Graphs with Distance Guarantees

Dagmar Handke

Dissertation der Universität Konstanz
Fakultät für Mathematik und Informatik

Dezember 1999

Teile dieser Arbeit wurden bereits in Brandes und Handke (1997, 1998), Handke (1998, 1999), Cicerone et. al. (1999) bzw. in den *Konstanzer Schriften in Mathematik und Informatik* Nr. 16, 53 und 81 veröffentlicht.

Graphs with Distance Guarantees

Dissertation zur Erlangung des akademischen Grades des
Doktors der Naturwissenschaften an der Universität Konstanz
Fakultät für Mathematik und Informatik

vorgelegt von Dagmar Handke

1. Referent: Prof. Dr. Dorothea Wagner, Universität Konstanz
2. Referent: Prof. Dr. Peter Widmayer, ETH Zürich

Tag der mündlichen Prüfung: 10. Dezember 1999

Abstract

One goal in network design is the construction of sparse networks that guarantee short distances with respect to some given distance requirements. By this, it can be guaranteed, for example, that delays that are incurred by link faults are bounded. An appropriate graph-theoretic model for this is the concept of k–spanners: Given a graph G, a *k–spanner* of G is a spanning subgraph S, such that the distance between any two vertices in S is at most k times longer than the distance in G. Research in this area has mainly concentrated on two aspects: *minimum k–spanner*s, i.e., k–spanners that contain the fewest edges among all k–spanners, and *tree k–spanner*s, i.e., k–spanners that are trees.

In this thesis, we use k–spanners to model further desirable properties from network design (such as reliability) within a graph-theoretic framework. Our main emphasis is on *sparse* graphs that guarantee *short distances*, and we are interested in *simple structures* and *fault-tolerance*.

Basically, our research comprises two major parts: In the first part, we use k–spanners as a means of analyzing a given graph: We are given a graph and the problem is to decide whether it contains some particular form of k–spanner. Often, k–spanners are difficult to find, and most problems in this area are $\mathcal{NP}$-hard. Moreover, both the concepts of minimum or tree k–spanners exhibit serious drawbacks with respect to typical applications in network design. To overcome these difficulties, we propose three approaches stemming from different thematic contexts:

- k–spanners within the context of planarity;
- k–spanners that are sparse, simply structured, and fault-tolerant;
- generalized k–spanners using auxiliary vertices.

To summarize, our results in this first part of the thesis indicate even more that the problem of finding k–spanners in their different shaping is difficult, and only some special cases can be solved efficiently.

In contrast, in the second part of this thesis, we use k–spanners to construct graphs from scratch, subject to some given requirements. We introduce graph-theoretic models for graphs that guarantee constant delays even if a multiple number of edges fail. In particular, we consider two cases: an unlimited and a limited number of edge faults. Though we cannot hope for finding efficient characterizations for both graph classes, we give characterizations and examine some popular graph classes, graph operations and network topologies with respect to the given requirements.

Deutsche Zusammenfassung

Ein wichtiges Teilproblem beim Entwurf von Netzwerken ist das Finden von dünnen Netzwerken, in denen die Abstände zwischen je zwei Knoten nicht zu groß bzgl. vorgegebener Entfernungsbedingungen werden. Auf diese Weise kann beispielsweise garantiert werden, dass Verzögerungen durch den Wegfall von Verbindungen unter Kontrolle gehalten werden. Ein geeignetes graphentheoretisches Modell dafür sind *k-Spanner*: Ein aufspannender Teilgraph S heißt *k–Spanner* eines Graphen G für ein $k \geq 1$, falls die Distanz in S für jedes Knotenpaar höchstens das k–fache der Distanz in G ist. Die Forschung im Bereich der k–Spanner hat sich meist auf das Studium von *minimalen k–Spannern* (also k–Spanner mit der kleinstmöglichen Kantenzahl) oder *k–Baumspannern* (also k–Spanner, die Baumstruktur haben) konzentriert.

In dieser Dissertation verwenden wir das Konzept der k–Spanner, um zusätzliche wünschenswerte Eigenschaften (wie zum Beispiel Zuverlässigkeit) graphentheoretisch zu fassen. Wir beschäftigen uns dabei hauptsächlich mit *dünnen* Graphen, die *kurze Distanzen* garantieren und dabei eine möglichst *einfache Struktur* besitzen beziehungsweise *fehlertolerant* sind.

