let $G = (\mathbf{V}, \mathbf{E})$ be a simple undirected graph. The following are standard notions of graph theory [27].

- G is a *chordal graph* if every cycle of length of 4 or more has a *chord*, i.e., an edge connecting two vertices that are not consecutive in the cycle.
- A *triangulation of G* is a chordal graph $G' = (\mathbf{V}', \mathbf{E}')$ where $\mathbf{V} = \mathbf{V}'$ and $\mathbf{E} \subseteq \mathbf{E}'$. In such a case, we say that G can be *triangulated into G'* , not uniquely in general.
- The *clique number* of G , denoted $\omega(G)$, is the size of a largest clique in G .
- There are different equivalent definitions of the *treewidth*. We here use a definition, or a consequence of the original definition, which is more convenient for our purposes [21, 27].

The *treewidth* of G is:

$$\min \{ \omega(G') \mid G' \text{ is a triangulation of } G \} - 1.$$

In words, among all triangulations G' of G , we choose a G' whose clique number is smallest: The *treewidth* of G is one less than the clique number of such a G' . \square

Lemma 70. *For every positive integers Δ and k , there is an algorithm \mathcal{A} using Δ and k as fixed parameters, such that, given an arbitrary simple undirected graph $G = (\mathbf{V}, \mathbf{E})$ as input to \mathcal{A} , if:*

1. *the maximum vertex degree in G is $\leq \Delta$, and*
2. *the treewidth of G is $\leq k$,*

then \mathcal{A} computes a minimum-cutwidth linear arrangement of G in time $\mathcal{O}(n^{\Delta k^2})$ where $n = |\mathbf{V}|$.

Proof. This is Theorem 4.2 in [93], where the algorithm not only computes the value of a minimum cutwidth, but can be adjusted to output the corresponding minimum-cutwidth linear arrangement $\varphi : \mathbf{V} \rightarrow \{1, \dots, n\}$. \square

In Lemma 71, a *cut vertex* in G is a vertex whose removal increases the number of connected components.

Lemma 71. *There is an algorithm \mathcal{A} such that, given an arbitrary simple undirected graph $G = (\mathbf{V}, \mathbf{E})$ as input to \mathcal{A} , if:*

1. *every vertex in G has degree ≤ 3 , and*
2. *every vertex in G of degree = 3 is a cut vertex,*

then \mathcal{A} computes a minimum-cutwidth linear arrangement of G in time $\mathcal{O}(n^{12})$ where $n = |\mathbf{V}|$.

Proof. We show that the treewidth k of G is ≤ 2 . Because the maximum vertex degree Δ of G is ≤ 3 , Lemma 70 implies the existence of an algorithm \mathcal{A} which runs in time $\mathcal{O}(n^{\Delta k^2}) = \mathcal{O}(n^{12})$.

To show that $k \leq 2$, consider a vertex v of degree = 3, which is therefore a cut vertex. The removal of v can have one of two possible outcomes:

- (a) disconnect G into 3 components, or
- (b) disconnect G into 2 components.

If every vertex of degree = 3 satisfies condition (a), then G is tree whose treewidth is 1, since its clique number $\omega(G) = 2$ in this case.

If C_1 and C_2 are cycles in G , each with 3 vertices or more, then C_1 and C_2 are non-overlapping, *i.e.*, C_1 and C_2 have no vertex in common and no edge in common. If they have an edge $\overline{v_1 w_1}$ in common, then there is an edge $\overline{v_2 w_2} \in C_1 \cap C_2$ such that $\text{degree}(v_2) = 3$ (or, resp., $\text{degree}(w_2) = 3$) and v_2 (or, resp., w_2) is not a cut vertex, contradicting the hypothesis. If C_1 and C_2 have no edge in common, but do have a vertex v in common, then $\text{degree}(v) > 3$, again contradicting the hypothesis.

In case one or more vertices satisfy condition (b), G can be therefore viewed as a finite collection of non-overlapping rings $\{R_1, \dots, R_p\}$, each ring being a cycle with at least 3 vertices, satisfying condition (c):

- (c) if two distinct rings $\{R_i, R_j\}$, with $i \neq j$, are connected by a path $P_{i,j}$, then the removal of all the vertices and edges of $P_{i,j}$ (in particular the two endpoints of $P_{i,j}$, one in R_i and one in R_j , which are necessarily vertices of degree = 3) disconnects G into 2 components.

Another way of expressing (c) is that, if all the rings $\{R_1, \dots, R_p\}$ are each contracted to a single vertex, then the result is a tree (where some of the internal vertices may now have degree larger than 3). Since the clique number of a ring is 3, the treewidth of a ring is 2, and the desired conclusion follows. \square

Lemma 72. *Let $G = (\mathbf{V}, \mathbf{E})$ be a simple undirected graph, where every vertex has degree ≤ 3 , and let (G, \mathcal{L}) be a linear reassembling of G . Consider the longest chain of nested clusters of size ≥ 2 , as in (A) in the opening paragraph of Section 5.3.3:*

$$X_1 \subsetneq X_2 \subsetneq \dots \subsetneq X_{n-1} = \mathbf{V}.$$

Conclusion: *If there is one vertex of degree = 3 in G which is not a cut vertex, then*
 $\max \{ \text{degree}(X_1), \dots, \text{degree}(X_{n-1}) \} \geq 3$.

Proof. We first show there are least two vertices of degree = 3 which are not cut vertices. Let v be a vertex of degree = 3 which is not a cut vertex, and let $\{\overline{vx}, \overline{vy}, \overline{vz}\}$ be the three edges incident to v . Because v is not a cut vertex, any two edges in $\{\overline{vx}, \overline{vy}, \overline{vz}\}$ are consecutive edges in a cycle

containing v . Let $C(v, x, y)$ be a cycle containing edges $\{\overline{vx}, \overline{vy}\}$, and define similarly cycles $C(v, x, z)$ and $C(v, y, z)$. If any of these three cycles contains a chord, then the two endpoints of the chord are vertices of degree = 3 which are not cut vertices. If none of these three cycles contain a chord, then we can combine any two of them, because they share an edge, to form another cycle with a chord, which again implies the existence of two vertices of degree = 3 which are not cut vertices.

To conclude the proof, consider the clusters of \mathcal{L} of size ≥ 2 : These are $\{X_1, \dots, X_{n-1}\}$, and the corresponding singleton clusters are $\{Y_0, \dots, Y_{n-1}\}$, as in (A) and (B) in Section 5.3.3. By the preceding argument, there are at least two vertices of degree = 3 which are not cut vertices. Let one of these two be v , with $Y_i = \{v\}$ for some $i \geq 1$.

We have $X_{i-1} \cap Y_i = \emptyset$ and $X_{i-1} \cup Y_i = X_i$. There are two cases: (1) For some vertex $w \in X_{i-1}$, there is an edge $\overline{vw} \in \mathbf{E}$, and (2) for every vertex $w \in X_{i-1}$, there is no such edge. We consider case (1) and leave the other (easier) case (2) to the reader.

We cannot have $\text{degree}(X_{i-1}) = 0$, otherwise G is disconnected, nor can we have $\text{degree}(X_{i-1}) = 1$, otherwise v is a cut vertex. Hence, $\text{degree}(X_{i-1}) \geq 2$. If $\text{degree}(X_{i-1}) \geq 3$, this is already the conclusion of the lemma and there is nothing else to prove. Suppose $\text{degree}(X_{i-1}) = 2$, the case left to consider.

Similarly, we cannot have $\text{degree}(X_i) = 0$, otherwise G is disconnected, nor $\text{degree}(X_i) = 1$, otherwise v is a cut vertex. Hence, $\text{degree}(X_i) \geq 2$. But if $\text{degree}(X_{i-1}) = 2$ and $\text{degree}(X_i) \geq 2$, with $\text{degree}(v) = 3$, then it must be that $\text{degree}(X_i) = 3$. \square

Lemma 73. *Let $G = (\mathbf{V}, \mathbf{E})$ be a simple undirected graph, where every vertex has degree ≤ 3 and where one vertex of degree = 3 is not a cut vertex. Let (G, \mathcal{L}) be a linear reassembling and (G, φ) a linear arrangement.*

Conclusion: *If (G, \mathcal{L}) is induced by (G, φ) , or if (G, φ) is induced by (G, \mathcal{L}) , then:*

- $\alpha(G, \mathcal{L}) = \alpha(G, \varphi)$.
- (G, \mathcal{L}) is α -optimal iff (G, φ) is α -optimal.

Proof. Straightforward consequence of Lemma 72, the definitions of $\alpha(G, \mathcal{L})$ and $\alpha(G, \varphi)$, and

what it means for (G, \mathcal{L}) to be induced by (G, φ) and for (G, φ) to be induced by (G, \mathcal{L}) . When there is at least one vertex of degree = 3 which is *not* a cut vertex, and therefore at least two of them by the proof of Lemma 72, we can ignore the degrees of singleton clusters in the computation of $\alpha(G, \mathcal{L})$. All details omitted. \square

Theorem 74. *For the class of simple undirected graphs G where every vertex has degree ≤ 3 , the α -optimality of linear arrangements (G, φ) is polynomial-time reducible to the α -optimality of linear reassemblings (G, \mathcal{L}) .*

More explicitly, a polynomial-time algorithm \mathcal{A} , which returns an α -optimal linear reassembling (G, \mathcal{L}) of a graph G where every vertex has degree ≤ 3 , can be used to return an α -optimal linear arrangement (G, φ) .

Proof. Consider an arbitrary G where every vertex has degree ≤ 3 . If every vertex in G of degree = 3 is a cut vertex, we use the algorithm in Lemma 71 to compute an α -optimal linear arrangement (G, φ) in time $\mathcal{O}(n^{12})$. If there is a vertex in G of degree = 3 which is not a cut vertex, we first compute an α -optimal linear reassembling (G, \mathcal{L}) and then return the linear arrangement (G, φ) induced by (G, \mathcal{L}) . The desired conclusion follows from Lemma 73. \square

Corollary 75. *For the class of all simple undirected graphs G , the computation of α -optimal linear reassemblings (G, \mathcal{L}) is an NP-hard problem.*

Proof. If there is a polynomial-time algorithm \mathcal{A} to compute, for an arbitrary simple undirected graph, an α -optimal linear reassembling, then the same algorithm \mathcal{A} can be used to compute in polynomial-time an α -optimal linear reassembling (G, \mathcal{L}) for a graph G where every vertex has degree ≤ 3 . By Theorem 74, \mathcal{A} can be further adapted to compute an α -optimal linear arrangement (G, φ) for such a graph G in polynomial time. But the latter problem (in the literature: the *minimum-cutwidth linear arrangement* problem) is known to be NP-hard [69, 76]. \square

Remark 76. To the best of our knowledge, the complexity status of the *minimum-cutwidth linear arrangement* problem for k -regular graphs for a fixed $k \geq 3$ is an open problem. If it were known to be NP-hard, we would be able to simplify our proof of Theorem 74 and its corollary considerably.

In particular, we would be able to eliminate Lemmas 70 and 71 and the supporting Definition 69, as well as simplify Lemmas 72 and 73 by restricting them to k -regular graphs. \square

Theorem 74 and Corollary 75 together say the α -optimality of linear arrangements (G, φ) is polynomial-time reducible to the α -optimality of linear reassemblings (G, \mathcal{L}) . For completeness, we show the converse in the next theorem.

Theorem 77. *For the class of simple undirected graphs G in general, the α -optimality of linear reassemblings (G, \mathcal{L}) is polynomial-time reducible to the α -optimality of linear arrangements (G, φ) .*

Proof. In Section 5.9. \square

5.6 β -Optimization of Linear Reassembling Is NP-Hard

We prove that the β -optimality of linear arrangements (in the literature: the *minimum-cost linear arrangement* problem or also the *optimal linear arrangement* problem) and the β -optimality of linear reassemblings are reducible to each other in polynomial time. Towards this end, we prove an intermediate result, which is also of independent interest (Theorem 83 which presupposes Definition 78).

Definition 78 (*Anchored linear reassemblings*). Let $G = (\mathbf{V}, \mathbf{E})$ be a simple undirected graph and let $w \in \mathbf{V}$. Let (G, \mathcal{L}) be a linear reassembling of G , whose longest chain of nested clusters of size ≥ 2 , as in (A) in the opening paragraph of Section 5.3.3, is:

$$X_1 \subsetneq X_2 \subsetneq \cdots \subsetneq X_{n-1} = V$$

and whose corresponding singleton clusters are $\{Y_0, \dots, Y_{n-1}\}$, as determined by (B) in the opening paragraph of Section 5.3.3. We say (G, \mathcal{L}) is a *linear reassembling anchored at $w \in \mathbf{V}$* iff there is a vertex $w' \in \mathbf{V}$ such that:

$$Y_0 = \{w\}, \quad Y_1 = \{w'\}, \quad \text{and } \text{degree}_G(w) \leq \text{degree}_G(w').$$

Note that we require that the immediate sibling $Y_1 = \{w'\}$ of the leaf node $Y_0 = \{w\}$ satisfy the condition $\text{degree}_G(w) \leq \text{degree}_G(w')$. This implies that, given an arbitrary vertex $w \in \mathbf{V}$, we cannot anchor a linear reassembling at w unless we find another vertex $w' \in \mathbf{V}$ such that $\text{degree}_G(w) \leq \text{degree}_G(w')$ and then make $\{w\}$ and $\{w'\}$ sibling leaf-nodes. This is a technical restriction to simplify the statement of Lemma 81.⁶ \square

Definition 79 (*Anchored linear arrangements*). Let $G = (\mathbf{V}, \mathbf{E})$ be a simple undirected graph and let $w \in \mathbf{V}$. Let (G, φ) be a linear arrangement of G . We say (G, φ) is a *linear arrangement anchored at $w \in \mathbf{V}$* iff there is a vertex $w' \in \mathbf{V}$ such that:

$$\varphi(w) = 1, \quad \varphi(w') = 2, \quad \text{and } \text{degree}_G(w) \leq \text{degree}_G(w').$$

Again, as in Definition 78, the condition $\text{degree}_G(w) \leq \text{degree}_G(w')$ is imposed in order to simplify the statement of Lemma 81. It is worth noting that, if we relax this condition and allow $\text{degree}_G(w) > \text{degree}_G(w')$, then the new arrangement φ' which permutes the positions of w and w' , i.e.:

$$\varphi'(w') = 1, \quad \varphi'(w) = 2, \quad \text{and } \varphi'(v) = \varphi(v) \text{ for all } v \in \mathbf{V} - \{w, w'\},$$

is such that $\beta(G, \varphi') < \beta(G, \varphi)$. In words, if we allowed $\text{degree}_G(w) > \text{degree}_G(w')$, the linear arrangement (G, φ) would not be β -optimal.⁷ \square

Example 80. Consider the linear reassemblings (S_7, \mathcal{B}_3) and (S_7, \mathcal{B}_5) in Example 65. (S_7, \mathcal{B}_3) is anchored at vertex “2”, but cannot be anchored at vertex “1”, while (S_7, \mathcal{B}_5) is anchored at vertex “2”, and can be anchored again at vertex “3”. Both (S_7, \mathcal{B}_3) and (S_7, \mathcal{B}_5) are α -optimal and, a fortiori, α -optimal for the class of all linear reassemblings of S_7 anchored at vertex “2”. Moreover, $\beta(S_7, \mathcal{B}_3) = 35$ and $\beta(S_7, \mathcal{B}_5) = 29$, so that (S_7, \mathcal{B}_3) is not β -optimal for the class of all linear reassemblings of S_7 anchored at vertex “2”, while (S_7, \mathcal{B}_5) is β -optimal for the same class.

⁶Thus, we cannot say that the linear reassembling (S_7, \mathcal{B}_3) , in Examples 65 and 66, is anchored at vertex “1”, though we can say that (S_7, \mathcal{B}_3) is anchored at vertex “2”. More generally, in the case of a star graph S_k with $k \geq 3$ leaves: There is no linear reassembling of S_k anchored at the internal vertex of S_k .

⁷A similar statement applies to the α -measure: If we allowed $\text{degree}_G(w) > \text{degree}_G(w')$, then the new arrangement φ' would be such that $\alpha(G, \varphi') \leq \alpha(G, \varphi)$, but not necessarily $\alpha(G, \varphi') < \alpha(G, \varphi)$.

Consider now the linear arrangements (S_7, φ_3) and (S_7, φ_5) induced by the linear reassemblings (S_7, \mathcal{B}_3) and (S_7, \mathcal{B}_5) , respectively. φ_3 and φ_5 are given in Example 66 and Example 67. Both (S_7, φ_3) and (S_7, φ_5) are anchored at vertex “2”. Moreover, (S_7, φ_3) cannot be anchored at vertex “1” (the sibling leaf of “2” in φ_3), while (S_7, φ_5) can be anchored again at vertex “3” (the sibling leaf of “2” in φ_5).

(S_7, φ_3) is neither α -optimal nor β -optimal for the class of all linear arrangements of S_7 anchored at “2”; hence, (S_7, φ_3) is neither α -optimal nor β -optimal for the super-class of all linear arrangements of S_7 . By contrast, (S_7, φ_5) is both α -optimal and β -optimal for the class of all linear arrangements of S_7 ; hence, (S_7, φ_5) is both α -optimal and β -optimal for the sub-class of all linear arrangements of S_7 anchored at “2”. \square

Let (G, \mathcal{L}) be a linear reassembling anchored at vertex $w \in \mathbf{V}$. We say (G, \mathcal{L}) is β -optimal relative to anchor w iff:

$$\beta(G, \mathcal{L}) = \min \{ \beta(G, \mathcal{L}') \mid (G, \mathcal{L}') \text{ is a linear reassembling anchored at } w \}.$$

Clearly, (G, \mathcal{L}) is a β -optimal linear reassembling, with no anchor restriction, iff:

$$\beta(G, \mathcal{L}) = \min \{ \beta(G, \mathcal{L}') \mid \text{there is a vertex } w \in \mathbf{V} \text{ and} \\ (G, \mathcal{L}') \text{ is a linear reassembling } \beta\text{-optimal relative to anchor } w \}.$$

Similar obvious conditions apply to what it means for (G, φ) to be a β -optimal linear arrangement relative to anchor w .

Lemma 81. *Let $G = (\mathbf{V}, \mathbf{E})$ be a simple undirected graph and $w \in \mathbf{V}$. Let (G, \mathcal{L}) be a linear reassembling of G anchored at w , and (G, φ) be a linear arrangement of G anchored at w , such that:*

$$(G, \varphi) \text{ is induced by } (G, \mathcal{L}) \quad \text{or} \quad (G, \mathcal{L}) \text{ is induced by } (G, \varphi).$$

Conclusion: (G, \mathcal{L}) is β -optimal relative to anchor w iff (G, φ) is β -optimal relative to anchor w .

Proof. Let $d = \text{degree}(w) \geq 1$ and $\Delta = \sum \{ \text{degree}(v) \mid v \in \mathbf{V} \text{ and } v \neq w \}$. Consider the case when arrangement (G, φ) is induced by reassembling (G, \mathcal{L}) . (We omit the case when reassembling (G, \mathcal{L}) is induced by arrangement (G, φ) , which is treated similarly.) From Definition 57,

$$\beta(G, \mathcal{L}) = d + \Delta + \sum \{ \text{degree}(X_i) \mid 1 \leq i \leq n-1 \}$$

where X_1, \dots, X_{n-1} are all the clusters of size ≥ 2 in \mathcal{L} . From Definitions 58 and 61,

$$\beta(G, \varphi) = d + \sum \{ \text{degree}(X_i) \mid 1 \leq i \leq n-1 \}.$$

Hence, both $\beta(G, \mathcal{L})$ and $\beta(G, \varphi)$ are minimized when the same quantity $\sum \{ \text{degree}(X_i) \mid 1 \leq i \leq n-1 \}$ is minimized. The desired conclusion follows. \square

Remark 82. There is an obvious definition of *anchored α -optimality*, similar to that of *anchored β -optimality* above. However, results for the latter do not necessarily have counterparts for the former. In particular, the conclusion of Lemma 81 does not hold for α -optimality. Specifically, there are simple counter-examples showing the existence of a simple undirected graph $G(\mathbf{V}, \mathbf{E})$ with a distinguished vertex $w \in \mathbf{V}$ such that:

- there is a linear reassembling (G, \mathcal{L}) which is α -optimal relative to anchor w ,
- but the linear arrangement (G, φ) induced by (G, \mathcal{L}) is not α -optimal relative to anchor w .

Such a counter-example is the linear reassembling (S_7, \mathcal{B}_3) and the linear arrangement (S_7, φ_3) it induces, in Example 80, both anchored at vertex “2”: the former is α -optimal for the class of all linear reassemblings of S_7 anchored at “2”, the latter is not α -optimal for the class of all linear arrangements of S_7 anchored at “2”.

There is an examination, yet to be undertaken, of the relation between linear reassemblings (G, \mathcal{L}) and linear arrangements (G, φ) that are α -optimal relative to the same anchor, similar to our study of anchored β -optimality below. This examination we do not pursue in this report. \square

Theorem 83. *For the class of all simple undirected graphs $G = (\mathbf{V}, \mathbf{E})$, each with a distinguished vertex $w \in \mathbf{V}$, the two following problems are polynomial-time reducible to each other:*

- the β -optimality of linear arrangements (G, φ) anchored at w ,
- the β -optimality of linear reassemblings (G, \mathcal{L}) anchored at w .

More explicitly, a polynomial-time algorithm \mathcal{A} , which returns a linear reassembling (G, \mathcal{L}) [resp. a linear arrangement (G, φ)] which is β -optimal relative to anchor w can be used to return in polynomial time a linear arrangement (G, φ) [resp. a linear reassembling (G, \mathcal{L})] which is β -optimal relative to anchor w .

Proof. This is an immediate consequence of Lemma 81. \square

Definition 84 (*Auxiliary graphs*). Let $G = (\mathbf{V}, \mathbf{E})$ be a simple undirected graph, with $|\mathbf{V}| = n$ and $|\mathbf{E}| = m$. For every $w \in \mathbf{V}$ we define what we call an *auxiliary graph* $G_w = (\mathbf{V}_w, \mathbf{E}_w)$ as follows:

- $\mathbf{V}_w := \mathbf{V} \uplus U$ where U is a fresh set of $p = \sum \{ \text{degree}_G(v) \mid v \in \mathbf{V} \}$ vertices.
- $\mathbf{E}_w := \mathbf{E} \uplus D_w$ where $D_w := \{ \overline{uw} \mid u \in U \} \cup \{ \overline{u_1 u_2} \mid u_1, u_2 \in U \text{ and } u_1 \neq u_2 \}$.

Thus, the subgraph of G_w induced by the set \mathbf{V} is simply the original graph G , and the subgraph of G_w induced by the set $U \cup \{w\}$ is the complete graph K_{p+1} over $p+1$ vertices.

Informally, G_w is constructed from G and the complete graph K_{p+1} by identifying vertex $w \in \mathbf{V}$ with one of the vertices of K_{p+1} . In particular, w is a cut vertex of the auxiliary graph G_w . We call w , which is the common vertex of G and K_{p+1} , the *distinguished vertex* of G_w . \square

Lemma 85. *If $G_w = (\mathbf{V}_w, \mathbf{E}_w)$ is the auxiliary graph for vertex $w \in \mathbf{V}$, as constructed in Definition 84, then $|\mathbf{V}_w| \leq n^2$ and $|\mathbf{E}_w| \leq (n^4 - 2n^3 + 3n^2 - 2n)/2$.*

Proof. The number m of edges in G is bounded by $(n^2 - n)/2$. Hence, $p = \sum \{ \text{degree}(v) \mid v \in \mathbf{V} \} \leq (n^2 - n)$, implying that the total number of vertices $p+n$ in G_w is $\leq (n^2 - n) + n = n^2$. The number of edges in K_p is $(p^2 - p)/2$, and in K_{p+1} it is $(p^2 + p)/2$, which is $\leq ((n^2 - n)^2 + (n^2 - n))/2 = (n^4 - 2n^3 + 2n^2 - n)/2$. Hence, the total number of edges in G_w is $m + (p^2 + p)/2 \leq (n^4 - 2n^3 + 3n^2 - 2n)/2$. \square

Let \mathcal{L} be a linear binary tree over \mathbf{V} where, as in (A) in the opening paragraph of Section 5.3.3,

the longest chain of nested clusters of size ≥ 2 is:

$$X_1 \subsetneq X_2 \subsetneq \cdots \subsetneq X_{n-1} = \mathbf{V},$$

and let the corresponding singleton clusters be $\{Y_0, \dots, Y_{n-1}\}$ as determined by (B). The linear tree \mathcal{L} is uniquely determined by a sequence of vertices written in the form:

$$[v_1 \cdots v_n]$$

where $Y_0 = \{v_1\}, Y_1 = \{v_2\}, \dots, Y_{n-1} = \{v_n\}$. We say $[v_1 \cdots v_n]$ is the *vertex sequence induced* by \mathcal{L} , and \mathcal{L} the *linear reassembling* (or the *linear binary tree*) induced by the vertex sequence $[v_1 \cdots v_n]$.

Similarly, if $\varphi : \mathbf{V} \rightarrow \{1, \dots, n\}$ is a linear arrangement of \mathbf{V} , then φ is uniquely determined by a sequence of vertices in the same form:

$$[v_1 \cdots v_n]$$

where $\varphi^{-1}(1) = v_1, \varphi^{-1}(2) = v_2, \dots, \varphi^{-1}(n) = v_n$. We say $[v_1 \cdots v_n]$ is the *vertex sequence induced* by φ , and φ the *linear arrangement induced* by the vertex sequence $[v_1 \cdots v_n]$.

For the auxiliary graph G_w , whether we deal with a linear reassembling (G_w, \mathcal{L}) or a linear arrangement (G_w, φ) , it is convenient to consider sequences of the following form, which interleaves vertices and cutwidths:

$$\mathcal{S} := [x_1 \quad (r_1, s_1) \quad x_2 \quad (r_2, s_2) \quad \cdots \quad \cdots \quad x_{n+p-1} \quad (r_{n+p-1}, s_{n+p-1}) \quad x_{n+p}] \quad (\diamond)$$

where $\{x_1, \dots, x_{n+p}\} = \mathbf{V}_w = \{v_1, \dots, v_n\} \cup \{u_1, \dots, u_p\}$, and for every $1 \leq i \leq n+p-1$:

$$r_i := \text{degree}_G(\{x_1, \dots, x_i\}) \quad \text{and} \quad s_i := \text{degree}_{K_{p+1}}(\{x_1, \dots, x_i\}).$$

We say the sequence \mathcal{S} in (\diamond) is the *sequence of vertices and cutwidths induced* by (G_w, \mathcal{L}) or by (G_w, φ) , whichever of the two is the case. The measure β on \mathcal{S} is:

$$\beta(\mathcal{S}) := \sum_{1 \leq i \leq n+p-1} (r_i + s_i).$$

Lemma 86. *Consider the sequence of vertices and cutwidths induced by (G_w, \mathcal{L}) or by (G_w, φ) , as just defined. **Conclusion:***

- For every $1 \leq i \leq n + p - 1$, it holds that $r_i + s_i = \text{degree}_{G_w}(\{x_1, \dots, x_i\})$.
- If the sequence in (\diamond) is induced by the linear arrangement (G_w, φ) , then

$$\beta(G_w, \varphi) = \beta(\mathcal{S}) = \sum_{1 \leq i \leq n+p-1} (r_i + s_i).$$

- If the sequence in (\diamond) is induced by the linear reassembling (G_w, \mathcal{L}) , then

$$\beta(G_w, \mathcal{L}) = \Delta + \beta(\mathcal{S}) = \Delta + \sum_{1 \leq i \leq n+p-1} (r_i + s_i),$$

where $\Delta = \sum \{\text{degree}_{G_w}(v) \mid v \in \mathbf{V}_w \text{ and } v \neq x_1\}$.

Proof. Straightforward consequence of the definitions. All details omitted. □

We say that the sequence \mathcal{S} is *scattered* if the vertices of K_{p+1} do not occur consecutively, i.e., the vertices of K_{p+1} are interspersed with vertices of $\mathbf{V} - \{w\}$.

Lemma 87. *Let $G_w = (\mathbf{V}_w, \mathbf{E}_w)$ be the auxiliary graph for vertex $w \in \mathbf{V}$, as constructed in Definition 84. Let \mathcal{S} be the sequence of vertices and cutwidths, as in (\diamond) , induced by a β -optimal linear reassembling (G_w, \mathcal{L}) or by a β -optimal linear arrangement (G_w, φ) . **Conclusion:** \mathcal{S} is not scattered.*

In words, in a β -optimal linear reassembling (G_w, \mathcal{L}) [or in a β -optimal linear arrangement (G_w, φ) , resp.] all the vertices of K_{p+1} are reassembled consecutively [or arranged consecutively, resp.] without intervening vertices from $\mathbf{V} - \{w\}$.

Proof. In Section 5.9. □

Consider again the sequence \mathcal{S} of vertices and cutwidths in (\diamond) . Suppose \mathcal{S} is not scattered. This means that the $p + 1$ vertices of K_{p+1} occur consecutively in \mathcal{S} . We say \mathcal{S} is *balanced* iff one of two conditions holds:

- $$(1) \quad \{x_1, \dots, x_{n-1}\} = \mathbf{V} - \{w\}, \quad \{x_n\} = \{w\}, \quad \{x_{n+1}, \dots, x_{n+p}\} = U,$$
- $$(2) \quad \{x_1, \dots, x_p\} = U, \quad \{x_{p+1}\} = \{w\}, \quad \{x_{p+2}, \dots, x_{n+p}\} = \mathbf{V} - \{w\}.$$

In words, \mathcal{S} is balanced if all the vertices of $\mathbf{V} - \{w\}$ are on the same side (on the left in (1), or on the right in (2)) of the distinguished vertex w and all the vertices of U are on the other side (on the right in (1), or on the left in (2)) of w . Put differently still, \mathcal{S} is balanced if all the vertices of $\mathbf{V} - \{w\}$ are together, all the vertices of U are together, and w is between the two sets of vertices. The following is a refinement of the preceding lemma and its proof is based on a similar argument.

Lemma 88. *Let $G_w = (\mathbf{V}_w, \mathbf{E}_w)$ be the auxiliary graph for vertex $w \in \mathbf{V}$, as constructed in Definition 84. Let \mathcal{S} be the sequence of vertices and cutwidths, as in (\diamond) , induced by a β -optimal linear reassembling (G_w, \mathcal{L}) or by a β -optimal linear arrangement (G_w, φ) . **Conclusion:** \mathcal{S} is balanced.*

Proof. In Section 5.9. □

By the preceding lemma, if the sequence \mathcal{S} in (\diamond) is induced by a β -optimal linear reassembling (G_w, \mathcal{L}) , or by a β -optimal linear arrangement (G_w, φ) , then \mathcal{S} is balanced, either on the left or on the right. For the rest of the analysis below, we assume that \mathcal{S} is balanced on the right, *i.e.*, all the vertices in U occur first, then w , and then all the vertices of $\mathbf{V} - \{w\}$.

Definition 89 (*Restrictions of linear reassemblings and linear arrangements*). Let \mathcal{L} be a linear binary tree over the set \mathbf{V} . If $\mathbf{V}' \subseteq \mathbf{V}$, the *restriction* of \mathcal{L} to \mathbf{V}' , denoted $(\mathcal{L} | \mathbf{V}')$, consists of the following clusters:

$$(\mathcal{L} | \mathbf{V}') := \{X \cap \mathbf{V}' \mid X \in \mathcal{L}\}.$$

It is a straightforward exercise to show that $(\mathcal{L} \mid \mathbf{V}')$ is a linear binary tree over \mathbf{V}' .

Let $\varphi : \mathbf{V} \rightarrow \{1, \dots, n\}$ be a linear arrangement of \mathbf{V} . The *restriction* of φ to \mathbf{V}' , denoted $(\varphi \mid \mathbf{V}')$, is defined as follows. For every $1 \leq i \leq n' = |\mathbf{V}'|$, let:

$$(\varphi \mid \mathbf{V}')(v) := i \quad \text{where } v = \varphi^{-1}(j) \text{ and } j \in \{1, \dots, n\} \text{ is} \\ \text{the largest integer such that } |\{\varphi^{-1}(1), \dots, \varphi^{-1}(j-1)\} \cap \mathbf{V}'| = i-1.$$

Again here, it is straightforward to show that $(\varphi \mid \mathbf{V}')$ is a linear arrangement of \mathbf{V}' such that:

$$(\varphi \mid \mathbf{V}')^{-1}(1), \dots, (\varphi \mid \mathbf{V}')^{-1}(n') \quad \text{is a subsequence of } \varphi^{-1}(1), \dots, \varphi^{-1}(n).$$

Moreover, if (G, \mathcal{L}) is a linear reassembling [resp. (G, φ) is a linear arrangement] of the simple undirected graph $G = (\mathbf{V}, \mathbf{E})$ and $G' = (\mathbf{V}', \mathbf{E}')$ is the subgraph of G induced by $\mathbf{V}' \subseteq \mathbf{V}$, then $(G', (\mathcal{L} \mid \mathbf{V}'))$ is a linear reassembling [resp. $(G', (\varphi \mid \mathbf{V}'))$ is a linear arrangement] of G' . \square

Lemma 90. *Let $G_w = (\mathbf{V}_w, \mathbf{E}_w)$ be the auxiliary graph for vertex $w \in \mathbf{V}$, as constructed in Definition 84.*

1. *If (G_w, \mathcal{L}) is a β -optimal linear reassembling of G_w with no anchor restriction, then $(G, (\mathcal{L} \mid \mathbf{V}))$ is a β -optimal linear reassembling relative to anchor w .*
2. *If (G_w, φ) is a β -optimal linear arrangement of G_w with no anchor restriction, then $(G, (\varphi \mid \mathbf{V}))$ is a β -optimal linear arrangement relative to anchor w .*

Proof. We prove part 1 only, the proof of part 2 is similar. By Lemma 88, the sequence \mathcal{S} induced by a β -optimal linear reassembling (G_w, \mathcal{L}) is balanced. By our assumption preceding Definition 78, we take \mathcal{S} to be balanced on the right, i.e., all the vertices in U occur first, then w , and then all the vertices of $\mathbf{V} - \{w\}$. There are no edges connecting vertices in U on the left to vertices in $\mathbf{V} - \{w\}$ on the right, with w a cut vertex in the middle. The β -optimality of (G_w, \mathcal{L}) implies the β -optimality of the linear reassembling $(G, (\mathcal{L} \mid \mathbf{V}))$ of the subgraph $G = (\mathbf{V}, \mathbf{E})$ of $G_w = (\mathbf{V}_w, \mathbf{E}_w)$. We omit all formal details. \square

Theorem 91. *For the class of all simple undirected graphs G , the two following problems are polynomial-time reducible to each other:*

- *the β -optimality of linear arrangements (G, φ) ,*
- *the β -optimality of linear reassemblings (G, \mathcal{L}) .*

More explicitly, a polynomial-time algorithm \mathcal{A} , which returns a β -optimal linear reassembling (G, \mathcal{L}) [resp. a β -optimal linear arrangement (G, φ)] of an arbitrary graph G , can be used to return a β -optimal linear arrangement (G, φ) [resp. a β -optimal linear reassembling (G, \mathcal{L})] in polynomial time.

Proof. We compute a β -optimal linear reassembling (G_{v_i}, \mathcal{L}_i) [resp. a β -optimal linear arrangement (G_{v_i}, φ_i)] of the auxiliary graph G_{v_i} , one for each vertex $v_i \in \mathbf{V} = \{v_1, \dots, v_n\}$. We next consider the linear reassembling $(G, (\mathcal{L}_i | \mathbf{V}))$ [resp. the linear arrangement $(G, (\varphi_i | \mathbf{V}))$] which, by Lemma 90, is a β -optimal linear reassembling relative to anchor v_i [resp. a β -optimal linear arrangement relative to anchor v_i], for every $1 \leq i \leq n$. Let (G, φ_i) be the linear arrangement induced by the linear reassembling $(G, (\mathcal{L}_i | \mathbf{V}))$ [resp. let (G, \mathcal{L}_i) be the linear reassembling induced by the linear arrangement $(G, (\varphi_i | \mathbf{V}))$]. By Lemma 81, (G, φ_i) is a β -optimal linear arrangement relative to anchor v_i [resp. (G, \mathcal{L}_i) is a β -optimal linear reassembling relative to anchor v_i], for every $1 \leq i \leq n$. Among these n linear arrangements [resp. n linear reassemblings], we choose one such that $\beta(G, \varphi_i)$ is minimized [resp. $\beta(G, \mathcal{L}_i)$ is minimized]. \square

Corollary 92. *For the class of all simple undirected graphs G , the computation of β -optimal linear reassemblings (G, \mathcal{L}) is an NP-hard problem.*

Proof. This follows from the NP-hardness of the *minimum-cost linear arrangement* problem (also called the *optimal linear arrangement* problem in the literature) [36]. This problem is the same as what we call, in this report, the problem of computing a β -optimal linear arrangement. \square

Remark 93. To the best of our knowledge, the complexity status of the *minimum-cost linear arrangement* problem (or *optimal linear arrangement* problem) for k -regular graphs for a fixed

$k \geq 3$ is an open problem. If it were known to be NP-hard, we would be able to simplify our proof of Theorem 91 and its corollary considerably. \square

5.7 α -Optimization of Balanced Reassembling Is NP-Hard

Consider an arbitrary graph $G = (\mathbf{V}, \mathbf{E})$. Let p be the smallest integer such that $2^p \geq |\mathbf{V}| = n \geq 2$. It follows that $n + r = 2^p$ for some even integer r such that $0 \leq r < n$. Another way of identifying r is to say it is the smallest even integer such $n + r$ is a power of 2. Our standing assumption is that n is even.

Let $q = (n/2) + r$. Following standard notation, K_q denotes the complete graph over q vertices. For Lemma 94, we construct an augmented graph \mathbb{G} consisting of the original G together with two disjoint copies of K_q , which we denote by the separate letters H and I for clarity. More precisely,

$$\begin{aligned} V(\mathbb{G}) &= V(G) \uplus V(H) \uplus V(I), \\ E(\mathbb{G}) &= E(G) \uplus E(H) \uplus E(I) \uplus \{ \overline{vw} \mid v \in V(G) \text{ and } w \in V(H) \cup V(I) \}, \end{aligned}$$

where we write “ \uplus ” for “disjoint union”. We thus assemble the new \mathbb{G} by connecting every vertex in G with all the vertices in H and I . There are no edges between H and I . There is a total of $n + q + q = 2n + 2r = 2^{p+1}$ vertices in \mathbb{G} . Thus, the graph \mathbb{G} has between $2n$ and $3n$ vertices and is a member of the class $\text{GRAPHS}(2^*)$.

Lemma 94. *Consider the graph \mathbb{G} as defined in the preceding paragraph. Every minimum bisection of \mathbb{G} must be of type $(V(H), V(I))$ – see Definition 50 – schematically shown in Figure 5.5a.*

Proof. We consider a bisection $\{A, B\}$ of \mathbb{G} which is *not* of type $(V(H), V(I))$, and then show that it cannot be a minimum bisection.

Because $\{A, B\}$ is a bisection, we have $|A| = |B| = n + r$. And because $|V(H)| = |V(I)| = (n/2) + r$, it is never the case that $A \cap (V(H) \uplus V(I)) = \emptyset$ or $B \cap (V(H) \uplus V(I)) = \emptyset$. Hence, there are two possible cases, one shown in Figure 5.5b and one in Figure 5.5c. In Figure 5.5b, the vertices of both H and I appear on both sides of the bisection. In Figure 5.5c, the vertices of H are

all on the same side of the bisection, while the vertices of I appear on both sides of the bisection.

In these figures, $\{X, X', Y, Y', Z, Z'\}$ is a 6-block partition of $V(\mathbb{G})$, defined by:

$$\begin{aligned} X &= A \cap V(G) & \text{and} & & X' &= B \cap V(G) & & \text{(the vertices of subgraph } G), \\ Y &= A \cap V(H) & \text{and} & & Y' &= B \cap V(H) & & \text{(the vertices of subgraph } H), \\ Z &= A \cap V(I) & \text{and} & & Z' &= B \cap V(I) & & \text{(the vertices of subgraph } I). \end{aligned}$$

In each of the two cases shown in Figures 5.5b and 5.5c, we need to prove that $|\partial(A, B)|$ can be decreased by moving an appropriate number of vertices of the subgraphs H and I from one side of the bisection to the other side.

Case 1. This is the case in Figure 5.5b. With no loss of generality, suppose:

$$|X| \geq |X'|.$$

Because $|A| = |B|$, this forces the inequality:

$$|Y| + |Z| \leq |Y'| + |Z'|.$$

Because $|Y| + |Y'| = |Z| + |Z'|$, this in turn forces one or both of the following inequalities:

$$|Y| \leq |Z'| \quad \text{or} \quad |Z| \leq |Y'|. \quad (\dagger)$$

With no loss of generality, assume the first inequality $|Y| \leq |Z'|$ in (\dagger) holds.

Let $|Y| = k \geq 1$. Select an arbitrary subset $U \subseteq Z'$ such that $|U| = k$. We define a new bisection $\{\tilde{A}, \tilde{B}\}$ of \mathbb{G} by moving: (1) all the vertices of Y from the A -side to the B -side, and (2) all the vertices of U from the B -side to the A -side. Specifically, let:

$$\tilde{A} = (A - Y) \cup U \quad \text{and} \quad \tilde{B} = (B - U) \cup Y.$$

The resulting set of edges connecting \tilde{A} and \tilde{B} is:

$$\begin{aligned} \partial(\tilde{A}, \tilde{B}) = & \left(\partial(A, B) - \left(\partial(Y, X') \cup \partial(Y, Y') \cup \partial(U, X) \cup \partial(U, Z) \right) \right) \\ & \cup \left(\partial(Y, X) \cup \partial(U, X') \cup \partial(U, Z' - U) \right). \end{aligned}$$

Because $|\partial(U, X)| = |\partial(Y, X)|$ and $|\partial(U, X')| = |\partial(Y, X')|$, it follows:

$$|\partial(\tilde{A}, \tilde{B})| = |\partial(A, B)| - |\partial(Y, Y')| - |\partial(U, Z)| + |\partial(U, Z' - U)|$$

With the fact that $|Y| = |U|$, the following inequality must hold:

$$|\partial(Y, Y')| \geq |\partial(U, Z' - U)|$$

otherwise, if it did not, we would have that $|Y'| < |Z' - U| = |Z'| - |U| = |Z'| - |Y|$, in turn implying that $|Y| + |Y'| < |Z'|$, which is a contradiction. Hence,

$$|\partial(\tilde{A}, \tilde{B})| \geq |\partial(A, B)| - |\partial(U, Z)| > |\partial(A, B)|.$$

We conclude that $|\partial(\tilde{A}, \tilde{B})| < |\partial(A, B)|$.

Case 2. This is the case in Figure 5.5c. It cannot be that $|X| < |X'|$, because if it were so, it would imply $|X| < (n/2)$ which, with the fact that $|Z \cup Z'| = (n/2) + r$, would in turn imply that $|A| < n + r$, thus contradicting the hypothesis that $\{A, B\}$ is a bisection of \mathbb{G} . Hence, it must be that:

$$|X| \geq |X'| \geq n/2.$$

The largest possible size of X , which is $n-1$, corresponds to the smallest possible size of Z which, because $|X| + |Z| = n + r$, must therefore be $r + 1$. Corresponding to the smallest possible size of Z is the largest possible size of Z' , which is therefore $|V(I)| - (r + 1) = (n/2) - 1$. Hence, it is always the case that $|Z'| < |X|$.