Die Ergebnisse lassen sich im wesentlichen in zwei Hauptteile untergliedern. Im ersten Teil verwenden wir k–Spanner, um einen vorgegebenen Graphen zu *analysieren*, indem wir untersuchen, ob er eine bestimmte Form von k–Spanner enthält. Für die bislang untersuchten, oben erwähnten Varianten hat sich dieses Problem jedoch meist als schwierig beziehungsweise $\mathcal{NP}$–schwer herausgestellt. Außerdem hat sowohl das Konzept der minimalen k–Spanner als auch das der k–Baumspanner Nachteile in Bezug auf netzwerktypische Anforderungen. Um dies zu überwinden, betrachten wir drei Modelle aus verschiedenen thematischen Kontexten:

- k–Spanner und Planarität;
- dünne, einfach strukturierte k–Spanner, die fehlertolerant sind;
- verallgemeinerte k–Spanner, die Hilfsknoten berücksichtigen.

Die Ergebnisse belegen, dass Probleme im Bereich der k–Spanner schwierig sind. Oft können nur Teilprobleme effizient gelöst werden.

Der zweite Teil dieser Dissertation verfolgt ein artverwandtes Problem, jedoch diesmal aus der Perspektive der Graph-*Konstruktion*: Unser Ziel ist es, Graphen zu konstruieren, die höchstens konstante Verzögerungen garantieren, selbst wenn beliebige Kanten des Graphen ausfallen. Es zeigt sich, dass das allgemeine Problem der Erkennung der jeweiligen Graphklassen $\mathcal{NP}$–schwer ist. Wir geben jedoch Charakterisierungen an und beschreiben, wie sich einige populäre Graphklassen, Graphoperationen und Netzwerk-Topologien bezüglich der gegebenen Eigenschaften verhalten.

Acknowledgments

First of all, my sincere thanks go to my adviser Professor Dorothea Wagner for giving me the opportunity to participate in her stimulating work group in Konstanz and for all her support. I am also very grateful to Professor Peter Widmayer for his encouragement and interest in my work, and for taking over the role of a second referee.

It was always a pleasure to work in the computer science group at the University of Konstanz, thanks to all colleagues, collectively. In particular, I thank Annegret Liebers and Sabine Cornelsen for careful reading and commenting on drafts of this thesis, and Torsten TEXy Grust for LATEXnical support.

Last but not least, I would like to express my gratitude to my family and my friends who always kept me dancing on.

While working on this thesis, I was supported financially by the Deutsche Forschungsgemeinschaft by grants Wa 654/10-1 and Wa 654/10-2.

Credits and Pre-Publications

Some of the results of this thesis have already been published and/or have been obtained jointly:

- The results on k–spanners in the context of planarity as reported in Sections 3.2 and 3.4 were obtained jointly with Ulrik Brandes from University of Konstanz, and have appeared in Brandes and Handke (1998, 1997).
- The work on fault-tolerant tree k–spanners as presented in Chapter 4 has been published in Handke (1999, 1998).
- Chapter 5 has benefited much from joint work with Guy Kortsarz from The Open University, Israel.
- The results on the survivable networks as outlined in Chapter 6 are joint work with Serafino Cicerone and Gabriele Di Stefano from University of L'Aquila, Italy. They will be published in Cicerone et al. (1999).

I am particularly grateful to all co-authors for the stimulating joint work including all the lively discussions, and for allowing me to include the results in this thesis. Moreover, I thank Dorothea Wagner for lots of valuable comments and ideas on different parts of this thesis.

Contents

Chapter 1

Introduction

A communication network is a means to provide the exchange of information between a set of sites. In order to achieve this, a direct communication between a pair of sites is carried out by a link. Accordingly, a collection of such links enables the transmission of messages between the sites. The static aspects of a network, such as the number and the location of the sites, as well as the links between the sites, are defined by the topology of the network.

Usually, networks are modeled as graphs. In this thesis, we discuss different topological aspects of networks. Our main focus, however, is of graph-theoretic nature.

1.1 Motivation: Spanners in Network Design

Network Design

One major concern in the design of a network for a given set of sites is to find a set of links, such that the communication between the sites is easy and quick, i.e., such that the distance between any pair of sites is short. As an example, this may mean that the communication between each pair of sites does not involve too many links.

Consider for example the following scenario: We are given a set of sites together with a list of communication requirements between pairs of sites, and the goal is to specify links such that these requirements are fulfilled in a suitable manner. The easiest way to do that would be to connect each pair of sites of the list directly by a link. On the other hand, the construction and maintenance of links usually causes significant costs, and it is hence desirable to minimize the number of links that are constructed eventually. However, if a link between two particular sites is omitted, the communication between

these sites has to be carried out indirectly via other links, and the length of the connection increases.

This leads to the following goal: Given a set of sites and a list of communication requirements, find a *sparse* network (i.e., one with few links) that guarantees *short connections* with respect to the requirements given.