We now proceed in a way similar to **Case 1**. Let $|Z'| = k \geq 1$. Select an arbitrary subset $U \subseteq X$ such that $|U| = k$ and $\partial(U, X') \neq \emptyset$. The new bisection $\{\tilde{A}, \tilde{B}\}$ of \mathbb{G} is obtained by moving: (1) all the vertices of Z' from the B -side to the A -side, and (2) all the vertices of U from the A -side to the B -side. Specifically, let:

$$\tilde{A} = (A - U) \cup Z' \quad \text{and} \quad \tilde{B} = (B - Z') \cup U.$$

The resulting set of edges connecting \tilde{A} and \tilde{B} is:

$$\begin{aligned} \partial(\tilde{A}, \tilde{B}) = & \left(\partial(A, B) - \left(\partial(Z', Z) \cup \partial(Z', X - U) \cup \partial(Y', U) \cup \partial(X', U) \right) \right) \\ & \cup \left(\partial(Z', X') \cup \partial(Z', U) \cup \partial(Z, U) \cup \partial(X - U, U) \right). \end{aligned}$$

Because $|\partial(Y', U)| = |\partial(Z \cup Z', U)| = |\partial(Z, U)| + |\partial(Z', U)|$, it follows that:

$$\begin{aligned} |\partial(\tilde{A}, \tilde{B})| = & |\partial(A, B)| - |\partial(Z', Z)| - |\partial(Z', X - U)| - |\partial(X', U)| \\ & + |\partial(Z', X')| + |\partial(X - U, U)|. \end{aligned}$$

Because $|\partial(Z', X - U)| \geq |\partial(X - U, U)|$, we obtain the inequality:

$$|\partial(\tilde{A}, \tilde{B})| \leq |\partial(A, B)| - |\partial(Z', Z)| - |\partial(X', U)| + |\partial(Z', X')|.$$

Because $|\partial(Z', Z)| \geq |\partial(Z', X')|$, it follows that:

$$|\partial(\tilde{A}, \tilde{B})| \leq |\partial(A, B)| - |\partial(X', U)|.$$

Because $|\partial(X', U)| \neq 0$, we conclude that $|\partial(\tilde{A}, \tilde{B})| < |\partial(A, B)|$. □

Lemma 95. *MinBisection(2*) is an NP-hard problem.*

Proof. We use the same notation as in the proof of Lemma 94. Given that subgraphs H and I of \mathbb{G} have an equal number $q = (n/2) + r$ of vertices, Lemma 94 implies we can reduce, in

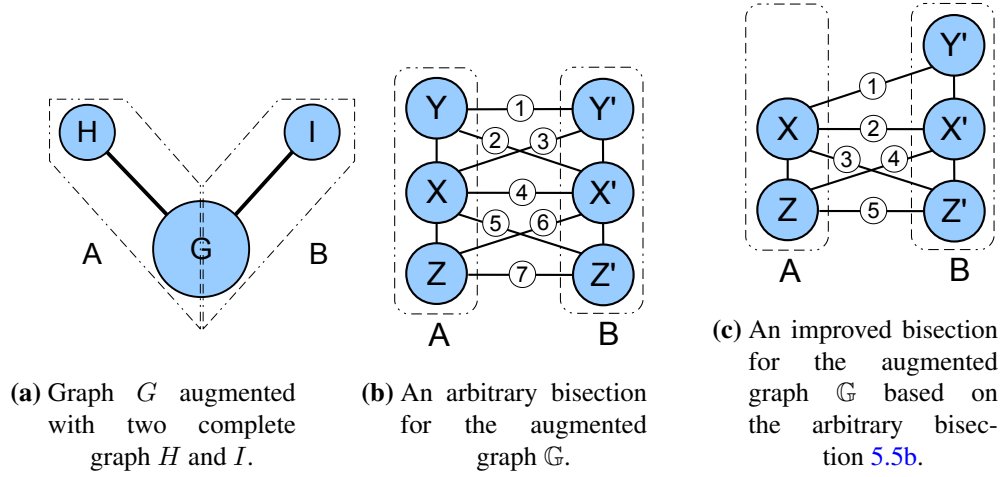


Figure 5.5: For the statement of Lemma 94.

polynomial time, MinBisection for an arbitrary graph G to $\text{MinBisection}(2^*)$ for graph \mathbb{G} . Hence, a minimum bisection for \mathbb{G} induces a minimum bisection for the given G . Hence, the NP-hardness of MinBisection in general implies the NP-hardness of $\text{MinBisection}(2^*)$. \square

Lemma 96. *Let $G = (V, E)$ be a graph in the class $\text{GRAPHS}(2^*)$ and let \mathbb{G} be the augmented graph of G as in Lemma 94. Let $(\mathbb{G}, \mathcal{B})$ be a balanced reassembling of \mathbb{G} where \mathcal{B} is a binary tree over $V(\mathbb{G})$ (see Definition 52). Let A and B be the two children nodes of the root node $V(\mathbb{G})$ in \mathcal{B} .*

Conclusion: *If $(\mathbb{G}, \mathcal{B})$ is an α -optimal balanced reassembling, then $\{A, B\}$ is a minimum bisection of \mathbb{G} .*

Proof. Because $G \in \text{GRAPHS}(2^*)$, each of the complete graphs H and I has $(n/2)$ vertices, i.e., in contrast to the proof of Lemma 94, here $r = 0$. The augmented graph \mathbb{G} is also in the class $\text{GRAPHS}(2^*)$, with $|V(G)| = n$ and $|V(\mathbb{G})| = 2n$. In the given balanced reassembling $(\mathbb{G}, \mathcal{B})$, we have that $\{A, B\}$ is a bisection of \mathbb{G} , with $|A| = |B| = n$. The subtrees \mathcal{B}_A and \mathcal{B}_B rooted at A and B are each over n vertices. Based on the definition of graph reassembling and Definition 57, we have $\alpha(\mathbb{G}, \mathcal{B}) \geq |\partial(A, B)|$. The conclusion of the lemma will follow from the fact that $\alpha(\mathbb{G}, \mathcal{B}) = |\partial(A, B)|$, which we show next.

In the rest of the proof we use the notation and definitions in the proof of Lemma 94. There are two cases, depending on whether $\{A, B\}$ is, or is not, of type $(V(H), V(I))$.

Case 1. If $\{A, B\}$ is of type $(V(H), V(I))$, we can assume that $V(H) \subseteq A$ and $V(I) \subseteq B$.

We pose:

$$X = A \cap V(G), \quad X' = B \cap V(G), \quad Y = A \cap V(H), \quad \text{and} \quad Z = B \cap V(I).$$

By construction, $|X| = |X'| = (n/2)$ and $|Y| = |Z| = (n/2)$. Let $C = |\partial(X, X')|$, which is the value of the bisection of the given graph G . Because H and I are copies of the complete graph $K_{(n/2)}$, the set of edges connecting Y to X' , and the set of edges connecting Z to X , satisfy the equalities:

$$|\partial(Y, X')| = |\partial(Z, X)| = (n/2)^2 = n^2/4.$$

Hence, using the notation of graph reassembling and Definition 57, we have:

$$|\partial(A, B)| = \text{degree}_{\mathbb{G}, \mathcal{B}}(A) = \text{degree}_{\mathbb{G}, \mathcal{B}}(B) = (n^2/4) + (n^2/4) + C = (n^2/2) + C.$$

Hence, $\alpha(\mathbb{G}, \mathcal{B}) \geq (n^2/2) + C$. The equality in fact holds because, as argued next, $\text{degree}_{\mathbb{G}, \mathcal{B}}(S) \leq (n^2/2) + C$ for every node/cluster of vertices S in the subtrees \mathcal{B}_A and \mathcal{B}_B .

Let S be a node in \mathcal{B}_A . (The same argument applies if S is a node in \mathcal{B}_B .) If $S = A$, we already know that $\text{degree}_{\mathbb{G}, \mathcal{B}}(S) \leq (n^2/2) + C$. Suppose S is not the root of \mathcal{B}_A , i.e., $S \neq A$. This implies $|S| \leq |A|/2$. Let $|S| = k$, so that $k \leq (n/2)$. Let $S_1 = S \cap X$ and $S_2 = S \cap Y$, with $|S_1| = k_1$ and $|S_2| = k_2$, so that also $k_1 + k_2 = k \leq (n/2)$. Note that $\partial(S_1, X') \subseteq \partial(X, X')$, so that if $C_1 = |\partial(S_1, X')|$, then $C_1 \leq C$. Also, $\partial(S_2, X') = \partial(S_2, Z) = \emptyset$. We conclude:

$$\text{degree}_{\mathbb{G}, \mathcal{B}}(S) = \underbrace{k_1(n/2) + C_1}_{|\partial(S_1, B)|} + \underbrace{k_1 k_2}_{|\partial(S_1, S_2)|} + \underbrace{k_2(n/2)}_{|\partial(S_2, X')|} = (k_1 + k_2)(n/2) + k_1 k_2 + C_1 \leq (n^2/2) + C.$$

Case 2. Suppose $\{A, B\}$ is not of type $(V(H), V(I))$. By Lemma 94, $\{A, B\}$ is not a minimum bisection and therefore $|\partial(A, B)| > (n^2/2) + C$ where C is defined as in Case 1 above. Hence $\alpha(\mathbb{G}, \mathcal{B}) > (n^2/2) + C$. By Case 1, $(\mathbb{G}, \mathcal{B})$ is not an α -optimal balanced reassembling. \square

Theorem 97. *For the class of simple undirected graphs G , the computation of α -optimal balanced reassemblings (G, \mathcal{B}) is an NP-hard problem.*

Proof. If a deterministic polynomial-time algorithm \mathcal{A} existed for computing an α -optimal balanced reassembling for an arbitrary graph G , then \mathcal{A} could be used for computing an α -optimal balanced reassembling for the augmented graph \mathbb{G} of an arbitrary graph $G \in \text{GRAPHS}(2^*)$. Hence, by Lemma 96, \mathcal{A} could also be used for computing a minimum bisection of \mathbb{G} in deterministic polynomial time. This would contradict the NP-hardness of $\text{MinBisection}(2^*)$, as asserted by Lemma 95. The desired conclusion follows. \square

5.8 β -Optimization of Balanced Reassembling Is NP-Hard

We need to introduce two variations of the k -CliqueCover problem (Definition 51).

Definition 98 (*Fixed-Size 4-CliqueCover, Equal-Size 4-CliqueCover*). In the *Fixed-Size 4-CliqueCover problem* we consider a graph G together with four positive integers $\{n_1, n_2, n_3, n_4\}$ such that $n_1 + n_2 + n_3 + n_4 = |V(G)|$ and we ask: Can we partition $V(G)$ into four disjoint subsets A_1, A_2, A_3 and A_4 of respective sizes n_1, n_2, n_3 and n_4 such that each of the induced subgraphs $G[A_1], G[A_2], G[A_3]$ and $G[A_4]$ is a complete graph (*i.e.*, A_1, A_2, A_3 and A_4 are cliques)?

In the *Equal-Size 4-CliqueCover problem* we consider a graph G and ask: Can we partition $V(G)$ into four disjoint subsets A_1, A_2, A_3 and A_4 of equal size, *i.e.*, $|A_1| = |A_2| = |A_3| = |A_4| = |V(G)|/4$, such that each of $G[A_1], G[A_2], G[A_3]$ and $G[A_4]$ is a complete graph (*i.e.*, A_1, A_2, A_3 and A_4 are cliques)? \square

Lemma 99. *Fixed-Size 4-CliqueCover is NP-complete.*

Proof. Given a 4-part partition $\{A_1, A_2, A_3, A_4\}$ of $V(G)$, we can verify in polynomial time that the induced graphs $\{G[A_1], G[A_2], G[A_3], G[A_4]\}$ are each complete and that their sizes are the

given $\{n_1, n_2, n_3, n_4\}$. So the problem is in NP.

We next show that NP-completeness follows by reduction from 4-CliqueCover, *i.e.*, the existence of a deterministic polynomial-time algorithm \mathcal{A} for Fixed-Size 4-CliqueCover would imply the existence of a deterministic polynomial-time algorithm for 4-CliqueCover. The input for the hypothetical \mathcal{A} consists of a graph G together with four positive integers $\{n_1, n_2, n_3, n_4\}$. For the desired reduction, we use the function $p(n, 4)$, a cubic polynomial in n , which counts the number of ways of partitioning positive integer n into 4 positive integers [4]:

$$p(n, 4) := \begin{cases} \left\lceil \frac{(n+1)^3}{144} - \frac{(n+1)}{48} \right\rceil & \text{if } n \text{ is even,} \\ \left\lceil \frac{(n+1)^3}{144} - \frac{(n+1)}{12} \right\rceil & \text{if } n \text{ is odd,} \end{cases}$$

where $\lceil x \rceil$ is the nearest integer to x . We leave to the reader the straightforward task of writing an algorithm \mathcal{A}' to generate all partitions of n into 4 positive integers, which runs in $\mathcal{O}(n^3)$ time. To decide whether an arbitrarily given graph G is a positive instance of 4-CliqueCover, we run algorithm \mathcal{A}' to generate the successive 4-part partitions of $n = |V(G)|$. The given G has a 4-clique cover iff algorithm \mathcal{A} returns “yes” when its input is: G together with at least one of these 4-part partitions of integer n . \square

Lemma 100. *Equal-Size 4-CliqueCover is NP-complete.*

Proof. If $\{A_1, A_2, A_3, A_4\}$ is a 4-part partition of $V(G)$, where $|A_1| = |A_2| = |A_3| = |A_4| = |V(G)|/4 = n/4$, then we can verify that the induced graphs $\{G[A_1], G[A_2], G[A_3], G[A_4]\}$ are each complete (can be done in polynomial time). So the problem is in NP.

NP-completeness follows by reduction from Fixed-Size 4-CliqueCover to Equal-Size 4-CliqueCover, *i.e.*, the existence of a deterministic polynomial-time algorithm \mathcal{A} for Equal-Size 4-CliqueCover would imply the existence of a deterministic polynomial-time algorithm for Fixed-Size 4-CliqueCover, as shown next. Given an arbitrarily given graph G and four positive integers $\{n_1, n_2, n_3, n_4\}$ such that $n_1 + n_2 + n_3 + n_4 = |V(G)| = n$, we introduce 4 new sets of vertices $\{A_1, A_2, A_3, A_4\}$ such

that:

$$|A_1| = n - n_1, \quad |A_2| = n - n_2, \quad |A_3| = n - n_3, \quad |A_4| = n - n_4.$$

We construct a new graph G' such that:

$$V(G') := V(G) \cup A_1 \cup A_2 \cup A_3 \cup A_4,$$

$$E(G') := E(G) \cup \{ \overline{vw} \mid v \in V(G) \text{ and } w \in A_1 \cup A_2 \cup A_3 \cup A_4 \}$$

$$\cup \{ \overline{vw} \mid v, w \in A_1 \} \cup \{ \overline{vw} \mid v, w \in A_2 \} \cup \{ \overline{vw} \mid v, w \in A_3 \} \cup \{ \overline{vw} \mid v, w \in A_4 \}.$$

In words, the new G' is obtained from G by adding four cliques, one clique on each of the new vertex sets in $\{A_1, A_2, A_3, A_4\}$, and by connecting every vertex in $A_1 \cup A_2 \cup A_3 \cup A_4$ with every vertex in $V(G)$. Hence, G' has $4n$ vertices, and there are no edges between the subgraphs $\{G[A_1], G[A_2], G[A_3], G[A_4]\}$.

Using the fact that no clique in G' can contain vertices from two distinct sets in $\{A_1, A_2, A_3, A_4\}$, we conclude that G is a positive instance of Fixed-Size 4-CliqueCover with sizes $\{n_1, n_2, n_3, n_4\}$ iff G' is a positive instance of Equal-Size 4-CliqueCover. \square

For the rest of this section we restrict attention to graphs G in the class $\text{GRAPHS}(2^*)$, introduced before Definition 50. A balanced reassembling (G, \mathcal{B}) of such a graph G is relative to a full binary tree \mathcal{B} of height $\log n$ (i.e., with $1 + \log n$ levels) where $n = |V(G)|$.

The nodes in a reassembling tree \mathcal{B} are each a cluster of vertices (Definition 52), a subset of $V(G)$. One of the measures on a node/cluster X in the reassembling (G, \mathcal{B}) is its height, denoted $height_{G, \mathcal{B}}(X)$. We can extend the measure $height$ to every edge $\overline{vw} \in E(G)$, by defining:

$$height_{G, \mathcal{B}}(\overline{vw}) := \min \{ height_{G, \mathcal{B}}(X) \mid X \in \mathcal{B} \text{ and both } v, w \in X \}.$$

In words, the height of \overline{vw} in (G, \mathcal{B}) is the height of the least node/cluster X that includes both endpoints of edge \overline{vw} . Note that, if \mathcal{B} is a full binary tree (the case of a balanced reassembling of $G \in \text{GRAPHS}(2^*)$), then the node/cluster X is at the same distance (or height) from the endpoints

(or leaf nodes) v and w .

Another way of understanding $\text{height}_{G,\mathcal{B}}(\overline{vw}) = p$, where $0 \leq p \leq \log n$, is that p is the level number in \mathcal{B} (starting from the bottom, with level 0 being the level of all leaf nodes) at which the two halves of edge \overline{vw} are spliced together, or at which the edge \overline{vw} is re-introduced in the reassembling.

Lemma 101. *If $G \in \text{GRAPHS}(2^*)$ and (G, \mathcal{B}) is a balanced reassembling of G , then:*

$$\beta(G, \mathcal{B}) = 2 \times \sum \{ \text{height}_{G,\mathcal{B}}(\overline{vw}) \mid \overline{vw} \in E(G) \}.$$

Informally, $\beta(G, \mathcal{B})$ is minimized (resp. maximized) when edges are spliced as soon as possible (as late as possible) in the reassembling.

Proof. Straightforward consequence of Definition 57 and the definition of the graph reassembling problem. A formal proof can be carried out by induction on $p \geq 1$, where $|V(G)| = n = 2^p$. All details omitted. \square

The proof of the next lemma is interesting in that it combines both algebraic reasoning (formulation of an instance of integer quadratic programming) and combinatorial reasoning (existence of a partition of $V(G)$ into four independent vertex sets of equal size).

Lemma 102. *Let $G \in \text{GRAPHS}(2^*)$ with $|V(G)| = n = 2^p$ for some $p \geq 2$. Let $A_1, A_2, A_3, A_4 \subseteq V(G)$ be four disjoint independent sets of vertices in G , each of size $n/4$. Let (G, \mathcal{B}) be a balanced reassembling and $X_1, X_2, X_3, X_4 \in \mathcal{B}$ be the nodes/vertex-clusters in the tree \mathcal{B} such that $|X_1| = |X_2| = |X_3| = |X_4| = n/4$.⁸*

Conclusion: *If the β -measure $\beta(G, \mathcal{B})$ is maximized, i.e.,*

$$\beta(G, \mathcal{B}) = \max \{ \beta(G, \mathcal{B}') \mid \mathcal{B}' \text{ is a balanced reassembling tree } \},$$

⁸ Stated differently, $\{X_1, X_2, X_3, X_4\}$ are the four nodes of \mathcal{B} such that:

$$\text{height}_{G,\mathcal{B}}(X_1) = \text{height}_{G,\mathcal{B}}(X_2) = \text{height}_{G,\mathcal{B}}(X_3) = \text{height}_{G,\mathcal{B}}(X_4) = p - 2,$$

where $p = \log n$ with $n = |V(G)|$, i.e., $\{X_1, X_2, X_3, X_4\}$ are the four grandchildren of the root node $V(G)$.

then $\{X_1, X_2, X_3, X_4\}$ are disjoint independent sets in G , not necessarily the same as $\{A_1, A_2, A_3, A_4\}$, each of size $n/4$.

Proof. Let $V(G) = \{v_1, \dots, v_n\}$. Assume $(X_1 \uplus X_2)$ and $(X_3 \uplus X_4)$ are the two children-nodes/vertex-clusters of the root $V(G)$ in \mathcal{B} . The height in (G, \mathcal{B}) of every vertex cluster in $\{X_1, X_2, X_3, X_4\}$ is $(p - 2)$, while the height of both $(X_1 \uplus X_2)$ and $(X_3 \uplus X_4)$ is $(p - 1)$, and the height of the root $V(G)$ is of course p .

Lemma 101 gives an alternative definition of $\beta(G, \mathcal{B})$, obtained by summing the heights in (G, \mathcal{B}) of all the edges. This definition is presumed in the formulation of the integer quadratic programming below. The problem of finding a balanced reassembling tree \mathcal{B} which maximizes $\beta(G, \mathcal{B})$ can be translated to an integer quadratic programming, as follows:

$$\begin{aligned}
 \text{maximize} \quad & \text{(i)} \quad 2p \times \sum_{\overline{v_i v_j} \in E(G)} (x_{i,1}x_{j,3} + x_{i,1}x_{j,4} + x_{i,2}x_{j,3} + x_{i,2}x_{j,4}) & + \\
 & \text{(ii)} \quad 2(p-1) \times \sum_{\overline{v_i v_j} \in E(G)} (x_{i,1}x_{j,2} + x_{i,3}x_{j,4}) & + \\
 & \text{(iii)} \quad 2(p-2) \times \sum_{\overline{v_i v_j} \in E(G)} (x_{i,1}x_{j,1} + x_{i,2}x_{j,2} + x_{i,3}x_{j,3} + x_{i,4}x_{j,4}) \\
 \\
 \text{subject to} \quad & \text{(i)} \quad \text{for all } 1 \leq i \leq n \text{ and } 1 \leq k \leq 4, \quad x_{i,k} \in \{0, 1\} \\
 & \text{(ii)} \quad \text{for all } 1 \leq k \leq 4, \quad \sum_{1 \leq i \leq n} x_{i,k} = \frac{n}{4} \\
 & \text{(iii)} \quad \text{for all } 1 \leq i \leq n, \quad \sum_{1 \leq k \leq 4} x_{i,k} = 1
 \end{aligned}$$

The optimization objective is quadratic and has three parts with respective coefficients $2p$, $2(p - 1)$, and $2(p - 2)$, while the constraints are linear. Every vertex $v_i \in V(G)$ corresponds to four variables $\{x_{i,1}, x_{i,2}, x_{i,3}, x_{i,4}\}$, which indicate which set in $\{X_1, X_2, X_3, X_4\}$ contains vertex v_i . Specifically, for every $1 \leq i \leq n$ and $1 \leq k \leq 4$:

$$x_{i,k} = \begin{cases} 1 & \text{if } v_i \in X_k, \\ 0 & \text{if } v_i \notin X_k. \end{cases}$$

To understand the preceding formulation as a $(0, 1)$ quadratic programming, observe that for every edge $\overline{v_i v_j}$:

$$x_{i,k} x_{j,\ell} = \begin{cases} 1 & \text{if } v_i \in X_k \text{ and } v_j \in X_\ell, \\ 0 & \text{if } v_i \notin X_k \text{ or } v_j \notin X_\ell. \end{cases}$$

Note that the suggested quadratic system is slightly relaxed in the sense that (as represented by part (iii) with coefficient $2(p-2)$ of the optimization objective) we assume that every edge $\overline{v_i v_j}$ whose endpoints are in the same X_k , *i.e.*, $v_i, v_j \in X_k$ for some $1 \leq k \leq 4$, contributes the maximum possible value, here $2(p-2)$, rather than its exact value to $\beta(G, \mathcal{B})$. The rest of the proof shows that this relaxation does not affect its correctness.

A straightforward re-ordering of terms shows that the optimization objective can be written as follows:

$$\theta := 2p \times \left(\sum_{\overline{v_i v_j} \in E(G), 1 \leq k \leq \ell \leq 4} x_{i,k} x_{j,\ell} \right) - 2\theta_1 - 4\theta_2 \quad \text{where}$$

$$\theta_1 := \sum_{\overline{v_i v_j} \in E(G)} (x_{i,1} x_{j,2} + x_{i,3} x_{j,4}) \quad \text{and}$$

$$\theta_2 := \sum_{\overline{v_i v_j} \in E(G), 1 \leq k \leq 4} x_{i,k} x_{j,k}.$$

The quantity θ_1 counts the number of edges $\overline{v_i v_j}$ satisfying one of two conditions:

- either the endpoints v_i and v_j are in X_1 and X_2 ,
- or the endpoints v_i and v_j are in X_3 and X_4 .

Every edge satisfying one of the two preceding conditions has height $(p-1)$ in (G, \mathcal{B}) . The quantity θ_2 counts the number of edges $\overline{v_i v_j}$ whose endpoints v_i and v_j are in the same X_k , and whose height is therefore $\leq (p-2)$ in (G, \mathcal{B}) . We now observe that:

$$\sum_{\overline{v_i v_j} \in E(G), 1 \leq k \leq \ell \leq 4} x_{i,k} x_{j,\ell} = |E(G)| = m.$$

The optimization objective can now be simplified to read:

$$\theta = 2pm - 2\theta_1 - 4\theta_2 = 2(p-1)m + 2(m - \theta_1 - \theta_2) - 2\theta_2 = 2(p-1)m + \theta' \quad \text{where}$$

$$\theta' := 2(m - \theta_1 - \theta_2) - 2\theta_2 .$$

Hence, maximizing θ is equivalent to maximizing θ' , and the latter is maximized when $(m - \theta_1 - \theta_2)$ is maximized and θ_2 is minimized. But $(m - \theta_1 - \theta_2)$ is the number of edges whose height in (G, \mathcal{B}) is p , *i.e.*, the edges $\overline{v_i v_j}$ such that $v_i \in (X_1 \uplus X_2)$ and $v_j \in (X_3 \uplus X_4)$, while θ_2 is the number of edges $\overline{v_i v_j}$ whose endpoints v_i and v_j are in the same X_k and whose height is $\leq (p-2)$.

We switch to combinatorial reasoning, by invoking the fact that G has four disjoint independent sets, each with $(n/4)$ vertices. There is no need to explicitly solve the integer quadratic programming above. Maximizing θ' means choosing $(X_1 \uplus X_2, X_3 \uplus X_4)$ as a bisection with a maximum cut $|\partial(X_1 \uplus X_2, X_3 \uplus X_4)|$, *i.e.*, choosing $(X_1 \uplus X_2)$ and $(X_3 \uplus X_4)$ as independent sets. And minimizing θ_2 , which is here possible down to $\theta_2 = 0$, means choosing each of X_1, X_2, X_3 , and X_4 , as an independent set. \square

Lemma 103. *Let $G \in \text{GRAPHS}(2^*)$ be a positive instance of Equal-Size 4-CliqueCover, with $|V(G)| = n = 2^p$ for some $p \geq 2$. Let (G, \mathcal{B}) be a balanced reassembling and $\{X_1, X_2, X_3, X_4\} \subseteq \mathcal{B}$ be the nodes/vertex-clusters in the tree \mathcal{B} such that $|X_1| = |X_2| = |X_3| = |X_4| = n/4$.*

Conclusion: *If (G, \mathcal{B}) is a β -optimal balanced reassembling, then $\{X_1, X_2, X_3, X_4\}$ is an Equal-Size 4-CliqueCover of G .*

Proof. Because (G, \mathcal{B}) is a β -optimal balanced reassembling (Definition 57), we have:

$$\beta(G, \mathcal{B}) := \min \{ \beta(G, \mathcal{B}') \mid \mathcal{B}' \text{ is a balanced binary tree over } V(G) \}.$$

We write K_n for the complete graph over $n = |V(G)|$ vertices, and \overline{G} for the complement of G . We thus have $V(\overline{G}) = V(G)$ and $E(\overline{G}) = E(K_n) - E(G)$. Clearly, $G = \overline{\overline{G}}$ and we can write

$G = K_n - \overline{G}$. Hence:

$$\beta(G, \mathcal{B}) = \min \{ \beta(K_n - \overline{G}, \mathcal{B}') \mid \mathcal{B}' \text{ is a balanced binary tree over } V(G) \}.$$

The β -measure of a balanced reassembling of K_n , call it M , does *not* depend on the reassembling tree, *i.e.*, for all balanced reassembling trees \mathcal{B}_1 and \mathcal{B}_2 on n vertices, it holds that:

$$\beta(K_n, \mathcal{B}_1) = \beta(K_n, \mathcal{B}_2) = M.$$

Hence, the following equality holds:

$$\beta(G, \mathcal{B}) = M - \max \{ \beta(\overline{G}, \mathcal{B}') \mid \mathcal{B}' \text{ is a balanced binary tree over } V(G) \}.$$

In words, the value of $\beta(G, \mathcal{B})$ is *minimized* when the value of $\beta(\overline{G}, \mathcal{B}')$ is *maximized*. Since (G, \mathcal{B}) is β -optimal, $\beta(\overline{G}, \mathcal{B}')$ is *maximized*.

By hypothesis, G is an instance of Equal-Size 4-CliqueCover, *i.e.*, there are disjoint sets A_1, A_2, A_3 and A_4 of vertices in G such that the induced subgraphs in $\{G[A_1], G[A_2], G[A_3], G[A_4]\}$ are each a complete graph with $n/4$ vertices. Hence, the corresponding subgraphs in \overline{G} , namely $\overline{G}[A_1], \overline{G}[A_2], \overline{G}[A_3]$ and $\overline{G}[A_4]$, are each an edgeless graph with $n/4$ vertices. Equivalently, A_1, A_2, A_3 and A_4 are disjoint independent sets in \overline{G} , each with $n/4$ vertices. Hence, by Lemma 102, $\{X_1, X_2, X_3, X_4\}$ are disjoint independent sets in \overline{G} , and therefore disjoint cliques in G , each with $n/4$ vertices. \square

Theorem 104. *For the class of simple undirected graphs G , the computation of β -optimal balanced reassemblings (G, \mathcal{B}) is an NP-hard problem.*

Proof. If a deterministic polynomial-time algorithm existed for producing a β -optimal balanced reassembling (G, \mathcal{B}) of an arbitrarily given G , then this algorithm could be used again to decide in deterministic polynomial-time whether a graph in $\text{GRAPHS}(2^*)$ is a positive instance of Equal-Size 4-CliqueCover, by Lemma 103. This would in turn contradict Lemma 100 asserting the NP-

completeness of Equal-Size 4-CliqueCover. The desired conclusion follows. \square

5.9 Supplementary Proofs for Sections 5.5 and 5.6

In order to facilitate the grasp of the different concepts and their mutual dependence, we supply the details of several long straightforward and/or highly technical proofs which we omitted in Sections 5.5 and 5.6.

Proof of Theorem 77. Let $G = (\mathbf{V}, \mathbf{E})$ be an arbitrary simple undirected graph, with $|\mathbf{V}| = n$. It suffices to show that if (G, φ) is an α -optimal linear arrangement, then the linear reassembling (G, \mathcal{L}) induced by (G, φ) is also α -optimal, using Definition 62.

In the notation of Definition 62, the clusters of \mathcal{L} of size ≥ 2 are $\{X_1, \dots, X_{n-1}\}$. For the singleton clusters of \mathcal{L} , we pose $Y_{i-1} := \{\varphi^{-1}(i)\}$, where $1 \leq i \leq n$. From Definition 58:

$$\begin{aligned}\alpha(G, \varphi) &= \max \{ \text{degree}(Y_0), \max \{ \text{degree}(X_j) \mid 1 \leq j \leq n-1 \} \}, \\ \alpha(G, \mathcal{L}) &= \max \{ \max \{ \text{degree}(Y_i) \mid 0 \leq i \leq n-1 \}, \max \{ \text{degree}(X_j) \mid 1 \leq j \leq n-1 \} \}.\end{aligned}$$

By way of getting a contradiction, assume that (G, φ) is α -optimal but that the induced (G, \mathcal{L}) is not α -optimal. Hence, there is another linear reassembling (G, \mathcal{L}') which is α -optimal such that $\alpha(G, \mathcal{L}') < \alpha(G, \mathcal{L})$. Using the same notation for both (G, \mathcal{L}) and (G, \mathcal{L}') , where every name related to the latter is decorated with a prime, the inequality $\alpha(G, \mathcal{L}') < \alpha(G, \mathcal{L})$ implies the inequality:

$$\begin{aligned}& \max \{ \max \{ \text{degree}(Y'_i) \mid 0 \leq i \leq n-1 \}, \max \{ \text{degree}(X'_j) \mid 1 \leq j \leq n-1 \} \} \\ & < \max \{ \max \{ \text{degree}(Y_i) \mid 0 \leq i \leq n-1 \}, \max \{ \text{degree}(X_j) \mid 1 \leq j \leq n-1 \} \}.\end{aligned}$$

But $\max \{ \text{degree}(Y'_i) \mid 0 \leq i \leq n-1 \} = \max \{ \text{degree}(Y_i) \mid 0 \leq i \leq n-1 \}$, which implies two

inequalities:

$$\max \{ \text{degree}(Y_i) \mid 1 \leq j \leq n-1 \} < \max \{ \text{degree}(X_j) \mid 1 \leq j \leq n-1 \}, \quad (1)$$

$$\max \{ \text{degree}(X'_j) \mid 1 \leq j \leq n-1 \} < \max \{ \text{degree}(X_j) \mid 1 \leq j \leq n-1 \}. \quad (2)$$

Hence, by inequality (1), we have:

$$\alpha(G, \varphi) = \alpha(G, \mathcal{L}) = \max \{ \text{degree}(X_j) \mid 1 \leq j \leq n-1 \}.$$

Consider now the linear arrangement (G, φ') induced by the linear reassembling (G, \mathcal{L}') , using Definition 61. We have:

$$\alpha(G, \varphi') = \max \{ \text{degree}(Y'_0), \max \{ \text{degree}(X'_j) \mid 1 \leq j \leq n-1 \} \}.$$

If $\text{degree}(Y'_0) \geq \max \{ \text{degree}(X'_j) \mid 1 \leq j \leq n-1 \}$, then inequality (1) implies $\alpha(G, \varphi') < \alpha(G, \varphi)$, else inequality (2) implies again $\alpha(G, \varphi') < \alpha(G, \varphi)$. In both cases, the α -optimality of (G, φ) is contradicted. \square

For the proofs of Lemma 87 and Lemma 88, we take a closer look at how the vertices of K_{p+1} are positioned in the sequence \mathcal{S} in (\diamond) in Section 5.6. From the fact that p is the sum of all the vertex degrees in G , it follows that p is even and $p+1$ odd. From the sequence \mathcal{S} , we can extract the subsequence $(\mathcal{S} \mid K_{p+1})$ consisting of all the vertices of K_{p+1} and corresponding cutwidths:

$$(\mathcal{S} \mid K_{p+1}) = [x_{i_1} \quad s_{i_1} \quad x_{i_2} \quad s_{i_2} \quad \cdots \quad \cdots \quad x_{i_p} \quad s_{i_p} \quad x_{i_{p+1}}] \quad (\diamond\diamond)$$

where $\{i_1, \dots, i_{p+1}\} \subseteq \{1, \dots, n+p\}$ and $\{x_{i_1}, \dots, x_{i_{p+1}}\} = \{u_1, \dots, u_p\} \cup \{w\}$. In the preceding sequence, every vertex has the same degree p in the subgraph K_{p+1} . In the full graph G_w , every vertex from K_{p+1} has again the same degree p , except for the distinguished vertex w which has

degree $p + d$ where $d = \text{degree}_G(w)$. In particular, we have:

$$s_{i_1} = p, \quad s_{i_2} = 2 \cdot (p - 1), \quad s_{i_3} = 3 \cdot (p - 2), \quad \dots, \quad s_{i_{p-1}} = (p - 1) \cdot 2, \quad s_{i_p} = p.$$

The mid-point of $(\mathcal{S} \mid K_{p+1})$ is $x_{i_{(p/2)+1}}$. The two adjacent cutwidths of the mid-point $x_{i_{(p/2)+1}}$ are:

$$s_{i_{(p/2)}} = \frac{p}{2} \cdot \left(\frac{p}{2} + 1\right) \quad \text{and} \quad s_{i_{(p/2)+1}} = \left(\frac{p}{2} + 1\right) \cdot \frac{p}{2},$$

so that also, as one can readily check:

$$s_{i_{(p/2)}} = s_{i_{(p/2)+1}} = \frac{p^2 + 2p}{4} = \max\{s_{i_1}, s_{i_2}, \dots, s_{i_p}\},$$

and the sequence of cutwidths $(s_{i_1}, \dots, s_{i_p})$ is equal to its own reverse $(s_{i_p}, \dots, s_{i_1})$. Moreover, for every j such that $1 \leq j < i_1$ or $i_{p+1} < j \leq n + p$, we have $s_j = 0$. Also, it is intuitively useful for the argument in the proof of Lemma 87 to keep in mind that:

$$\begin{aligned} (s_{i_1} - s_j) &= p, & \text{for every } 1 \leq j < i_1, \\ (s_{i_2} - s_j) &= p - 2, & \text{for every } i_1 \leq j < i_2, \\ (s_{i_3} - s_j) &= p - 4, & \text{for every } i_2 \leq j < i_3, \\ \dots & \dots & \dots \\ (s_{i_{(p/2)}} - s_j) &= 2, & \text{for every } i_{(p/2)-1} \leq j < i_{p/2}, \\ (s_{i_{(p/2)+1}} - s_j) &= 0, & \text{for every } i_{(p/2)} \leq j < i_{(p/2)+1}. \end{aligned}$$

Proof of Lemma 87. In the sequence \mathcal{S} in (\diamond) in Section 5.6, suppose:

- x_i is the leftmost vertex in $U \cup \{w\}$,
- x_j is the leftmost vertex in $\mathbf{V} - \{w\}$ to the right of x_i ,
- x_ℓ is the rightmost vertex in $U \cup \{w\}$,
- x_k is the rightmost vertex in $\mathbf{V} - \{w\}$ to the left of x_ℓ ,

where $1 \leq i \leq j \leq k \leq \ell \leq p+n$. Graphically, \mathcal{S} can be represented by:

$$\underbrace{x_1 \cdots x_{i-1}}_{\text{all in } \mathbf{V} - \{w\}} \quad \underbrace{x_i \cdots x_{j-1}}_{\text{all in } U \cup \{w\}} \quad \textcircled{x_j} \quad x_{j+1} \cdots x_{k-1} \quad \textcircled{x_k} \quad \underbrace{x_{k+1} \cdots x_\ell}_{\text{all in } U \cup \{w\}} \quad \underbrace{x_{\ell+1} \cdots x_{p+n}}_{\text{all in } \mathbf{V} - \{w\}}$$

The circled vertices, x_j and x_k , are in $\mathbf{V} - \{w\}$. If \mathcal{S} is scattered, then $1 \leq i < j$ and/or $k < \ell \leq n+p$, with the possibility that $j = k$ in which case there is only one vertex in $\mathbf{V} - \{w\}$ inserted between all the vertices of $U \cup \{w\}$. We define:

$$\text{scatter}(\mathcal{S}) := \min \{j - i, \ell - k\} \geq 1,$$

when \mathcal{S} is scattered. If \mathcal{S} is not scattered, we set $\text{scatter}(\mathcal{S}) := 0$, so that \mathcal{S} is scattered iff $\text{scatter}(\mathcal{S}) \geq 1$. Moreover, with p even and $p+1$ odd, it is always the case that $\text{scatter}(\mathcal{S}) \leq p/2$, so that if \mathcal{S} is scattered, then:

$$1 \leq \text{scatter}(\mathcal{S}) \leq \frac{p}{2}.$$

To complete the proof, it suffices to show that if \mathcal{S} is scattered, then we can define another sequence \mathcal{S}' from \mathcal{S} such that:

$$\beta(\mathcal{S}') < \beta(\mathcal{S}) \quad \text{and} \quad \text{scatter}(\mathcal{S}') < \text{scatter}(\mathcal{S}).$$

We obtain \mathcal{S}' from \mathcal{S} as follows:

- if $j - i \leq \ell - k$, remove x_j from the j -th position and insert it between x_{i-1} and x_i ,
- if $j - i > \ell - k$, remove x_k from the k -th position and insert it between x_ℓ and $x_{\ell+1}$.

With no loss of generality, let $j - i \leq \ell - k$. The portion of \mathcal{S} under consideration is therefore:

$$(r_{i-1}, s_{i-1}) \quad \underbrace{x_i \quad (r_i, s_i) \quad \cdots \quad (r_{j-2}, s_{j-2}) \quad x_{j-1}}_{\text{all in } U \cup \{w\}} \quad (r_{j-1}, s_{j-1}) \quad \textcircled{x_j} \quad (r_j, s_j)$$

and the order of all the vertices in the new \mathcal{S}' is:

$$\underbrace{x_1 \cdots x_{i-1}}_{\text{all in } \mathbf{V} - \{w\}} \quad \textcircled{x_j} \quad \underbrace{x_i \cdots x_{j-1}}_{\text{all in } U \cup \{w\}} \quad x_{j+1} \cdots x_{k-1} \quad \textcircled{x_k} \quad \underbrace{x_{k+1} \cdots x_\ell}_{\text{all in } U \cup \{w\}} \quad \underbrace{x_{\ell+1} \cdots x_{p+n}}_{\text{all in } \mathbf{V} - \{w\}}$$

Let $x_j = v \in \mathbf{V} - \{w\}$ and partition $d = \text{degree}_G(v)$ into $d = d^L + d^R$, where:

- d^L is the number of vertices in $\{x_1, \dots, x_{j-1}\} \cap \mathbf{V}$ which are connected to v ,
- d^R is the number of vertices in $\{x_{j+1}, \dots, x_{n+p}\} \cap \mathbf{V}$ which are connected to v .

There are different cases, depending on:

- the value of $j - i$ between 1 and $p/2$,
- the value of $d^R - d^L$ between $-d$ and $+d$,
- whether the distinguished vertex w is in $\{x_i, \dots, x_{j-1}\}$ or in $\{x_{j+1}, \dots, x_\ell\}$,
- whether v is connected to w or not.

We consider only one of the cases, which is also a “worst case” to explain, and leave to the reader all the other cases, which are simple variations of this “worst case”. For the “worst case” which we choose to consider, let:

- (1) $j = i + p/2$ so that $j - i = p/2$,
- (2) $d^L = 0$ so that $d^R - d^L = d$,
- (3) w is in $\{x_{j+1}, \dots, x_\ell\}$,
- (4) there is an edge \overline{vw} connecting v and w .

With assumptions (1) to (4), as well as after:

- substituting $i + (p/2)$ for j ,
- replacing the sequence of cutwidths $s_{i-1}, s_i, \dots, s_{i+(p/2)-1}, s_{i+(p/2)}, s_{i+(p/2)+1}$ by their actual values $0, p, \dots, (p^2 + 2p - 8)/4, (p^2 + 2p)/4, (p^2 + 2p)/4$, respectively,
- and posing $r := r_{i-1}$,

the portion of \mathcal{S} under consideration becomes:

$$(r, 0) \quad \underbrace{x_i \quad (r, p) \cdots \left(r, \frac{p^2 + 2p - 8}{4}\right) \quad x_{i+(p/2)} \quad \left(r, \frac{p^2 + 2p}{4}\right)}_{\text{all in } U \cup \{w\}} \quad \bigcirc(v) \quad \left(r + d, \frac{p^2 + 2p}{4}\right)$$

The corresponding portion in the new \mathcal{S}' is:

$$(r, 0) \quad \bigcirc(v) \quad (r + d, 0) \quad \underbrace{x_i \quad (r + d, p) \cdots \left(r + d, \frac{p^2 + 2p - 8}{4}\right) \quad x_{i+(p/2)} \quad \left(r + d, \frac{p^2 + 2p}{4}\right)}_{\text{all in } U \cup \{w\}}$$

with all the cutwidths to the left and to the right of the shown portion being identical in \mathcal{S} and \mathcal{S}' .

It is now readily seen that the value of $\beta(\mathcal{S}')$ is:

$$\beta(\mathcal{S}') = \beta(\mathcal{S}) - (p^2 + 2p)/4 + (p/2)d = \beta(\mathcal{S}) - \frac{p^2 - 2(d-1)p}{4}$$

Let $\Delta := \sum \{ \text{degree}_G(x) \mid x \in \mathbf{V} \}$. By the construction of the auxiliary graph G_w , we have $p = \Delta$.

The value of $d = \text{degree}_G(v) \leq \Delta/2$, the upper bound $\Delta/2$ being the extreme case when G is a star graph with: v at its center, $\Delta/2$ leaf vertices among $\{x_{j+1}, \dots, x_{n+p}\} \cap \mathbf{V}$, and all other vertices of \mathbf{V} being isolated. Hence,

$$p^2 - 2(d-1)p \leq \Delta^2 - 2\left(\frac{\Delta}{2} - 1\right)\Delta = \Delta^2 - \Delta^2 + 2\Delta = 2\Delta.$$

Hence, $\beta(\mathcal{S}') \leq \beta(\mathcal{S}) - \Delta/2$, so that $\beta(\mathcal{S}') < \beta(\mathcal{S})$ which is the desired conclusion. \square

For precision in the next proof, we introduce the measure of *unbalance*.

Definition 105 (Unbalance). Consider the sequence \mathcal{S} in (\diamond) in Section 5.6.

- Let $a^L \geq 0$ be the number of vertices from $\mathbf{V} - \{w\}$ to the left of w , and $a^R \geq 0$ be the number of vertices from $\mathbf{V} - \{w\}$ to the right of w .
- Let $b^L \geq 0$ be the number of vertices from U to the left of w , and $b^R \geq 0$ be the number of vertices from U to the right of w .

The unbalance of \mathcal{S} is measured by:

$$\text{unbal}(\mathcal{S}) := \min \{ (n - a^L - 1) + (p - b^R), (n - a^R - 1) + (p - b^L) \}.$$

The quantity $(n - a^L - 1) + (p - b^R)$ measures \mathcal{S} 's unbalance on the left, and similarly $(n - a^R - 1) + (p - b^L)$ measures \mathcal{S} 's unbalance on the right. It is useful to keep in mind that:

$$(p - b^R) + (p - b^L) = p \quad \text{and} \quad (n - a^L - 1) + (n - a^R - 1) = n - 1,$$

so that, if the quantity $(n - a^L - 1) + (p - b^R)$ or the quantity $(n - a^R - 1) + (p - b^L)$ is reduced to 0, then the other of these two quantities is increased to $n - 1 + p$. \square

Proof of Lemma 88. By Lemma 87, we can assume that \mathcal{S} is not scattered. It suffices to show that if $\text{unbal}(\mathcal{S}) \geq 1$, we can define another sequence \mathcal{S}' from \mathcal{S} such that $\beta(\mathcal{S}') < \beta(\mathcal{S})$ and $\text{unbal}(\mathcal{S}') < \text{unbal}(\mathcal{S})$. We use the notation in the proof of Lemma 87. The portion of \mathcal{S} that we examine closely is:

$$(r_{i-1}, s_{i-1}) \quad x_i \quad \underbrace{(r_i, s_i) \quad x_{i+1} \quad (r_{i+1}, s_{i+1}) \quad \cdots \quad (r_{i+p-1}, s_{i+p-1}) \quad x_{i+p}}_{\text{all in } U \cup \{w\}} \quad (r_{i+p}, s_{i+p})$$

where:

$$\mathbf{V} - \{w\} = \{x_1, \dots, x_{i-1}\} \cup \{x_{i+p+1}, \dots, x_{n+p}\}, \quad \text{with } 1 \leq i \leq n,$$

$$U \cup \{w\} = \{x_i, \dots, x_{i+p}\}, \quad \text{with } w = x_{i+k} \text{ and } 0 \leq k \leq p.$$

We partition $d = \text{degree}_G(x_{i+k}) = \text{degree}_G(w)$ into $d = d^L + d^R$, where:

- $d^L \geq 0$ is the number of vertices in $\{x_1, \dots, x_{i-1}\}$ which are connected to $w = x_{i+k}$ in G ,
- $d^R \geq 0$ is the number of vertices in $\{x_{i+p+1}, \dots, x_{n+p}\}$ which are connected to $w = x_{i+k}$ in G .

And we partition $e = \text{degree}_{K_{p+1}}(x_{i+k}) = \text{degree}_{K_{p+1}}(w)$ into $e = e^L + e^R = p$, where:

- $e^L = k$ is the number of vertices in $\{x_i, \dots, x_{i+k-1}\}$ which are connected to $w = x_{i+k}$ in K_{p+1} ,
- $e^R = (p-k)$ is the number of vertices in $\{x_{i+k+1}, \dots, x_{i+p}\}$ which are connected to $w = x_{i+k}$ in K_{p+1} .

Though not explicitly used below, it is worth noting that e^L and e^R here are the same as b^L and b^R in Definition 105 because K_{p+1} is a complete graph (but d^L and d^R are not the same as a^L and a^R). We consider 5 separate cases, $\{(a), (b), (c), (d), (e)\}$:

- (a) $k = 0$, which implies $e^L = 0$ and $e^R = p$.

In case (a), because $\text{unbal}(\mathcal{S}) \neq 0$ by hypothesis, it must be that $\{x_{i+p+1}, \dots, x_{n+p}\} \neq \emptyset$. It suffices to move the vertices in $\{x_{i+p+1}, \dots, x_{n+p}\}$ to the left of $w = x_i$, also preserving their order

$$x_1, \dots, x_{i-1}, x_{i+p+1}, \dots, x_{n+p}.$$

Using a reasoning similar to that in the proof of Lemma 87, we leave it to the reader to show that $\text{unbal}(\mathcal{S}') = 0$ and $\beta(\mathcal{S}') < \beta(\mathcal{S})$ for the resulting sequence \mathcal{S}' .