Apart from a small number of links, other properties of a network may also be desirable, depending on the concrete application:

- Often it is important that the topology satisfies some *structural properties*. Consider the case where we are given a network and the goal is to route a message, i.e., to find an appropriate path within this network from a specified site to another one. If the structure of the network guarantees that there exist only few such paths, the routing problem is easy. In order to minimize computational overhead, it can thus be helpful to impose a certain network structure.

- As mentioned above, the main function of a network is to provide connectivity between the sites. Especially in the light of communication networks, it is crucial, that the connectivity (and maybe other properties) are preserved to a certain degree, even in the case of faults in either sites or links. The corresponding aspect in network design is called *fault-tolerance* or *survivability*.

There are many other additional properties that may be important in network design according to a concrete scenario. In the design of parallel computers, for example, it is inevitable to bound the number of links that are incident to one particular site, in order to avoid the congestion of this site. Throughout this thesis, we concentrate only on the two aspects mentioned in the list above: Our main emphasis is on *sparse* networks that guarantee *short distances*, and we are interested in *simple structures* and *fault-tolerance*.

Spanners: A Graph Theoretic Model

We study networks within the following graph-theoretic model: We use graphs in which vertices represent the sites of the network, and edges represent links between pairs of sites. In most parts, we assume that the transmission of a message incurs a constant unit cost for every link within the network, and that links can be used bidirectionally. Accordingly, in our model we are dealing with unweighted, and undirected graphs. The distance between two sites is measured by the length of a shortest path in the underlying graph, i.e., by the minimum number of edges that have to be traversed in order to reach one vertex from the other.

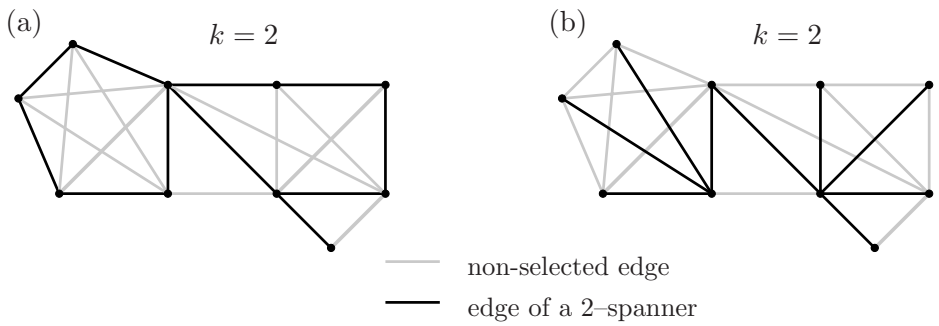

Figure 1.1: A non-planar graph G together with (a) a 2–spanner that is planar, and (b) that is even a tree.

Spanners. As mentioned above, our goal is to bound distances within the network. To be more precise, we want to achieve constant distance guarantees. For this, we use the concept of k–spanners, as it has been introduced by Peleg and Ullman (1987): Given a graph G, a *k–spanner* of G is a spanning subgraph S (i.e., a graph that consists of the same set of vertices and uses a subset of the edges), such that the distance between any two vertices in S is at most k times longer than the distance in G. The parameter k is called *stretch factor*. See Figure 1.1 for examples of 2–spanners.

It is easy to see that every graph G contains a k–spanner for any fixed k, because G itself is a k–spanner for all k. Observe that we consider k to be constant, i.e., k is independent of the number of vertices or edges of G.

Research in the context of k–spanners has been very active (see, e.g., Peleg and Schaeffer (1989); Soares (1992); Cai and Corneil (1995); Venkatesan et al. (1997); Kortsarz (1998); Brandstädt et al. (1999a)). Furthermore, k–spanners have applications in many different areas (see, e.g., Chapter 2).

Sparsest spanners. As our second objective apart from the distance constraint is the design of sparse networks, it is a natural approach to consider *minimum k–spanners*, i.e., k–spanners that have a minimum number of edges among all k–spanners: A minimum k–spanner is the least expensive approximation of a network, in which the delay can be guaranteed not to exceed a factor of k. For an example, reconsider Figure 1.1: Both subgraphs given there form a 2–spanner, but the subgraph in Part (b) contains fewer edges. In fact, the subgraph in (b) is a minimum 2–spanner because it is a tree and is thus minimally connected.

As every graph contains a k–spanner, in particular, it also contains a minimum k–spanner. Accordingly, the problem of interest is an optimization problem (see, e.g., Cai (1994)).

Sparse spanners. Another idea towards finding sparse k–spanners is as follows: Instead of searching for a k–spanner with the fewest edges, we select a k–spanner that has a specified structure which guarantees sparseness. Now, a new question arises: If we restrict the structure of the subgraph, it cannot be guaranteed anymore that a given graph contains a subgraph such that both the distance constraint and the structure constraint can be fulfilled. Thus, we also have to consider the decision problem for the existence of such a particular k–spanner.