- (b) $k = p$, which implies $e^L = p$ and $e^R = 0$.

Case (b) is similar to case (a). Because $\text{unbal}(\mathcal{S}) \neq 0$ by hypothesis, it must be that $\{x_1, \dots, x_{i-1}\} \neq \emptyset$. In this case, we move the vertices in $\{x_1, \dots, x_{i-1}\}$ to the right of $w = x_{i+p}$. Again, we leave it to the reader to show that $\text{unbal}(\mathcal{S}') = 0$ and $\beta(\mathcal{S}') < \beta(\mathcal{S})$ for the resulting sequence \mathcal{S}' .

For the three remaining cases, we can assume that neither $k = 0$ nor $k = p$, *i.e.*, both $w \neq x_i$ and $w \neq x_{i+p}$. Two cases of these three are:

- (c) $d^L > d^R$, in which case we transpose $w = x_{i+k}$ and x_i .
- (d) $d^L < d^R$, in which case we transpose $w = x_{i+k}$ and x_{i+p} .

By a reasoning similar to that in the proof of Lemma 87, we leave to the reader the straightforward details showing that $\beta(\mathcal{S}') < \beta(\mathcal{S})$ in both case (c) and case (d).

- (e) $d^L = d^R$, in which case the value of $\beta(\mathcal{S})$ remains unchanged by transposing $w = x_{i+k}$ and x_i , or by transposing $w = x_{i+k}$ and x_{i+p} , and so we need an additional argument.

The additional argument for case (e), is to first transpose $w = x_{i+k}$ and x_i , or alternatively transpose $w = x_{i+k}$ and x_{i+p} , thus reducing case (e) to case (a), or alternatively reducing case (e) to case (b).

□

5.10 Conclusion and Future Work

In Remarks 76, 82, several open problems from the literature were mentioned which are still unresolved to the best of our knowledge, and 93. If these open problems were solved, partially or optimally, they would permit various simplifications in our proofs. In particular, even though one of our reductions can be carried out in polynomial time by invoking an earlier result on cutwidths (Lemmas 70 and 71), its $\mathcal{O}(n^{12})$ complexity is prohibitive (see the proof of Theorem 74); this is the reduction that reduces the α -optimality of linear arrangements to the α -optimality of linear reassemblings.

Future work related to graph reassembling includes a study of classes of graphs for which α -optimization and/or β -optimization of their reassembling can be carried out in low-degree polynomial times. On the other hand, as indicated before, the smaller the α and β measures are, the more efficient the execution of programs are in our compositional approach presented in Chapter 3. Beinstock in [17] presents some elementary classes of graphs with small optimal tree congestion as well as an upper bound for the value of optimal tree congestion based on the *tree decomposition* of the graphs. Deciding whether the width of tree decomposition of an arbitrary graph is at most k is NP-complete [5], however the problem is tractable for small and fixed values of k . Bodlaender in [18] surveys a list of graph classes for which the treewidth can be computed in polynomial time. Similarly relating tree width to minimum tree congestion of graphs, [13] characterizes the class of graphs that have α -measure at most 3, but extending these result to different variety of graph reassembling such as balanced case and also study of the characterization of the classes of graphs with higher value of α -measure is open to be investigated in future.

Finally, there is the question of computing approximations of α -optimal and β -optimal graph reassembling, whereby we can turn the NP-hardness of any of the preceding optimizations into polynomially-solvable optimizations. In the case of linear reassembling, the literature on approximation algorithms dealing with graph layout problems is likely to be an important resource to draw from (among many other papers, the older [7, 66, 81] the more recent [23], and the survey [79]). In [65] the problem of finding call routing trees with minimum congestion has been studied and an approximate method for finding a solution within a $O(\log n)$ factor from the optimal solution is suggested. As discussed in Chapter 6, the problem of finding call routing trees with minimum congestion is directly related to our α -optimal graph reassembling problem. Hence the natural question is if similar methods can be facilitated in order to find approximation of α -optimal and β -optimal for balanced reassembling as well as other variations of graph reassembling.

Chapter 6

Graph Reassembling as a Graph Embedding Problem

The communication tree embedding problem is the problem of mapping a set of communicating terminals, represented by a graph G , into the set of vertices of some physical network represented by a tree \mathbf{T} . In the case where the vertices of G are mapped into the leaves of the host tree \mathbf{T} the underlying tree is called a *routing tree* and if the internal vertices of \mathbf{T} are forced to have degree 3, the host tree is known as *layout tree*. Different optimization problems have been studied in the class of communication tree problems such as well-known minimum edge *dilation* and minimum edge *congestion* problems. In this chapter we study the relation between tree layout problem and graph reassembling problem.

6.1 Background and Motivation

In this chapter we study the relation between tree layout problem and graph reassembling problem as defined in Chapter 5.

Consider a (not necessary simple) graph $G = (\mathbf{V}, \mathbf{E})$, partitioned into the set of $|\mathbf{V}|$ one-vertex components by cutting every edge into into two halves. Reassembling of the graph G corresponds to the problem of finding the sequence of edge splicings that minimizes two measures that depend on the edge-boundary degrees of assembled components in the intermediate steps of reassembling G . The first step of the reassembling sequence initiates with the set of one-vertex components, and the final step results in the input graph G . The optimization goal of the graph reassembling can be either minimizing the maximum edge-boundary degree encountered during the reassembling process, which is called the α -measure of the reassembling, or the sum of all edge-boundary degrees,

denoted by β -measure.

Every reassembling sequence for graph G correspond to a unique binary tree \mathcal{B} (called *binary reassembling tree*), where the set of leaf nodes is bijectively related to the set of one-vertex components and the root node correspond to the reassembled graph G . Every internal node $w \in \mathbf{V}_I(\mathcal{B})$, as the root of a subtree, represented by \mathcal{B}_w , correspond to the vertex induced sub-graph of G comprising the set of vertices represented by leaf nodes of \mathcal{B}_w . Accordingly, the graph reassembling problem can be translated into a special case of graph embedding problems in the sense that, given an input graph G , the goal is to find a binary tree \mathcal{B} and a bijective mapping from the set of vertices of G to the leaf nodes of \mathcal{B} . Hence, in this chapter we study the graph reassembling problem as a variation of graph embedding problem, called *tree layout* problem. Informally speaking, tree layout problem concerns the task of embedding the vertices of an input graph G into the leaf nodes of some rooted binary tree \mathbf{T} .

6.2 Definitions and Problem Statement

Consider a group of terminals communicating via a finite network $G = (\mathbf{V}, \mathbf{E})$, where the set of vertices (finite set \mathbf{V}) and edges (finite set \mathbf{E}), respectively represent the collection of terminals and their direct communication paths. We show $|\mathbf{V}|$ by n and $|\mathbf{E}|$ as m . The general communication tree embedding problem is the problem of mapping the set of terminals into the set of vertices of some physical network represented by a tree \mathbf{T} . Accordingly, the two vertices $v, u \in \mathbf{V}(G)$ that are directly connected via $e \in \mathbf{E}(G)$, are connected indirectly via some path $P_{\mathbf{T}}(v, u)$ in \mathbf{T} . In the case where the vertices of G are mapped to the leaves of the host tree, the underlying tree is called a *routing tree*. Initially in this chapter we focus on the case where the internal vertices of the host tree have degree 3 (known as *tree layout* problem). We denote the sets of leaf nodes and internal nodes of tree \mathbf{T} respectively by $\mathbf{V}_L(\mathbf{T})$ and $\mathbf{V}_I(\mathbf{T})$.¹

For a graph G and a communication tree \mathbf{T} for G , there are different measures defined in this context. In following we define the two measures that we are the related to the previously defined

¹We try to use the term node in case of trees as opposed to the term vertex, which we use for general graphs.

α and β measures in the context of graph reassembling problem. For a comprehensive list of measures, an interested reader can refer to [52].

Definition 106 (Edge Dilation). Consider a graph G and a communication tree \mathbf{T} and a bijection φ from vertices of G to leaf nodes of \mathbf{T} . The *dilation* of an edge $\{u, v\} \in \mathbf{E}(G)$ (shown by $\lambda(uv, \mathbf{T}, \varphi, G)$) is the distance between $\varphi(u)$ and $\varphi(v)$ in \mathbf{T} . \square

We represent the distance of two vertices $\{u, v\}$ in a graph G with $d_G(u, v)$

Definition 107 (Edge Congestion). Give a graph G and a communication tree \mathbf{T} and a bijective mapping $\varphi : \mathbf{V}(G) \rightarrow \mathbf{V}_L(\mathbf{T})$. The *congestion* of an edge $\{x, y\} \in \mathbf{E}(\mathbf{T})$ (denoted by $\delta(xv, \mathbf{T}, \varphi, G)$) is the the number of edges in $\{u, v\} \in \mathbf{E}(G)$ that in \mathbf{T} , the path $P_{\mathbf{T}}(\varphi(v), \varphi(u))$ traverse trough $\{x, y\}$. \square

Based on the definition of the communication tree for a graph G , removal of every edge $\{x, y\} \in \mathbf{E}(\mathbf{T})$ partitions the set of vertices of G into two component. Hence every edge of tree corresponds to a cut in G . Therefore, the congestion of $\{x, y\} \in \mathbf{E}(\mathbf{T})$ is the size of the cut it corresponds to.

Several optimization problems can be defined based on these two measures. *Minimum tree layout dilation* is the problem of finding a tree layout for a given graph G such that the maximum edge dilation is minimized, where the maximum is taken over all edge of G . In [74] it is shown that the problem of finding a tree layout with minimum dilation is NP-hard, when the layout tree is rooted. Similarly, given a graph G , in *minimum tree layout congestion* problem the goal is to find a tree layout \mathbf{T} , such that the maximum edge congestion is minimized. In [86] Seymour and Thomas show that the minimum tree layout congestion problem is polynomially solvable for the case of planer graphs, and is NP-hard when considering general graphs. In the rest of this chapter we focus on *minimum tree layout length* problem, formally defined as it follows.

Definition 108 (Minimum tree layout length). Consider the finite undirected graph $G = (\mathbf{V}, \mathbf{E})$. The minimum tree layout length problem is the problem of finding layout tree \mathbf{T} and a bijective mapping $\varphi : \mathbf{V}(G) \rightarrow \mathbf{V}_L(\mathbf{T})$ such that $L(\mathbf{T}, \varphi, G) = \sum_{\{u,v\} \in \mathbf{E}(G)} \lambda(uv, \mathbf{T}, \varphi, G)$ is minimized. \square

It is not hard to see that $\sum_{\{u,v\} \in \mathbf{E}(G)} \lambda(uv, \mathbf{T}, \varphi, G) = \sum_{\{x,y\} \in \mathbf{E}(\mathbf{T})} \delta(xv, \mathbf{T}, \varphi, G)$. Hence, in the rest of this chapter we may use them interchangeably.

In the context of communication graph embedding, the dilation of an edge $\{u, v\} \in \mathbf{E}(G)$ abstractly represent the communication delay between vertices u and v . Similarly the congestion of an edge $e \in \mathbf{E}(\mathbf{T})$ is a representative for the traffic on the physical link e . Hence tree length measure corresponds to the average delay between the vertices of G and also to the average edge congestion of the host tree.

6.3 Minimum Length of Tree Layout

In tree layout problem, as a variation of graph embedding problems, the underlying host graph is a tree \mathbf{T} where the degree of every node is either 1 or 3 and the vertices of G are being mapped to leaves of \mathbf{T} . In this section we study tree layout problem, where we show that the problem of finding a tree layout with minimum length for a given input graph G (Min Tree Length for short) is NP-hard for general (not necessarily simple) graphs.

6.3.1 Min Tree Length of Complete Graphs

Consider the complete graph $G = (\mathbf{V}, \mathbf{E})$ where $\forall u, v \in \mathbf{V}(G), \{u, v\} \in \mathbf{E}(G)$. It is not hard to see that a layout tree \mathbf{T} is a solution for the Min Tree Layout problem for G , iff $|\mathbf{V}_L(\mathbf{T})| = n$ (and hence $|\mathbf{V}(\mathbf{T})| = 2n - 1$), and the summation of distance of leaf nodes of \mathbf{T} is minimized. We denote the summation of distances of leaves of a tree \mathbf{T} by:

$$\sigma_{LL}(\mathbf{T}) = \frac{1}{2} \sum_{x,y \in \mathbf{V}_L(\mathbf{T})} d_{\mathbf{T}}(x,y)$$

Leaf to leaf distance summation measure σ_{LL} is very similar to the definition of *Wiener index* (proposed by chemist Wiener [97]), which is the summation of distances of all vertices of a given

graph G as represented in following equation.

$$\sigma(G) = \frac{1}{2} \sum_{u,v \in \mathbf{V}(G)} d_G(u,v)$$

Wiener index is widely studied both in mathematical and chemical literature. In [33] Fischermann *et al.* study Wiener index of trees. In this works authors represent the structure of the family of the trees that have minimum (or maximum) Wiener index among all the trees of the same order with maximum node degree $\Delta \geq 3$. Due to similarity of σ_{LL} measure and Wiener index, in the rest of this subsection we borrow some of the notations and definitions from [33] in order to study σ_{LL} for trees with maximum degree Δ .

Definition 109 ($\mathbb{T}(\mathcal{R}, \Delta)$ tree family). Consider integers Δ and $\mathcal{R} \in \{\Delta, \Delta - 1\}$. For a given $n \in \mathbb{N}$, the family $\mathbb{T}(\mathcal{R}, \Delta)$ of trees with n nodes has a unique member T up to isomorphism, defined using a planar embedding as it follows.

Let $M_k(\mathcal{R}, \Delta) \leq n < M_{k+1}(\mathcal{R}, \Delta)$ where:

$$M_k(\mathcal{R}, \Delta) = \begin{cases} 1 & k = 0 \\ 1 + \mathcal{R} + \mathcal{R} \times (\Delta - 1) + \dots + \mathcal{R} \times (\Delta - 1)^{k-1} & k \geq 1 \end{cases}$$

Figure 6.1 depicts the embedding of tree T with the following properties:

1. all nodes of T lie on some line \mathbf{L}_i for $0 \leq i \leq k + 1$
2. exactly one node v lies on the line \mathbf{L}_0 which has $\min\{n - 1, \mathcal{R}\}$ children on line \mathbf{L}_1
3. for $i = 1$ to $k - 1$ every node on line \mathbf{L}_i is connect to $\Delta - 1$ nodes on line \mathbf{L}_{i+1} and one node on line \mathbf{L}_{i-1}
4. the only line that may be incomplete, is line \mathbf{L}_{k+1} . Let $n - M_k(\mathcal{R}, \Delta) = m \times (\Delta - 1) + r$ for $0 \leq r < \Delta - 1$, where $n - M_k(\mathcal{R}, \Delta)$ is the number of remaining nodes on line \mathbf{L}_{k+1} . Also let $\{v_1, ..v_{m+1}\}$ be the set of m left most nodes on line \mathbf{L}_k where v_{m+1} is the right most one

in the set. Each of v_1, \dots, v_m nodes is connected to $\Delta - 1$ nodes on line \mathbf{L}_{k+1} , while v_{m+1} is connected to r nodes from line \mathbf{L}_{k+1} (see figure 6.1).

□

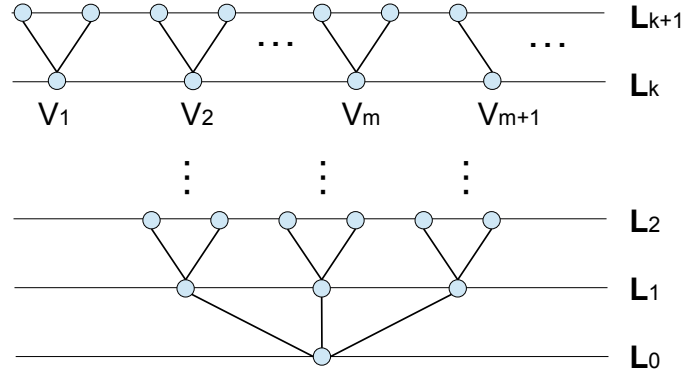


Figure 6.1: Planar embedding of T represented as the embedding of nodes on $k+2$ lines \mathbf{L}_i for $0 \leq i \leq k+1$.

We defined the family $T(\mathcal{R}, \Delta)$ for the general case of trees with maximum degree Δ , while we focus on the case where the degree of every internal node is $\Delta = 3$, but all the results presented in the rest of this subsection extend to all trees with arbitrary max degree $\Delta \geq 3$.

Lemma 110. *Consider tree $T \in T(\mathcal{R}, \Delta)$ of order n , and assume $M_k(\mathcal{R}, \Delta) < n < M_{k+1}(\mathcal{R}, \Delta)$ for $k \geq 1$. Let \bar{T} be an arbitrary tree of order n constructed from tree $\bar{T}_0 \in T(\mathcal{R}, \Delta)$, with $M_k(\mathcal{R}, \Delta)$ nodes, by attaching $n - M_k(\mathcal{R}, \Delta)$ nodes to the leaf nodes of \bar{T}_0 (which lie on the line \mathbf{L}_k). Then it is the case that either $\sigma_{LL}(\bar{T}) > \sigma_{LL}(T)$ or \bar{T} is isomorphic with T .*

Proof. We proof this lemma using induction on the height of tree \bar{T} , when embedded on the plane as explained in definition 109. It is easy to check the correctness of theorem for trees of height 1 and 2. Assume tree \bar{T} of height k and tree $T \in T(\mathcal{R}, \Delta)$ are not isomorphic. Let $v \in \mathbf{V}(\bar{T})$ be the node on line \mathbf{L}_0 . Node v is connected to \mathcal{R} subtrees $\{\bar{T}_1, \dots, \bar{T}_{\mathcal{R}}\}$. Based on the assumption of induction every subtrees \bar{T}_i is a member of the family $T(\Delta - 1, \Delta)$ of height k or $k - 1$. Since \bar{T} and T are not isomorph, there are at least two subtrees \bar{T}_i and \bar{T}_j for $i \neq j$ where are incomplete on

line \mathbf{L}_k . Formally speaking $\exists 1 \leq i \neq j \leq \mathcal{R}, \bar{\mathbf{T}}_i, \bar{\mathbf{T}}_j \in \mathbf{T}(\Delta - 1, \Delta)$ and $|\mathbf{V}(\bar{\mathbf{T}}_i)| - M_{k-1}(\Delta - 1, \Delta) = r_i > 0 \wedge |\mathbf{V}(\bar{\mathbf{T}}_j)| - M_{k-1}(\Delta - 1, \Delta) = r_j > 0$. Without loss of generality we assume $i = 1$ and $j = 1$ and also $|\mathbf{V}(\bar{\mathbf{T}}_1)| \leq |\mathbf{V}(\bar{\mathbf{T}}_2)|$. Figure 6.2a abstractly represents tree $\bar{\mathbf{T}}$.

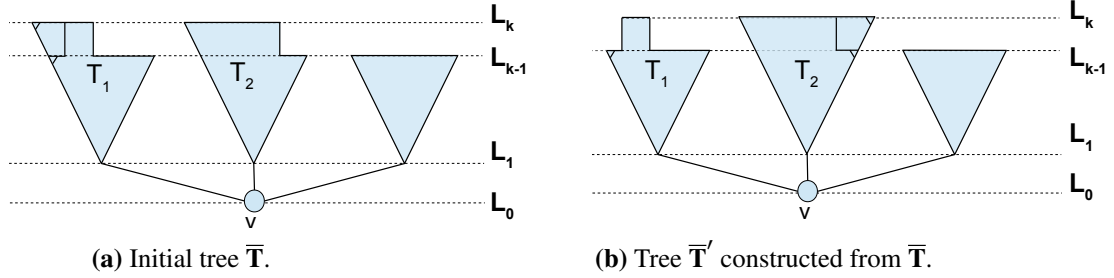


Figure 6.2: Figure 6.2a represents tree $\bar{\mathbf{T}} \notin \mathbf{T}(\mathcal{R}, \Delta)$ of height $k + 1$, constructed from tree $\bar{\mathbf{T}}_0 \in \mathbf{T}(\mathcal{R}, \Delta)$ of height k . Node v is connected to at least two subtrees $\bar{\mathbf{T}}_i, \bar{\mathbf{T}}_j \in \mathbf{T}(\mathcal{R}, \Delta)$ of height k which are incomplete on line \mathbf{L}_k . Alternative tree $\bar{\mathbf{T}}'$ depicted in figure 6.2b, is constructed by relocating some leaf nodes of subtree $\bar{\mathbf{T}}_1$ that are on line \mathbf{L}_k . As you see in this construction r_2 left most leaves of $\bar{\mathbf{T}}_1$ are relocated to complete subtree $\bar{\mathbf{T}}_2$. In this example the number of leaf nodes of $\bar{\mathbf{T}}_1$ on \mathbf{L}_k , is large enough to complete the subtree $\bar{\mathbf{T}}_2$.

Let $\mathbf{L}_l(\mathbf{T})$ denote the set of nodes of tree \mathbf{T} on line l . We present an alternative tree $\bar{\mathbf{T}}'$, by relocating some leaf nodes of $\mathbf{L}_k(\bar{\mathbf{T}}_1)$ (in order from left to right) to complete the line \mathbf{L}_k of $\bar{\mathbf{T}}_2$ (in order from right to left). Let $\mathcal{L} \subseteq \mathbf{L}_k(\bar{\mathbf{T}}_1)$ be the set of leaf nodes of $\bar{\mathbf{T}}_1$ on line \mathbf{L}_k , candidate for relocation. Figure 6.2b depicts tree $\bar{\mathbf{T}}'$ constructed from \mathbf{T} .

In the alternative tree $\bar{\mathbf{T}}'$, consider a bijective mapping φ_0 from the nodes of $\bar{\mathbf{T}}_1$ to nodes of subtree $\bar{\mathbf{T}}'_2$, where $\bar{\mathbf{T}}'_2$ is the modified version of $\bar{\mathbf{T}}_2$ in $\bar{\mathbf{T}}$. More specifically, mapping φ_0 is a reflection form $\bar{\mathbf{T}}_1$ to $\bar{\mathbf{T}}_2$, which reflects nodes of $\mathbf{L}_l(\bar{\mathbf{T}}_1)$ to the nodes of $\mathbf{L}_{l-1}(\bar{\mathbf{T}}'_2)$ for $2 \leq l \leq k$, such that the left most node on line \mathbf{L}_l of $\bar{\mathbf{T}}_1$ is mapped to the right most node of $\bar{\mathbf{T}}'_2$ on line \mathbf{L}_l . Accordingly φ_0 maps every leaf node of $\bar{\mathcal{L}} = \mathbf{L}_k(\bar{\mathbf{T}}_1) - \mathcal{L}$ (the set of remaining leaf nodes of $\bar{\mathbf{T}}_1$ on line \mathbf{L}_k) to one leaf node of $\bar{\mathbf{T}}_2$ on line \mathbf{L}_k . On the other hand every leaf node of $\mathbf{L}_{k-1}(\bar{\mathbf{T}}_1)$ is mapped to one node (leaf or internal) of $\bar{\mathbf{T}}_2$ on line \mathbf{L}_{k-1} .

From φ_0 we construct a bijective mapping φ from leaf nodes of $\bar{\mathbf{T}}_1$ to sets of leaf nodes of $\bar{\mathbf{T}}'_2$. For every $w \in \mathbf{V}_L(\mathbf{T})$, $\varphi(w) = \{\varphi_0(w)\}$ if $\varphi_0(w) \in \mathbf{V}_L(\bar{\mathbf{T}}'_2)$ is a leaf node node, otherwise ($\varphi_0(w)$ is an internal node of $\bar{\mathbf{T}}'_2$ on line \mathbf{L}_{k-1}), φ maps w to the set of direct children of $\varphi_0(w)$ on line \mathbf{L}_k ².

²In this case $|\varphi(w)| \leq \Delta - 1$.

Using the bijective mapping φ , we analyze the change in value of σ_{LL} in the process of constructing $\bar{\mathbf{T}}'$ from $\bar{\mathbf{T}}$ as it follows.

1. Clearly the internal summation of distances of nodes in \mathcal{L} stays unchanged.
2. For every leaf node $w_1 \in \mathcal{L}, w_2 \in \mathbf{V}_L(\mathbf{T}) - \mathbf{V}_L(\mathbf{T}_1) \uplus \mathbf{V}_L(\mathbf{T}_2)$, it is the case that $d_{\bar{\mathbf{T}}}(w_1, w_2) = d_{\bar{\mathbf{T}}'}(w_1, w_2)$. Hence, the summation of distances among nodes in \mathcal{L} and leaf nodes of $\mathbf{V}_L(\bar{\mathbf{T}}) - \mathbf{V}_L(\bar{\mathbf{T}}_1) \uplus \mathbf{V}_L(\bar{\mathbf{T}}_2)$ also does not change.
3. We show that for every $w_1 \in \mathcal{L}$ the summation of distances of w_1 from leaf nodes of $\mathbf{V}_L(\bar{\mathbf{T}}_2) \uplus \mathbf{V}_L(\bar{\mathbf{T}}_1) - \{w_1\}$ is greater than the summation of distances of $\varphi(w_1)$ from leaf nodes of $\mathbf{V}_L(\bar{\mathbf{T}}'_1) \uplus \mathbf{V}_L(\bar{\mathbf{T}}'_2) - \{\varphi w_1\}$.

Notation 111. Let $\mathcal{L}_1, \mathcal{L}_2 \subset \mathbf{V}(\mathbf{T})$, for an arbitrary tree \mathbf{T} . By $\sigma_{\mathbf{T}}(\mathcal{L}_1, \mathcal{L}_2)$ we denote the summation of distances of nodes of \mathcal{L}_1 and \mathcal{L}_2 . Formally speaking:

$$\sigma_{\mathbf{T}}(\mathcal{L}_1, \mathcal{L}_2) = \frac{1}{2} \sum_{v \in \mathcal{L}_1} \sum_{u \in \mathcal{L}_2} d_{\mathbf{T}}(v, u)$$

For every $w_2 \in \mathbf{V}_L(\bar{\mathbf{T}}_1) - \mathcal{L}$:

- If w_2 is on line \mathbf{L}_k , it is the case that $d_{\bar{\mathbf{T}}}(w_1, w_2) = d_{\bar{\mathbf{T}}'}(\varphi(w_1), \varphi(w_2))$. Therefore:

$$\sigma_{\bar{\mathbf{T}}}(\{w_1\}, \{w_2\} \uplus \varphi(w_2)) = \sigma_{\bar{\mathbf{T}}'}(\{\varphi(w_1)\}, \{w_2\} \uplus \varphi(w_2)) \quad (6.1)$$

- If w_2 is on line \mathbf{L}_{k-1} and $\varphi(w_2)$ maps w_2 to a leaf node of $\bar{\mathbf{T}}'_2$ on line \mathbf{L}_{k-1} , similar to previous case we have:

$$\sigma_{\bar{\mathbf{T}}}(\{w_1\}, \{w_2\} \uplus \varphi(w_2)) = \sigma_{\bar{\mathbf{T}}'}(\{\varphi(w_1)\}, \{w_2\} \uplus \varphi(w_2)) \quad (6.2)$$

- Otherwise w_2 is on line \mathbf{L}_{k-1} and $\varphi(w_2)$ maps w_2 to a non-empty set of leaf node of $\bar{\mathbf{T}}'_2$ on line \mathbf{L}_k . Assume the size of this set is $1 \leq \nabla \leq \Delta$. Since for some nodes w_2

where $|\varphi(w_2)| = \nabla = \Delta - 1 > 1$, then for such w_2 , the following equally holds:

$$\begin{aligned}
\sigma_{\bar{\mathbf{T}}}(\{w_1\}, \{w_2\} \uplus \varphi(w_2)) - \sigma_{\bar{\mathbf{T}}'}(\{\varphi(w_1)\}, \{w_2\} \uplus \varphi(w_2)) &= \quad (6.3) \\
\left(d_{\bar{\mathbf{T}}}(w_1, w_2) + 2k \times \nabla\right) - \left(2k + (d_{\bar{\mathbf{T}}'}(\varphi(w_1), \varphi_0(w_2)) + 1) \times \nabla\right) &= \\
\left(d_{\bar{\mathbf{T}}}(w_1, w_2) + 2k \times \nabla\right) - \left(2k + (d_{\bar{\mathbf{T}}}(w_1, w_2) + 1) \times \nabla\right) &= \\
(2k - d_{\bar{\mathbf{T}}}(w_1, w_2)) \times (\nabla - 1) - d_{\bar{\mathbf{T}}}(w_1, w_2) &\geq \\
2k - 2d_{\bar{\mathbf{T}}}(w_1, w_2) &> 0
\end{aligned}$$

Putting the results of previous cases and equations 6.1, 6.2 and 6.3, we conclude that $\sigma_{LL}(\bar{\mathbf{T}}) > \sigma_{LL}(\bar{\mathbf{T}}')$. If $\bar{\mathbf{T}}'_1 \notin \mathbf{T}(\mathcal{R}, \Delta)$, then based on the assumption of induction, replacing the subtree $\bar{\mathbf{T}}'_1$ with subtree $\bar{\mathbf{T}}''_1 \in \mathbf{T}(\mathcal{R}, \Delta)$ of the same order, results in tree $\bar{\mathbf{T}}''$, where $\sigma_{LL}(\bar{\mathbf{T}}'') < \sigma_{LL}(\bar{\mathbf{T}}')$.

Using the same approach and continuing with $\bar{\mathbf{T}}''$, in a sequence of leaf node relocations, we can construct the final tree $\tilde{\mathbf{T}}$, such that in $\tilde{\mathbf{T}}$, node v on line \mathbf{L}_0 has exactly one incomplete subtree $\tilde{\mathbf{T}}_i$ where $\tilde{\mathbf{T}}_i \in \mathbf{T}(\Delta - 1, \Delta)$. More specifically, $\forall 1 \leq l < i$, all leaf nodes of $\tilde{\mathbf{T}}_l$ lie on line \mathbf{L}_k and $\forall i < l \leq \Delta$, all leaf nodes of $\tilde{\mathbf{T}}_l$ lie on line \mathbf{L}_{k-1} , i.e. $\tilde{\mathbf{T}} \in \mathbf{T}(\mathcal{R}, \Delta)$. \square

Notation 112. Given an arbitrary tree \mathbf{T} , we define the planar line embedding of \mathbf{T} similar to the the approach in the definition 109. Starting from a designated $v \in \mathbf{V}(\mathbf{T})$, we embed \mathbf{T} on lines of plane, where v lies on line \mathbf{L}_0 and direct neighbours of v are placed on line \mathbf{L}_1 . Similarly all the nodes in distance d from v are placed on line \mathbf{L}_d . Also for $u \in \mathbf{V}(\mathbf{T})$ on line \mathbf{L}_l , the subtree rooted at u , where all its nodes are on lines $\mathbf{L}_{l'}$ for $l' \geq l$ is denoted by \mathbf{T}_u . Formally speaking $w \in \mathbf{V}(\mathbf{T}_u)$ iff u is on the shortest path from w to v on line \mathbf{L}_0 .

Notation 113. Consider a tree \mathbf{T} of order n and a node $v \in \mathbf{V}(\mathbf{T})$ of degree ∇ . Removing v form \mathbf{T} partitions \mathbf{T} into a set of subtrees $\{\mathcal{T}_1, \dots, \mathcal{T}_\nabla\}$. We call node v central if $\forall 1 \leq i \leq \nabla, |\mathbf{V}(\mathcal{T}_i)| \leq \frac{n}{2}$. The set of central nodes of tree \mathbf{T} is represented by $\mathcal{C}_{\mathbf{T}}$.

Theorem 114. Every arbitrary tree \mathbf{T} has at least one and at most two central nodes. In other word $1 \leq |\mathcal{C}_{\mathbf{T}}| \leq 2$.

For proof see [87]. Using this theorem, we proof the main result of this subsection as we present in the theorem 115.

Theorem 115. *Consider tree \mathbf{T} with maximum node degree Δ . where $\sigma_{LL}(\mathbf{T}) \leq \sigma_{LL}(\mathbf{T}')$ for every tree \mathbf{T}' (that $|\mathbf{V}_L(\mathbf{T}')| = |\mathbf{V}_L(\mathbf{T})|$). Then in the planar line embedding of \mathbf{T} with central node $v \in \mathcal{C}_{\mathbf{T}}$ on fixed line \mathbf{L}_0 , it is the case that:*

- $\mathbf{T} \in \mathbb{T}(\Delta, \Delta)$
- $\mathbf{T}_u \in \mathbb{T}(\Delta - 1, \Delta)$ for $u \neq v$

Proof. The proof of this theorem is carried out using induction on the height of planar line embedding of tree \mathbf{T} . It is not hard to check the correctness of theorem for trees of height 1 and 2. Let \mathbf{T} be a graph of height $h \geq 3$, and let u_1, \dots, u_{Δ} be the direct neighbours of v on line \mathbf{L}_0 and $\mathbf{T}_1, \dots, \mathbf{T}_{\Delta}$ respectively be their corresponding subtrees.

Case 1: $\exists w_1, w_2 \in \mathbf{V}_L(\mathbf{T}), d_{\mathbf{T}}(w_1, v) \geq d_{\mathbf{T}}(w_2, v) + 2$. Based on the assumption of induction w_1 and w_2 can not be on the same subtree. Without loss of the generality assume w_1 and w_2 are the two leaf nodes with maximum distance and $w_1 \in \mathbf{T}_1$ and $w_2 \in \mathbf{T}_2$. Let $\mathbf{T}_{11}, \mathbf{T}_{12}, \dots, \mathbf{T}_{1, \Delta-1} \subset \mathbf{T}_1$ be subtrees respectively with roots $u_{11}, \dots, u_{1, \Delta-1}$ connected to u_1 (nodes $u_{11}, \dots, u_{1, \Delta-1}$ lie on line \mathbf{L}_2). Also assume $w_1 \in \mathbf{V}_L(\mathbf{T}_{11})$.

Case 1.1: $|\mathbf{V}_L(\mathbf{T}_{11})| > |\mathbf{V}_L(\mathbf{T}_2)|$. We construct an alternative tree $\tilde{\mathbf{T}}$ by removing edges $\{u_1, u_{11}\}$ and $\{v, u_2\}$ and introducing two new edges $\{v, u_{11}\}$ and $\{u_1, u_2\}$. The structures of initial tree \mathbf{T} and the alternative tree $\tilde{\mathbf{T}}$ are represented in figure 6.3. One can verify that the following equation 6.4 correctly represents the relation between $\sigma_{LL}(\mathbf{T})$ and $\sigma_{LL}(\tilde{\mathbf{T}})$.

$$\begin{aligned} \sigma_{LL}(\mathbf{T}) - \sigma_{LL}(\tilde{\mathbf{T}}) = & \hspace{20em} (6.4) \\ & |\mathbf{V}_L(\mathbf{T}_{11})| \times \left(\sum_{2 \leq i \leq \Delta-1} |\mathbf{V}_L(\mathbf{T}_{1i})| - \sum_{3 \leq i \leq \Delta} |\mathbf{V}_L(\mathbf{T}_i)| \right) \\ & + |\mathbf{V}_L(\mathbf{T}_2)| \times \left(\sum_{3 \leq i \leq \Delta} |\mathbf{V}_L(\mathbf{T}_i)| - \sum_{2 \leq i \leq \Delta-1} |\mathbf{V}_L(\mathbf{T}_{1i})| \right) \end{aligned}$$

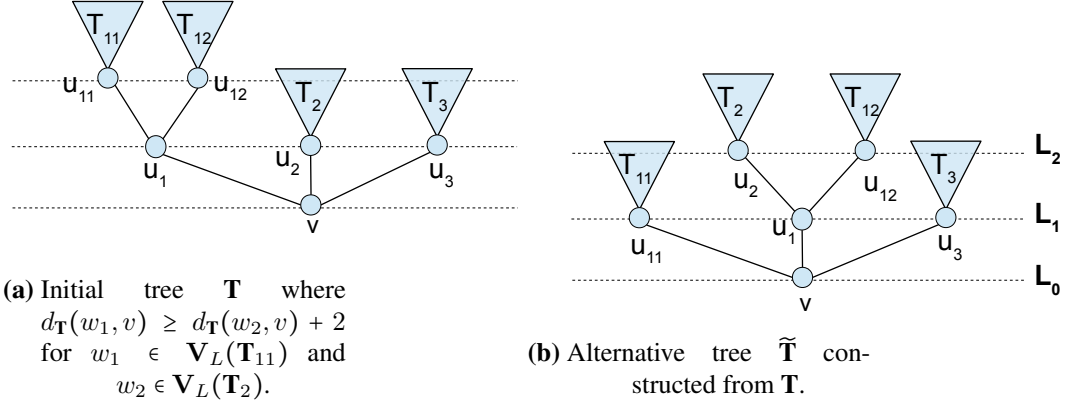


Figure 6.3: Representation of initial tree \mathbf{T} and its modified version $\tilde{\mathbf{T}}$.

$$= (|\mathbf{V}_L(\mathbf{T}_{11})| - |\mathbf{V}_L(\mathbf{T}_2)|) \times \left(\sum_{2 \leq i \leq \Delta-1} |\mathbf{V}_L(\mathbf{T}_{1i})| - \sum_{3 \leq i \leq \Delta} |\mathbf{V}_L(\mathbf{T}_i)| \right)$$

It is the case that $\sum_{2 \leq i \leq \Delta-1} |\mathbf{V}_L(\mathbf{T}_{1i})| < \sum_{3 \leq i \leq \Delta} |\mathbf{V}_L(\mathbf{T}_i)|$ otherwise it must be the case that $|\mathbf{V}(\mathbf{T}_1)| > \sum_{2 \leq i \leq \Delta} |\mathbf{V}(\mathbf{T}_i)| > \frac{|\mathbf{V}(\mathbf{T})|}{2}$, which is in contradiction with the centrality of node v . Therefore, $\sigma_{LL}(\mathbf{T}) - \sigma_{LL}(\tilde{\mathbf{T}}) > 0$, which contradicts the optimality of \mathbf{T} .

Case 1.2: $|\mathbf{V}_L(\mathbf{T}_{11})| \leq |\mathbf{V}_L(\mathbf{T}_2)|$. Hence, $d_{\mathbf{T}}(w_1, v) = d_{\mathbf{T}}(w_2, v) + 2$ and $w_1 \in \mathbf{V}_L(\mathbf{T}_{11})$ is located on line \mathbf{L}_k and $w_2 \in \mathbf{V}_L(\mathbf{T}_2)$ lies on line \mathbf{L}_{k-2} . Also non of subtrees \mathbf{T}_{11} and \mathbf{T}_2 can be complete respectively on lines \mathbf{L}_k and \mathbf{L}_{k-1} .

Let $\mathbf{L}_l(\mathbf{T})$ denote the set of nodes of tree \mathbf{T} on line l . Similar to the proof of lemma 110, we present an alternative tree $\tilde{\mathbf{T}}$, by relocating some leaf nodes of $\mathbf{L}_k(\mathbf{T}_{11})$ (in order from left to right) to complete the line \mathbf{L}_{k-1} of \mathbf{T}_2 (in order from right to left). Let $\mathcal{L} \subseteq \mathbf{L}_k(\mathbf{T}_{11})$ be the set of leaf nodes of \mathbf{T}_{11} on line \mathbf{L}_k , candidate for relocation. Based on an exact reasoning as in lemma 110 (case 3), which we omit, it can be inferred that the summation of distance of leaf nodes in \mathcal{L} from leaf nodes of $\mathbf{T}_{11} \uplus \mathbf{T}_2$ reduces going from \mathbf{T} to $\tilde{\mathbf{T}}$. Formally it can be deduced that:

$$\sigma_{\mathbf{T}}(\mathcal{L}, \mathbf{V}_L(\mathbf{T}_{11})) + \sigma_{\mathbf{T}}(\mathcal{L}, \mathbf{V}_L(\mathbf{T}_2)) > \sigma_{\tilde{\mathbf{T}}}(\tilde{\mathcal{L}}, \mathbf{V}_L(\tilde{\mathbf{T}}_{11})) + \sigma_{\tilde{\mathbf{T}}}(\tilde{\mathcal{L}}, \mathbf{V}_L(\tilde{\mathbf{T}}_2)) \quad (6.5)$$

Where $\tilde{\mathbf{T}}_{11}$ and $\tilde{\mathbf{T}}_2$ respectively correspond to \mathbf{T}_{11} and \mathbf{T}_2 after relocating leaf nodes of \mathcal{L} (repre-

sented by $\tilde{\mathcal{L}}$ in $\tilde{\mathbf{T}}$).

On the other hand, relocating \mathcal{L} , increases the distance of every leaf node in \mathcal{L} from leaf node of $\mathbf{T}_{12}, \dots, \mathbf{T}_{1,\Delta-1}$ by 1 unit, while it decreases the distance of every node of \mathcal{L} from every leaf node of $\mathbf{T}_3, \dots, \mathbf{T}_\Delta$. Since we assumed that $w_1 \in \mathbf{V}_L(\mathbf{T}_{11})$ and $w_2 \in \mathbf{V}_L(\mathbf{T}_2)$ have the maximum distance among all leaf nodes, then:

$$|\mathbf{V}_L(\mathbf{T}_{12})| + \dots + |\mathbf{V}_L(\mathbf{T}_{1,\Delta-1})| < |\mathbf{V}_L(\mathbf{T}_3)| + \dots + |\mathbf{V}_L(\mathbf{T}_\Delta)| \quad (6.6)$$

From equations 6.5 and 6.6 we conclude the following contradictory result:

$$\begin{aligned} \sigma_{LL}(\mathbf{T}) - \sigma_{LL}(\tilde{\mathbf{T}}) = & \quad (6.7) \\ & \sigma_{\mathbf{T}}(\mathcal{L}, \mathbf{V}_L(\mathbf{T}_{11})) - \sigma_{\tilde{\mathbf{T}}}(\tilde{\mathcal{L}}, \mathbf{V}_L(\tilde{\mathbf{T}}_2)) + \\ & \sigma_{\mathbf{T}}(\mathcal{L}, \mathbf{V}_L(\mathbf{T}_2)) - \sigma_{\tilde{\mathbf{T}}}(\tilde{\mathcal{L}}, \mathbf{V}_L(\tilde{\mathbf{T}}_{11})) + \\ & \sigma_{\mathbf{T}}(\mathcal{L}, \mathbf{V}_L(\mathbf{T}) - \mathbf{V}_L(\mathbf{T}_{11}) \uplus \mathbf{V}_L(\mathbf{T}_2)) - \sigma_{\tilde{\mathbf{T}}}(\tilde{\mathcal{L}}, \mathbf{V}_L(\tilde{\mathbf{T}}) - \mathbf{V}_L(\tilde{\mathbf{T}}_{11}) \uplus \mathbf{V}_L(\tilde{\mathbf{T}}_2)) \\ & > 0 \end{aligned}$$

Case 2: $\forall w_1, w_2 \in \mathbf{V}_L(\mathbf{T}), |d_{\mathbf{T}}(w_1, v) - d_{\mathbf{T}}(w_2, v)| < 2$. Since based on the assumption of induction $\mathbf{T}_1, \dots, \mathbf{T}_\Delta \in \mathbf{T}(\Delta - 1, \Delta)$, then \mathbf{T} can be constructed from some tree $\mathbf{T}_0 \in \mathbf{T}(\mathcal{R}, \Delta)$ of order $|\mathbf{V}(\mathbf{T}_0)| = M_k(\mathcal{R}, \Delta)$, by attaching $n - M_k(\mathcal{R}, \Delta)$ nodes to the leaf nodes of \mathbf{T}_0 . Therefore, based on lemma 110, \mathbf{T} is optimal iff $\mathbf{T} \in \mathbf{T}(\Delta, \Delta)$. \square

Corollary 116. Consider the complete graph G of order $n = |\mathbf{V}(G)|$. Tree \mathbf{T} is an optimal tree layout for G iff $|\mathbf{V}_L(\mathbf{T})| = n$ and $\mathbf{T} \in \mathbf{T}(3, 3)$.

Example 117. Let G be a complete graph of order $n = 2^l$ for some $l > 1$. Based on the result of theorem 115, a layout tree \mathbf{T} for G (of order $2n - 1$) has minimum value σ_{LL} (and accordingly is a solution for Min Layout Length) iff it is isomorphic to some tree with a structure similar to the tree in figure 6.4.

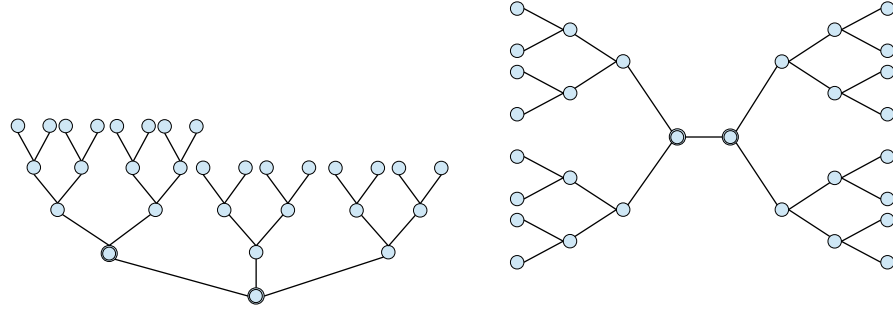


Figure 6.4: Two planar embeddings of an optimal layout tree \mathbf{T} of size $2n-1$, corresponding to the complete graph G with $n = 2^l$ vertices.

6.3.2 Min Tree Length of General Graphs

In the context of communication tree embedding, the direct connections of the input graphs can be weighted. Hence, the input graph is not necessary simple. Accordingly, in this chapter we always assume that the input graph is a general multi-graph. Graph G is a multi-graph if either it is a simple undirected graph, or it can be constructed from a simple undirected graph by adding parallel, undirected edges. In our main result of this section we show that the Min Tree Length problem is NP-hard for class of multi-graphs.

Theorem 118. *Min Tree Length problem is NP-hard for the class of multi-graphs.*

Proof. The correctness of the theorem can be represented using a polynomial reduction from Equal-Size 4-CliqueCover problem, which is shown to be NP-Complete in Lemma 100

Consider an arbitrary graph G , as an instance input of Equal-Size 4-CliqueCover problem, where $|\mathbf{V}(G)| = 2^l$ for some $l \in \mathbb{N}$. Let G' be the multi-graph obtained from G by introducing $M = m \times (2n - 2)$ parallel edges between every two vertices $u, v \in \mathbf{V}(G)$. Notice that every tree layout \mathbf{T} for graph G has $2n - 2$ edges where the congestion of each edge is less than $m = |\mathbf{E}(G)|$. Considering graph G as an vertex induced subgraph of G' , then $G' = G \uplus \tilde{G}$, where \tilde{G} is another vertex induced subgraph of G' . Subgraph \tilde{G} is complete multi-graph. Hence for every tree layout

\mathbf{T} and $\varphi : \mathbf{V}(G) \rightarrow \mathbf{V}_L(\mathbf{T})$ for G' we have:

$$L(\mathbf{T}, \varphi, G') = L(\mathbf{T}, G) + L(\mathbf{T}, \varphi, \tilde{G}) \quad (6.8)$$

From corollary 116 we know that a layout tree $\tilde{\mathbf{T}}$ for \tilde{G} is optimal iff $\tilde{\mathbf{T}} \in \mathbf{T}(3, 3)$. Also for every layout tree $\mathbf{T} \notin \mathbf{T}(3, 3)$, it is the case that $L(\mathbf{T}, \tilde{G}) \geq L(\tilde{\mathbf{T}}, \tilde{G}) + M$. On the other hand for $n > 2$ and every layout tree \mathbf{T} where $|\mathbf{V}_L(\mathbf{T})| = n$, it is always the case that $L(\mathbf{T}, G) < M$.

Therefore, a layout tree \mathbf{T} for G' is optimal iff \mathbf{T} and $\tilde{\mathbf{T}}$ are isomorphic. In other words \mathbf{T} for G' is optimal iff $\mathbf{T} \in \mathbf{T}(3, 3)$. Hence the Min Tree Length problem for G' reduces to the problem of finding an optimal bijection φ from vertices of G to leaf nodes of $\tilde{\mathbf{T}} \in \mathbf{T}(3, 3)$, such that the summation of edge dilations for all $\{u, v\} \in \mathbf{E}(G)$ is minimized. Formally speaking:

$$\arg \min_{(\mathbf{T}, \varphi)} L(\mathbf{T}, \varphi, G') = (\tilde{\mathbf{T}}, \arg \min_{\varphi} L(\tilde{\mathbf{T}}, \varphi, G)) \quad (6.9)$$

Let \overline{G} denote the complement of graph G . One can easily check that:

$$L(\tilde{\mathbf{T}}, \varphi, G) = L(\tilde{\mathbf{T}}, \varphi, K_n) - L(\tilde{\mathbf{T}}, \varphi, \overline{G}) \quad (6.10)$$

Where K_n is a complete graph of size n . Therefore:

$$\arg \min_{\varphi} L(\tilde{\mathbf{T}}, \varphi, G) = \arg \max_{\varphi} L(\tilde{\mathbf{T}}, \varphi, G) = \arg \max_{\varphi} \sum_{\{u, v\} \in \mathbf{E}(\overline{G})} \lambda(uv, \tilde{\mathbf{T}}, \varphi, G) \quad (6.11)$$

Also based on the structure of $\tilde{\mathbf{T}}$, $\exists c_1, c_2 \in \mathcal{C}_{\tilde{\mathbf{T}}}$ (in figure 6.4 shown by double lined circles). Since $n = 2^l$ for $l \in \mathbb{N}$, removing central nodes c_1 and c_2 partition leaf nodes of $\tilde{\mathbf{T}}$ into 4 equally sized partitions L_1, \dots, L_4 .

Finally, we can deduce that the original graph G can be partitioned into 4 complete sub-graphs of size $\frac{n}{4}$, iff there exist bijection φ such that $\forall \{u, v\} \in \mathbf{E}(\overline{G}), \varphi(u) \in L_i \wedge \varphi(v) \in L_j$ for $1 \leq i \neq j \leq 4$. In other words, graph G can be partitioned into 4 complete sub-graphs G_1, \dots, G_4 of size $\frac{n}{4}$, iff $(\tilde{\mathbf{T}}, \varphi)$ is a solution for Min Tree Length of G' , where φ maps vertices of G_i to leaf nodes of L_i for

$1 \leq i \neq j \leq 4$. Which infers the NP-hardness of Min Tree Length problem for the multi-graphs. \square

6.4 Layout Tree Problem in Relation with Graph Reassembling

The α -optimal reassembling problem for graph G is the problem of finding a *rooted* binary tree \mathcal{B} and a bijective mapping from vertices of G to leaf node of \mathcal{B} , such that the maximum edge congestion of \mathcal{B} is minimized. Similarly the β -optimal reassembling problem is the problem of finding a *rooted* binary tree \mathcal{B} and a bijective mapping from vertices of G to leaf node of \mathcal{B} , where tree length of \mathcal{B} is minimized. It is easy to see that all the result for the minimum tree layout congestion problem can be directly inferred for the case where the underlying tree is rooted. On the other hand the same statement can not immediately be inferred for minimum tree layout length problem. Therefore, in this section we study the Min Tree Length problem where the host graph is a rooted binary tree (Min Rooted Tree Length problem for short).

Lemma 119. *Min Tree Length problem is NP-hard for the class of graphs $\mathcal{G}_{\nabla=1}$ where for every member $G \in \mathcal{G}_{\nabla=1}$, G is connected and exists $v \in \mathbf{V}(G)$ of degree 1 (i.e. $\mathcal{G}_{\nabla=1}$ is a the class of graphs with min degree $\nabla = 1$).*

Proof. An immediate result of the approach that is used in proof of the Theorem 118 is that the Min Tree Length problem stays NP-hard for the class of *congested graphs*. A graph G with minimum vertex degree ∇ is called congested if for every tree layout \mathbf{T} for G it is the case $\sigma(e, \mathbf{T}, G) \geq \nabla$ for every edge $e \in \mathbf{E}(\mathbf{T})$.³

Consider congested graph G with min degree ∇ . if $\nabla = 1$ we are done, otherwise let $v \in \mathbf{V}(G)$ be a vertex with degree ∇ and G' be the graph constructed by augmenting G with a new vertex u and edge $\{u, v\}$. Also let tree layout \mathbf{T}' and mapping φ' be a solution for Min Tree Length problem of G' where $\{\varphi'(v), w\}$ is edge incident to leaf node $\varphi'(v)$ (with congestion ∇). Is not hard to verify that it must be the case that $\{\varphi'(u), w\} \in \mathbf{E}_{\mathbf{E}}(\mathbf{T}')$.

Let \mathbf{T} be a layout tree for G , obtained from \mathbf{T}' after removing vertices $\varphi'(u)$ and w (and their incident edges) and introducing edge $\{\varphi'(v), w'\}$. Where w' is the third neighbor of w in \mathbf{T}' .

³The graphs that are used in both proof are clearly congested graphs.

Hence the following equality holds.

$$LA(\mathbf{T}', \varphi', G') = LA(\mathbf{T}, \varphi, G) + 2$$

Where $\varphi(v) = \varphi'(v)$ for every $v \in \mathbf{V}(G)$.

Claim 120. *Tree layout \mathbf{T} and along side with the bijective mapping φ is a solution for Min Tree Length problem of G .*

Assume there exist Tree layout $\tilde{\mathbf{T}}$ and bijective mapping $\tilde{\varphi}$ such that $LA(\tilde{\mathbf{T}}, \tilde{\varphi}, G) < LA(\mathbf{T}, \varphi, G)$. Hence one can construct a tree layout $\tilde{\mathbf{T}}'$ and bijective mapping $\tilde{\varphi}'$ for G' by subdividing the edge incident to leaf node $\tilde{\varphi}(v)$ and introducing internal node w which is directly connected to leaf node $\tilde{\varphi}'(u)$. Therefore:

$$LA(\tilde{\mathbf{T}}', \tilde{\varphi}', G') = LA(\tilde{\mathbf{T}}, \tilde{\varphi}, G) + 2 < LA(\mathbf{T}', \varphi', G')$$

Which contradicts the assumption that \mathbf{T}' and φ' are a solution for Min Tree Length problem of G' . □

Using the result of lemma 119 it can be shown that Min Rooted Tree Length is NP-hard for the class of non necessary connected graphs. Later on we extend the this result to the class of connected graphs.

Theorem 121. *Consider the finite undirected graph G (non necessarily connected). The problem of finding a bijective mapping from the vertices of G to some rooted binary tree \mathbf{T} with minimum length is not polynomially solvable unless $P=NP$.*

Proof. We proof this theorem using the NP-hardness of Min Tree Length problem for the class of $\mathcal{G}_{\nabla=1}$. Hence given a graph $G \in \mathcal{G}_{\nabla=1}$, we obtain a disconnected graph $\overline{G} = G \uplus G_0$ where $G_0 = (\{v\}, \{\})$ is a one-vertex graph. Assume rooted binary tree $\overline{\mathbf{T}}$ and bijective mapping $\overline{\varphi}$ are the solution for the Min Rooted Tree Length problem of the augmented graph \overline{G} .

Claim 122. (I) $w_1 = \overline{\varphi}(v)$ is directly connected to the root node r of $\overline{\mathbf{T}}$, and

(II) let w_2 be the second direct neighbor of the root of $\bar{\mathbf{T}}$. Node w_2 subdivides an edge (or equivalently, is connected to two edges) with congestion 1. Figure 6.5 depicts the claimed structure of $\bar{\mathbf{T}}$.

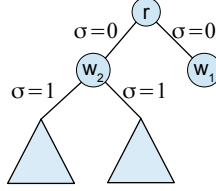


Figure 6.5: The structure of rooted tree layout for augmented graph \bar{G} .

Proof of I. Assume the opposite holds. Therefore, $w_1 = \bar{\varphi}(v)$ is directly connected to an internal node of $w' \in \mathbf{V}_I(\bar{\mathbf{T}})$, where w' is incident to two edge $e_1, e_2 \in \mathbf{E}(\bar{\mathbf{T}})$ such that $\sigma(e_1, \bar{\mathbf{T}}, \bar{\varphi}, G) = \sigma(e_2, \bar{\mathbf{T}}, \bar{\varphi}, G) > 0$. We construct an alternative rooted tree layout $\bar{\mathbf{T}}'$ from $\bar{\mathbf{T}}$ as described in following. We remove edge $\{w_1, w'\}$ and w' and introducing a new edge, replacing e_1 and e_2 . Also we introduce a new root node r' and connect it to w_1 and r . Based on the construction process of $\bar{\mathbf{T}}'$, it is the case that $LA(\bar{\mathbf{T}}, \bar{G}) - LA(\bar{\mathbf{T}}', \bar{G}) = \sigma(e_2, \bar{\mathbf{T}}, \bar{\varphi}, \bar{G}) > 0$, which contradicts the assumption of optimality of $\bar{\mathbf{T}}$.

Proof of II. Clearly exists edge $e \in \mathbf{E}(\bar{\mathbf{T}})$ such that $\sigma(e, \bar{\mathbf{T}}, \bar{\varphi}, \bar{G}) = 1$. Hence assuming that w_2 subdivides an edge with congestion greater than 1, similar to the approach in the proof of II, one can obtain an alternative rooted tree layout $\bar{\mathbf{T}}'$ with the contradictory property $LA(\bar{\mathbf{T}}', \bar{G}) < LA(\bar{\mathbf{T}}, \bar{G})$.

Claim 123. Graph $G \in \mathcal{G}_{\nabla=1}$ has tree length less than k , iff the augmented graph \bar{G} has a rooted tree layout with tree length less than $k + 1$. More specifically given an rooted tree layout $\bar{\mathbf{T}}$ and bijective mapping $\bar{\varphi}$ for augmented tree \bar{G} (with the structure presented in claim 122 and figure 6.5), removing nodes r, w_1 and w_2 and joining the two edges of congestion 1 incident to w_2 , obtains an optimal tree layout \mathbf{T} and mapping φ for graph G (where $\varphi(u) = \bar{\varphi}(u)$ for every $u \in \mathbf{V}(G)$).

Proof. It is easy to see that $LA(\bar{\mathbf{T}}, \bar{G}) = LA(\mathbf{T}, G) + 1$. Now Assume there exist a tree layout \mathbf{T}' and bijective mapping φ' for G such that $LA(\mathbf{T}', \varphi', G) < LA(\mathbf{T}, \varphi, G)$. On the other hand for

every tree layout \mathbf{T}' for G , exists $e \in \mathbf{E}(\mathbf{T}')$ where $\sigma(e, \mathbf{T}', G) = 1$. From \mathbf{T}' (and φ'), an alternative rooted tree layout $\bar{\mathbf{T}}'$ (and bijective mapping $\bar{\varphi}'$) can be obtained by introducing three new nodes, node r designated as the root of $\bar{\mathbf{T}}'$, leaf node w_1 , directly connected to r (where $\bar{\varphi}'(w_1) = v$) and internal node w_2 directly connected to r which subdivides edge e . It can be verified that $LA(\bar{\mathbf{T}}', \bar{\varphi}', \bar{G}) = LA(\mathbf{T}', \varphi', G) + 1$. Hence the following contradictory result concludes the proof of this theorem:

$$LA(\bar{\mathbf{T}}', \bar{\varphi}', \bar{G}) < LA(\mathbf{T}, \varphi, G) + 1 = LA(\bar{\mathbf{T}}, \bar{\varphi}, \bar{G})$$

□

6.5 Tree Length of Routing Trees

In the previous sections we focused on the problem of embedding vertices of an input graph G into leaf nodes of a host tree \mathbf{T} , where the degree of every internal node of \mathbf{T} is 3, known as tree layout problem. In this section we extend some results to the general routing tree problems. In this problem the vertices of the source graph G are being embedded into the leaf nodes of some communication tree \mathbf{T} with fixed maximum degree Δ .

Definition 124 (*Minimum Routing Tree Length*). Given graph G and integer Δ , Minimum Routing Tree Length problem (Min Routing Length for short) is the problem of finding tree \mathbf{T} with maximum degree Δ and a bijective mapping $\varphi : \mathbf{V}(G) \rightarrow \mathbf{V}_L(\mathbf{T})$, such that $LA(\mathbf{T}, \varphi, G)$ is minimized.

□

Proof of our final result on Min Routing Length problem is built on some intermediate result as presented in what follows.

Definition 125 (*Fixed Size k -Clique Cover*). Consider graph $G = (\mathbf{V}, \mathbf{E})$ and k positive integers n_1, \dots, n_k where $\sum_{1 \leq i \leq k} n_i = n$. Fixed Size k -Clique Cover problem is the problem of partitioning \mathbf{V} into k disjoint subsets $\mathbf{V}_1, \dots, \mathbf{V}_k$ s.t. $G(\mathbf{V}_i)$ is a clique of size n_i , for $1 \leq i \leq k$. □

Lemma 126. *Fixed Size k -Clique Cover is NP-complete.*

Proof. Similar to the proof of NP-completeness of Equal-Size 4-CliqueCover problem in Lemma 100 and using general graph k -colorability problem. \square

Definition 127 (*Equal Size k -Clique Cover*). Given graph $G = (\mathbf{V}, \mathbf{E})$ where $|V| = k \times l$ for some $l \in \mathbb{N}$, Equal Size k -Clique Cover problem is the problem of partitioning \mathbf{V} into k disjoint subsets $\mathbf{V}_1, \dots, \mathbf{V}_k$ s.t. $G(\mathbf{V}_i)$ is a clique of size $\frac{n}{k}$, for $1 \leq i \leq k$. \square

Lemma 128. *Equal Size k -Clique Cover problem is NP-complete.*

Proof. Similar to the proof of NP-completeness of Equal Size k -Clique Cover problem. \square

Consider the class of graphs where for every graph G in this class, $\exists l \in \mathbb{N}$ such that $|\mathbf{V}(G)| = k \times (k-1)^l$. Equal Size k -Clique Cover problem stays NP-complete for this class. Concisely in what follows, a polynomial reduction from Equal Size k -Clique Cover problem for general graphs is presented. Given graph $G = (\mathbf{V}, \mathbf{E})$ where $|V| = k \times n'$ for $n' \in \mathbb{N}$, one can obtain graph G' by augmenting G with k complete components C_1, \dots, C_k of size $(k-1)^l - n'$, where l is the smallest integer such that $(k-1)^l \geq n'$. Also in G' every newly introduces vertex $v \in \mathbf{V}(C_1) \uplus \dots \uplus \mathbf{V}(C_k)$ is connected to every vertex $u \in \mathbf{V}(G)$. It is not hard to check that $|\mathbf{V}(G')| = k \times (k-1)^l$ and more importantly G is a positive instance of Equal Size k -Clique Cover problem iff G' is a positive instance of Equal Size k -Clique Cover problem.

In the rest of this section we only consider graph of order $|\mathbf{V}(G)| = k \times (k-1)^l$. Obviously all the harness results for this class immediately extend to the class of general sized graphs.

Theorem 129. *Given multi-graph G and integer Δ , the problem of finding a routing tree \mathbf{T} and bijective mapping $\varphi : \mathbf{V}(G) \rightarrow \mathbf{V}_L(\mathbf{T})$ with minimum tree length is NP-hard.*

Proof. Similar to the proof of theorem 118, it can be shown that the Equal Size k -Clique Cover problem is not harder than Minimum Routing Tree Length problem.

Hence, consider graph G as the input of the Equal Size k -Clique Cover problem where $|\mathbf{V}(G)| = k \times (k-1)^l$ for some $l \in \mathbb{N}$. Let G' be the multi-graph obtained from G by introducing $M = m \times (2n-2)$ parallel edges between every two vertices $u, v \in \mathbf{V}(G)$. Therefore, $G' = G \uplus \tilde{G}$, where complete multi-graph \tilde{G} , is a vertex induced subgraph of G' . Using similar reasoning as in the

proof of theorem 118, \tilde{G} dictates the structure of optimal routing tree for G' . In other words, the problem of finding optimal routing tree \mathbf{T} and mapping φ for G' reduces to the problem of finding mapping φ from vertices of original graph G to the leaf nodes of a fixed-structure tree \mathbf{T} such that $LA(\mathbf{T}, \varphi, G)$ is minimized. Based on corollary 116 $\mathbf{T} \in \mathbf{T}(k, k)$.

Removing the only central node of $\mathbf{T} \in \mathbf{T}(k, k)$ partitions \mathbf{T} into k subtrees $\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_k \in \mathbf{T}(k, k)$ of the same order and $(k-1)^l$ leaf nodes. As explained with more details in the proof of theorem 118, it can be inferred that G is a positive instance of the Equal Size k -Clique Cover problem (in other words vertices of G can be partitioned into k equal sized complete sub-graphs G_1, G_2, \dots, G_k) iff given routing tree \mathbf{T} and mapping φ as the solution for Min Routing Length problem (with $\Delta = k$), φ maps vertices of G_i to leaf nodes of \mathcal{T}_i for $1 \leq i \leq k$.

□

6.6 Conclusion and Future Work

In this section we investigated the relation between “tree layout problem” and graph reassembling problem. Every reassembling sequence for graph G correspond to a unique binary reassembling tree. Hence, the graph reassembling problem can be translated into tree layout problem as a special case of graph embedding problems. Given an input graph G , the goal in tree layout problem is to find a binary tree \mathbf{T} and a bijective mapping from the set of vertices of G to the leaf nodes of \mathbf{T} . Regarding the tree layout problem, there are different measures defined to be optimized. In this chapter show that our previously define β measure is tightly related to “tree length” measure, as is defined. Initially we showed that tree layout problem (finding a tree layout \mathbf{T} for arbitrary graph G) is NP-hard, when trying to minimize the tree length of the binary tree \mathbf{T} . Finally, we used this result to show the NP-hardness of β -optimal graph reassembling problem for general graphs.

Future work related to tree layout problem is similar to that of the graph reassembling problem as state in Chapter 5, *i.e.* it includes: (1) the study of classes of graphs for which tree layout problem (with respect to different optimization goals) can be carried out in low-degree polynomial times, (2) investigating the properties of classes of graphs with small optimal tree length values (relative

to the size of graph) and (3) the study of approximability of this problem and therefore finding approximation algorithms for it.

Chapter 7

Conclusion

Many large-scale systems can be considered as interconnected sub-networks communicating via flows of some sort, where each Sub-system can be consumer and/or producer of flow of other(s). The traditional design and implementation of flow networks provides analysis and certification of desirable invariants of the system, provided having the complete view of the system as a whole network. Analysis and verification of different properties across such large-scale systems can quickly grow costly and get intractable as the size and complexity of a model increases. As an alternative solution, in this thesis we studied the "theory of network typings" as a solution for a compositional, algebraic and polyhedral analysis of a more general class of flow networks, called additive flow network. The background to this work is a group effort to develop an integrated framework suitable for compositional analysis of flow networks (in the standard definition) that is simultaneously: modular, incremental, and order-oblivious. These are the three defining properties of what we call a seamlessly compositional approach to system modeling and system analysis. Several previous works explain the methodology behind this environment for standard flow networks, as well as its state of development and implementation which can be found in [14, 15, 16, 59, 60, 62, 63, 88].

In this thesis we covered an augmented variation of this framework which enables the compositional analysis of a more general definition of flow networks, *i.e.* additive flow network. The central concept of typings framework is an abstract representation of flow information of a flow network (or sub-network) \mathcal{N} using what we call a network typing. Given a network \mathcal{N} , with multiple input edges and multiple output edges, a typing for \mathcal{N} is a formal algebraic characterization of flow assignments to its set of input/output edges. A typing for \mathcal{N} is *principal* if it encompasses exactly all the feasible flows information in \mathcal{N} restricted to its input/output edges, including, in particu-

lar, all maximal feasible flows (*i.e.* maximum in/out flows). In addition to the presenting necessary notions, definitions and theorems on the typings framework for additive flow networks, we also formally present an efficient approach for finding a principal typing for an arbitrary flow network \mathcal{N} when the underlying graph of \mathcal{N} is k -outerplanar (where k is a small and constant integer value relative to the size of \mathcal{N}).

Separately in Chapter 4, we studied maximum in/out flow problems in additive flow networks and it is showed maximum in/out flow problems are NP-hard tasks for the class of networks with underlying planar graphs.

The theory of network typings gives rise to problems of polyhedral analysis and different forms of graph decomposition. Some of these problems are new and still to be examined, others are reducible or equivalent to problems already resolved or partially resolved by other researchers. Among those, in this thesis we put our focus on one problem, which we call the "graph reassembling" problem. We studied the complexity of different variations of graph reassembling problem (with respect to minimizing α or β measures) and also their relation with problems of graph embedding, studied by other researchers such as Linear Arrangement, Routing Tree Embedding, and Tree Layout. In Chapter 5 we presented different variations of graph reassembling problems such as "linear reassembling" and "balanced reassembling". In this chapter we showed that, with the goal of minimizing α or β measures, both these variations are NP-hard problems and reducible to different well-known optimization problems.

Finally, in Chapter 6 it is shown that the graph reassembling problem can be translated into a special cases of graph embedding problem, in the sense that, given an input graph G , the goal is to find a binary tree \mathcal{B} and a bijective mapping from the set of vertices of G to the leaf nodes of \mathcal{B} . Thus, graph reassembling problem is tightly related to a graph embedding problem, called "tree layout problem". Tree layout problem concerns the task of embedding the vertices of an input graph G into the leaf nodes of some binary tree \mathbf{T} . In the context of tree layout problem, we investigated the less investigate measure to be optimized, called "tree length", which is a representative for average "edge dilation" measure and average "edge congestion" measure. We show that finding a tree layout \mathbf{T} for an arbitrary graph G with minimum tree length is an NP-hard task.

Bibliography

- [1] D Adolphson and T Ch Hu. Optimal linear ordering. *SIAM Journal on Applied Mathematics*, 25(3):403–423, 1973. 10
- [2] R.K. Ahuja, T. L. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, N.J., 1993. 63
- [3] Martin Aigner and Günter M Ziegler. Three applications of eulers formula. In *Proofs from THE BOOK*, pages 57–62. Springer, 1998. 42
- [4] George E. Andrews. *The Theory of Partitions (Encyclopedia of Mathematics and its Applications)*. Cambridge University Press, 1998. 129
- [5] Stefan Arnborg, Derek G Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in ak-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987. 144
- [6] Stefan Arnborg and Andrzej Proskurowski. Characterization and recognition of partial 3-trees. *SIAM Journal on Algebraic Discrete Methods*, 7(2):305–314, 1986. 12
- [7] Sanjeev Arora, Alan Frieze, and Haim Kaplan. A new rounding procedure for the assignment problem with applications to dense graph arrangement problems. In *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on*, pages 21–30. IEEE, 1996. 11, 145
- [8] Takao Asano and Yasuhito Asano. Recent Developments in Maximum Flow Algorithms. *Journal of the Operations Research Society of Japan*, 43(1), March 2000. 14
- [9] Hakan Aydin, Rami Melhem, Daniel Mossé, and Pedro Mejía-Alvarez. Determining optimal processor speeds for periodic real-time tasks with different power characteristics. In *Real-Time Systems, 13th Euromicro Conference on, 2001.*, pages 225–232. IEEE, 2001. 7
- [10] Roberto Bagnara, Patricia Hill, and Enea Zaffanella. A New Encoding of Not Necessarily Closed Convex Polyhedra, Extended Abstract. In *Proc. of Implementation Technology for Computational Logic Systems*, Madrid, Spain, September 2002. Available at <https://cliplab.org/Conferences/Colognet/ITCLS-2002/PAPERS/RobertoBagnara.pdf>. 15
- [11] Roberto Bagnara, Patricia Hill, and Enea Zaffanella. A New Encoding and Implementation of Not Necessarily Closed Convex Polyhedra. In M. Leuschel, S. Gruner, and S. Lo Presti, editors, *Proc. of 3rd Workshop on Automated Verification of Critical Systems*, pages 161–176, 2003. Available at <http://bugseng.com/products/pp1/documentation/BagnaraHZ03a.pdf>. 15

- [12] Roberto Bagnara, Patricia Hill, and Enea Zaffanella. Not Necessarily Closed Convex Polyhedra and the Double Description Method. *Formal Aspects of Computing*, 17:222–257, 2005. [15](#)
- [13] Rémy Belmonte, Pim vant Hof, Marcin Kamiński, Daniël Paulusma, and Dimitrios M Thilikos. Characterizing graphs of small carving-width. In *Combinatorial Optimization and Applications*, pages 360–370. Springer, 2012. [12](#), [144](#)
- [14] A. Bestavros, A. Kfoury, A. Lapets, and M. Ocean. Safe Compositional Network Sketches: Tool and Use Cases. In *IEEE Workshop on Compositional Theory and Technology for Real-Time Embedded Systems*, Wash D.C., December 2009. [2](#), [8](#), [14](#), [167](#)
- [15] A. Bestavros, A. Kfoury, A. Lapets, and M. Ocean. Safe Compositional Network Sketches: The Formal Framework. In *13th ACM HSCC*, Stockholm, April 2010. [2](#), [8](#), [14](#), [167](#)
- [16] Azer Bestavros and Assaf Kfoury. A Domain-Specific Language for Incremental and Modular Design of Large-Scale Verifiably-Safe Flow Networks. In *Proc. of IFIP Working Conference on Domain-Specific Languages (DSL 2011), EPTCS Volume 66*, pages 24–47, Sept 2011. [2](#), [8](#), [9](#), [14](#), [167](#)
- [17] Dan Bienstock. On embedding graphs in trees. *Journal of Combinatorial Theory, Series B*, 49(1):103–136, 1990. [144](#)
- [18] Hans L Bodlaender. A tourist guide through treewidth. *Acta cybernetica*, 11(1-2):1, 1994. [144](#)
- [19] Hans L Bodlaender and Dimitrios M Thilikos. Graphs with branchwidth at most three. *Journal of Algorithms*, 32(2):167–194, 1999. [12](#)
- [20] Franz J Brandenburg and Mao-cheng Cai. Shortest path and maximum flow problems in networks with additive losses and gains. *Theoretical Computer Science*, 412(4):391–401, 2011. [15](#), [18](#), [65](#), [66](#), [70](#), [71](#), [79](#), [83](#)
- [21] Daniela Bronner and Bernard Ries. An Introduction to Treewidth. Technical Report ROSE-REPORT-2006-004, Ecole Polytechnique Fédérale de Lausanne, 2006. [107](#)
- [22] Bala G. Chandran and Dorit S. Hochbaum. A Computational Study of the Pseudoflow and Push-Relabel Algorithms for the Maximum Flow Problem. *Oper. Res.*, 57(2):358–376, March/April 2009. [15](#)
- [23] Moses Charikar, Mohammad Taghi Hajiaghayi, Howard Karloff, and Satish Rao. ℓ_2^2 Spreading Metrics for Vertex Ordering Problems. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1018–1027. Society for Industrial and Applied Mathematics, 2006. [145](#)
- [24] Moon-Jung Chung, Fillia Makedon, Ivan Hal Sudborough, and Jonathan Turner. Polynomial time algorithms for the min cut problem on degree restricted trees. *SIAM Journal on Computing*, 14(1):158–177, 1985. [11](#)

- [25] Johanne Cohen, Fedor Fomin, Pinar Heggernes, Dieter Kratsch, and Gregory Kucherov. Optimal linear arrangement of interval graphs. In *International Symposium on Mathematical Foundations of Computer Science*, pages 267–279. Springer, 2006. [12](#)
- [26] Samir Datta and Prakriya. Planarity Testing Revisited. *Electronic Colloquium on Computational Complexity*, 9, 2011. [41](#)
- [27] Reinhard Deistel. *Graph Theory*. Springer Verlag, 2012. [41](#), [42](#), [107](#)
- [28] Zhong Deng and JW-S Liu. Scheduling real-time applications in an open environment. In *Real-Time Systems Symposium, 1997. Proceedings., The 18th IEEE*, pages 308–319. IEEE, 1997. [7](#)
- [29] E.A. Dinic. Algorithm for Solution of a Problem of Maximum Flow in Networks with Power Estimation. *Soviet Mathematics Doklady*, 11:1277–1280, 1970. [14](#)
- [30] Zdeněk Dvořák, Archontia C Giannopoulou, and Dimitrios M Thilikos. Forbidden graphs for tree-depth. *European Journal of Combinatorics*, 33(5):969–979, 2012. [12](#)
- [31] T Easton, S Horton, and RG Parker. A solvable case of the optimal linear arrangement problem on halin graphs. *Congressus Numerantium*, pages 3–18, 1996. [12](#)
- [32] Shimon Even and Yossi Shiloach. Np-completeness of several arrangement problems. *Department of Computer Science, Israel Institute of Technology, Haifa, Isreal, Tech. Rep*, 1975. [12](#)
- [33] Miranca Fischermann, Arne Hoffmann, Dieter Rautenbach, László Székely, and Lutz Volkmann. Wiener index versus maximum degree in trees. *Discrete Applied Mathematics*, 122(1):127–137, 2002. [150](#)
- [34] L.R. Ford and D.R. Fulkerson. Maximal Flow Through a Network. *Canadian Journal of Mathematics*, 8:399–404, 1956. [14](#)
- [35] Greg N Frederickson and Susanne E Hambrusch. Planar linear arrangements of outerplanar graphs. *IEEE Transactions on Circuits and Systems*, 35(3):323–333, 1988. [12](#)
- [36] Michael R. Garey, David S. Johnson, and Larry Stockmeyer. Some Simplified NP-Complete Graph Problems. *Theoretical Computer Science*, 1:237–267, 1976. [121](#)
- [37] Michael R Garey, David S. Johnson, and Larry Stockmeyer. Some simplified np-complete graph problems. *Theoretical computer science*, 1(3):237–267, 1976. [12](#), [92](#)
- [38] Fanica Gavril. Some np-complete problems on graphs. In *Proc. 11th Conf. on Information Sciences and Systems*, pages 91–95, 1977. [11](#)
- [39] Andrew V. Goldberg. A New Max-Flow Algorithm. Technical Report MIT/LCS/TM-291, Laboratory for Computer Science, MIT, 1985. [14](#)
- [40] Andrew V. Goldberg. Recent Developments in Maximum Flow Algorithms (Invited Lecture). In *SWAT '98: Proc. of 6th Scandinavian Workshop on Algorithm Theory*, pages 1–10. Springer Verlag, 1998. [14](#)

- [41] Andrew V. Goldberg. Two-Level Push-Relabel Algorithm for the Maximum Flow Problem. In *Proc. of 5th International Conference on Algorithmic Aspects in Information and Management*, AAIM '09, pages 212–225, Berlin, Heidelberg, 2009. Springer-Verlag. [15](#)
- [42] Andrew V Goldberg, Serge A Plotkin, and Éva Tardos. Combinatorial algorithms for the generalized circulation problem. *Mathematics of Operations Research*, 16(2):351–381, 1991. [65](#)
- [43] Andrew V. Goldberg and Satish Rao. Beyond the Flow Decomposition Barrier. *J. ACM*, 45(5):783–797, September 1998. [15](#)
- [44] Andrew V. Goldberg and Robert E. Tarjan. A New Approach to the Maximum Flow Problem. In *STOC 1986, Proc. of 18th Annual ACM Symposium on the Theory of Computing*, pages 136–146, 1986. [14](#)
- [45] MK Goldberg and IA Klipker. An algorithm for minimal numeration of tree vertices. *Sakharth. SSR Mecn. Akad. Moambe*, 81(3):553–556, 1976. [12](#)
- [46] Donald Goldfarb, Zhiying Jin, and Yiqing Lin. A polynomial dual simplex algorithm for the generalized circulation problem. *Mathematical programming*, 91(2):271–288, 2002. [65](#)
- [47] Donald Goldfarb and Yiqing Lin. Combinatorial interior point methods for generalized network flow problems. *Mathematical programming*, 93(2):227–246, 2002. [65](#)
- [48] Qian-Ping Gu and Hisao Tamaki. Optimal branch-decomposition of planar graphs in $o(n^3)$ time. *ACM Transactions on Algorithms (TALG)*, 4(3):30, 2008. [11](#)
- [49] Pinar Heggernes, Pim van Hof, Daniel Lokshtanov, and Jesper Nederlof. Computing the cutwidth of bipartite permutation graphs in linear time. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 75–87. Springer, 2010. [11](#)
- [50] Illya V Hicks. Planar branch decompositions ii: The cycle method. *INFORMS Journal on Computing*, 17(4):413–421, 2005. [11](#)
- [51] Dorit S. Hochbaum. The Pseudoflow Algorithm: A New Algorithm for the Maximum-Flow Problem. *Oper. Res.*, 56(4):992–1009, July 2008. [15](#)
- [52] Jordi Petit i Silvestre. *Layout problems*. PhD thesis, Ph. D. thesis, Universitat Politècnica de Catalunya, Barcelona, 25 May, 2001. [148](#)
- [53] Yuan Jinjiang and Zhou Sanming. Optimal labelling of unit interval graphs. *Applied Mathematics*, 10(3):337–344, 1995. [11](#), [12](#)
- [54] Anil Kamath and Omri Palmon. Improved interior point algorithms for exact and approximate solution of multicommodity flow problems. In *SODA*, volume 95, pages 502–511. Citeseer, 1995. [65](#)
- [55] Frank Kammer. Determining the Smallest k Such That G Is k -Outerplanar. In Lars Arge, Michael Hoffmann, and Emo Welzl, editors, *Proc. of 15th Annual European Symposium on Algorithms, ESA 2007*, pages 359–370. LNCS 4698, Springer Verlag, September 2007. [41](#)

- [56] Mamadou Moustapha Kanté, Fatima Zahra Moataz, Benjamin Momege, and Nicolas Nisse. Finding paths in grids with forbidden transitions. In *WG 2015, 41st International Workshop on Graph-Theoretic Concepts in Computer Science*, 2015. [71](#), [72](#)
- [57] Sanjiv Kapoor and Pravin M Vaidya. Speeding up karmarkar’s algorithm for multicommodity flows. *Mathematical programming*, 73(1):111–127, 1996. [65](#)
- [58] Richard M Karp. *Reducibility among combinatorial problems*. Springer, 1972. [92](#)
- [59] Assaf Kfoury. A Domain-Specific Language for Incremental and Modular Design of Large-Scale Verifiably-Safe Flow Networks (Part 1). Technical Report BUCS-TR-2011-011, CS Dept, Boston Univ, May 2011. [2](#), [8](#), [9](#), [14](#), [167](#)
- [60] Assaf Kfoury. The Denotational, Operational, and Static Semantics of a Domain-Specific Language for the Design of Flow Networks. In *Proc. of SBLP 2011: Brazilian Symposium on Programming Languages*, Sept 2011. [2](#), [14](#), [167](#)
- [61] Assaf Kfoury. A Compositional Approach to Network Algorithms. Technical Report BUCS-TR-2013-015, CS Dept, Boston University, November 2013. [9](#)
- [62] Assaf Kfoury. A compositional approach to network algorithms. Technical report, Computer Science Department, Boston University, 2013. [2](#), [9](#), [14](#), [25](#), [167](#)
- [63] Assaf Kfoury and Saber Mirzaei. A Different Approach to the Design and Analysis of Network Algorithms. Technical Report BUCS-TR-2012-019, CS Dept, Boston Univ, 2013. [2](#), [9](#), [14](#), [18](#), [63](#), [167](#)
- [64] Assaf Kfoury and Saber Mirzaei. Efficient reassembling of graphs, part 1: the linear case. *Journal of Combinatorial Optimization*, pages 1–33, 2016. [25](#), [89](#)
- [65] Samir Khuller, Balaji Raghavachari, and Neal Young. Designing multi-commodity flow trees. *Information Processing Letters*, 50(1):49–55, 1994. [11](#), [145](#)
- [66] Tom Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM (JACM)*, 46(6):787–832, 1999. [11](#), [145](#)
- [67] Sung-Soo Lim, Young Hyun Bae, Gyu Tae Jang, Byung-Do Rhee, Sang Lyul Min, Chang Yun Park, Heonshik Shin, Kunsoo Park, Soo-Mook Moon, and Chong Sang Kim. An accurate worst case timing analysis for risc processors. *IEEE Transactions on Software Engineering*, 21(7):593–604, 1995. [7](#)
- [68] Fillia Makedon and Ivan Hal Sudborough. On minimizing width in linear layouts. *Discrete Applied Mathematics*, 23(3):243–265, 1989. [10](#)
- [69] Fillia S. Makedon, Christos H. Papadimitriou, and Ivan Hal Sudborough. Topological Bandwidth. *SIAM Journal on Algebraic and Discrete Methods*, 6(3):418–444, 1985. [111](#)
- [70] G. Mazzoni, S. Pallottino, and M.G. Scutella. The Maximum Flow Problem: A Max-Preflow Approach. *European Journal of Operational Research*, 53:257–278, 1991. [15](#)

- [71] Saber Mirzaei. Minimum average delay of routing trees. *arXiv preprint arXiv:1601.02697*, 2016. [12](#)
- [72] Saber Mirzaei and Assaf Kfoury. Linear arrangement of halin graphs. *arXiv preprint arXiv:1509.08145*, 2015. [12](#)
- [73] Saber Mirzaei and Assaf Kfoury. Shortest Path and Maximum Flow Problems in Planar Flow Networks with Additive Gains and Losses. *CoRR*, abs/1603.08997, April 2016. Available at <http://arxiv.org/abs/1603.08997>. [18](#)
- [74] Burkhard Monien. The complexity of embedding graphs into binary trees. In *Fundamentals of computation theory*, pages 300–309. Springer, 1985. [148](#)
- [75] Burkhard Monien and Ivan Hal Sudborough. Min cut is np-complete for edge weighted trees. *Theoretical Computer Science*, 58(1):209–229, 1988. [11](#)
- [76] Burkhard Monien and Ivan Hal Sudborough. Min Cut is NP-Complete for Edge Weighted Trees. *Theoretical Computer Science*, 58(1–3):209–229, 1988. [111](#)
- [77] Steven M Murray. *An interior point approach to the generalized flow problem with costs and related problems*. 1992. [65](#)
- [78] Kenji Onaga. Dynamic programming of optimum flows in lossy communication nets. *Circuit Theory, IEEE Transactions on Circuits and Systems*, 13(3):282–287, 1966. [65](#)
- [79] Jordi Petit. Addenda to the Survey of Layout Problems. *Bulletin of the EATCS*, (105):177–201, October 2011. [91](#), [97](#), [145](#)
- [80] Johan Pouwelse, Koen Langendoen, and Henk Sips. Dynamic voltage scaling on a low-power microprocessor. In *Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 251–259. ACM, 2001. [7](#)
- [81] Satish Rao and Andréa W Richa. New Approximation Techniques for Some Ordering Problems. In *SODA*, volume 98, pages 211–219, 1998. [145](#)
- [82] John Regehr. Inferring scheduling behavior with hourglass. In *USENIX Annual Technical Conference, FREENIX Track*, pages 143–156, 2002. [7](#)
- [83] Neil Robertson and Paul Seymour. Graph minors xxiii. nash-williams’ immersion conjecture. *Journal of Combinatorial Theory, Series B*, 100(2):181–205, 2010. [11](#)
- [84] Habib Rostami and Jafar Habibi. Minimum linear arrangement of chord graphs. *Applied Mathematics and Computation*, 203(1):358–367, 2008. [12](#)
- [85] Douglas C Schmidt, David L Levine, and Sumedh Mungee. The design of the tao real-time object request broker. *Computer Communications*, 21(4):294–324, 1998. [7](#)
- [86] Paul D. Seymour and Robin Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994. [11](#), [148](#)

- [87] Yossi Shiloach. A minimum linear arrangement algorithm for undirected trees. *SIAM Journal on Computing*, 8(1):15–32, 1979. [12](#), [155](#)
- [88] Nate Soule, Azer Bestavros, Assaf Kfoury, and Andrei Lapets. Safe Compositional Equation-based Modeling of Constrained Flow Networks. In *Proc. of 4th Int'l Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, Zürich, September 2011. [2](#), [8](#), [14](#), [167](#)
- [89] John A Stankovic. Vesta toolset for constructing and analyzing component based embedded systems. In *International Workshop on Embedded Software*, pages 390–402. Springer, 2001. [7](#)
- [90] Atsushi Takahashi, Shuichi Ueno, and Yoji Kajitani. Minimal acyclic forbidden minors for the family of graphs with bounded path-width. *Discrete Mathematics*, 127(1):293–304, 1994. [12](#)
- [91] Eva Tardos and Kevin D Wayne. Simple generalized maximum flow algorithms. In *Integer Programming and Combinatorial Optimization*, pages 310–324. Springer, 1998. [65](#)
- [92] Dimitrios M Thilikos. Algorithms and obstructions for linear-width and related search parameters. *Discrete Applied Mathematics*, 105(1):239–271, 2000. [12](#)
- [93] Dimitrios M. Thilikos, Maria J. Serna, and Hans L. Bodlaender. A Polynomial Time Algorithm for the Cutwidth of Bounded Degree Graphs with Small Treewidth. In F. Meyer auf der Heide, editor, *Algorithms, ESA 2001, LNCS 2161*, pages 380–390. Springer Verlag, 2001. [108](#)
- [94] K Truemper. On max flows with gains and pure min-cost flows. *SIAM Journal on Applied Mathematics*, 32(2):450–456, 1977. [65](#)
- [95] László A. Végh. A strongly polynomial algorithm for generalized flow maximization. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, STOC '14, pages 644–653, New York, NY, USA, 2014. ACM. [65](#)
- [96] Daniel Villeneuve and Guy Desaulniers. The shortest path problem with forbidden paths. *European Journal of Operational Research*, 165(1):97–107, 2005. [71](#)
- [97] Harry Wiener. Structural determination of paraffin boiling points. *Journal of the American Chemical Society*, 69(1):17–20, 1947. [149](#)
- [98] Lidong Wu. On strongly planar 3sat. *Journal of Combinatorial Optimization*, pages 1–6, 2015. [80](#)
- [99] Mihalis Yannakakis. A polynomial algorithm for the min-cut linear arrangement of trees. *Journal of the ACM (JACM)*, 32(4):950–988, 1985. [11](#)

Curriculum Vitae

- Contact** Saber Mirzaei
Department of Computer Science, Boston University, 111 Cummington Street,
Boston, MA 02215, USA
- Education** **Sharif University of Technology**, B.Sc., Computer Engineering, September
2008.
Iran University of Science and Technology, M.Sc, Software Engineering, May
2011.
Boston University PhD candidate, September 2016. Thesis advisor: Assaf
Kfoury.
- Publications** 1. Mirzaei, Saber and Kfoury, Assaf. *Shortest Path and Maximum Flow Problems
in Planar Flow Networks with Additive Gains and Losses*. Submitted to Net-
works journal, March 2016.
2. Mirzaei, Saber and Kfoury, Assaf. *Efficient Reassembling of Graphs, Part 2: The
Balanced Case*. Submitted to the journal of Fundamenta Informaticae, March
2016.
3. Kfoury, Assaf and Mirzaei, Saber. *Efficient Reassembling of Graphs, Part 1: The
Linear Case*. Journal of Combinatorial Optimization, Publish Date: May 2016.
4. Mirzaei, Saber. *Minimum Average Delay of Routing Trees*. Preprint BUCS-TR-
2016-001, CS Dept., Boston University, January 2016.
5. Mirzaei, Saber and Kfoury, Assaf. *Linear Arrangement of Halin Graphs*. Preprint
BUCS-TR-2015-012, CS Dept., Boston University, September 2015.
6. Mirzaei, Saber and Esposito, Flavio. *An Alloy Verification Model for Consensus-
Based Auction Protocols*. IEEE 35th ICDCSW (ADSN 2015) June 2015.
7. Motallebi, Hassan; Azgomi, Mohammad ; Mirzaei, Saber and Movaghar, Ali.
*A New Extension of Activity Networks for Modelling and Verification of Timed
Systems*. Turk J of EE & CS, Vol. 21, No. 6, Oct. 2013, pp. 1751-1779.
8. Mirzaei, Saber; Bahargam, Sanaz; Skowya, Richard; Kfoury, Assaf and Bestavros,
Azer. *Using Alloy to Formally Model and Reason About an OpenFlow Network
Switch*. Preprint BUCS-TR-2013-007, CS Dept., Boston University September,
2013.
9. Kfoury, Assaf and Mirzaei, Saber. *A Different Approach to the Design and Anal-
ysis of Network Algorithms*. Preprint BUCS-TR-2012-019, CS Dept., Boston
University December, 2012.

10. Lapets, Andrei and Mirzaei, Saber. *Towards Lightweight Integration of SMT Solvers*. Preprint BUCS-TR-2012-017, CS Dept., Boston University December, 2012.