Trees are the sparsest connected graphs with respect to the number of edges. Apart from their sparseness, these graphs also have some nice properties, which are interesting from the point of view of routing: paths between two vertices are unique. It is therefore interesting to consider tree networks in the context of k–spanners. But observe that an arbitrary spanning subtree of a graph may cause a stretch factor that is not constant, but in the order of the number of vertices, see Figure 1.2. To overcome this, *tree k–spanner*s, i.e., k–spanners that are trees have been studied (see, e.g., Cai and Corneil (1995)).

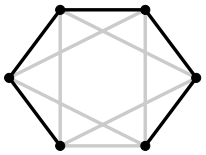

Figure 1.2: A spanning tree with stretch factor 5.

1.2 New Topics on Spanners

Research in the area of k–spanners has concentrated on minimum and tree k–spanners, by this modeling the aspects of optimal sparseness and simple structure. In this thesis, we make extensive use of the concept of k–spanners in order to model other related properties that are desirable within the framework of network design.

Basically, our research can be divided into two major parts: In the first part, which consists of Chapters 3, 4, and 5, we use k–spanners more or less within the context in which they have been defined, as a means of *analyzing* a given network. We are given a graph and the problem is to decide whether some particular form of k–spanner (or a variant thereof) exists within this graph. In contrast to this, in the second part of this thesis in Chapter 6, we use k–spanners to *construct* networks from scratch, subject to some given requirements.

1.2.1 Analysis of Graphs via Variants of k–Spanners

Let us first turn to the goal of analyzing a given graph with respect to the existence of k–spanners within that graph. As it has been shown, k–spanners are often difficult to find, and most problems in this area are $\mathcal{NP}$-hard (see, e.g., Cai (1994); Cai and Corneil (1995)). In particular, the problems of finding minimum k–spanners or tree k–spanners are $\mathcal{NP}$-hard in general graphs for almost all values of k. A second problem with minimum and, in particular, tree k–spanner is as follows: Both concepts exhibit drawbacks with respect to network design. In order to overcome these difficulties, we propose three models for different objectives. In the following, we give a short overview on the respective topics and results. For a detailed motivation and discussion we refer to the individual chapters.

Minimum Spanners and Planarity

Since the problem of finding minimum k–spanners is $\mathcal{NP}$-hard for general graphs, it is interesting to study the problem within the context of *planar graphs*, i.e., graphs that can be drawn in the plane without edge crossings. We do this in Chapter 3, where we first restrict the set of *input graphs* to planar graphs, and reconsider the problem of finding minimum k–spanners within these. Similar to many other $\mathcal{NP}$-complete graph theoretic problems, it turns out that the complexity status remains hard for planar instances for most values of k, be it on unweighted, weighted or directed graphs.

Motivated by this, we further restrict the input instances, and consider *(ℓ–)outerplanar* graphs, a proper subclass of planar graphs. Here, the border to intractability is crossed, and we can give an efficient algorithm for finding minimum k–spanners in these graphs.

Following the line of tree k–spanners, we also consider the case where we restrict the *output* to be planar. This leads to the definition of *planar k–spanner*s, i.e., k–spanners that are planar. Such graphs are also guaranteed to be sparse and may serve as planar representatives of non-planar graphs, such that distances do not increase too much. Unfortunately, the problem of finding planar k–spanners of minimum size is $\mathcal{NP}$-hard again for most k.

Fault-Tolerant Tree Spanners

In Chapter 4, we turn back to the concept of tree k–spanners. Apart from their nice properties of optimal sparseness and simple structure, tree networks have one important drawback: in the case of a fault of just one site or link, the network gets disconnected. As mentioned in the motivation, one of the objectives in our work is *fault-tolerance*. In order to incorporate this into the

concept of k–spanners, we introduce *independent tree k–spanners*, which may serve as a concept for more reliable, but still simply structured networks that guarantee short delays even in the case of edge or vertex faults. In particular, we show how to decide the existence of independent tree 2–spanners in linear time. For $k \geq 4$, however, the corresponding problem is proved to be $\mathcal{NP}$-complete, be it for the case of edge or vertex faults.

As it turns out, the concept of independent tree k–spanners is very restrictive in the sense that the class of graphs that have a pair of such independent tree k–spanners for a fixed stretch factor k is quite small. In many settings in the design of subnetworks, however, it is not strictly necessary that the distance guarantee holds for all pairs of vertices, but only for the distances of the vertices from one distinguished root vertex. This leads to the definition of a relaxed version of k–spanners called *k–root-spanners*, and in particular to *independent tree k–root-spanners*.

Since the problem of finding independent tree k–root-spanners emerges to be $\mathcal{NP}$-hard in general for both edge and vertex faults, we also consider a natural restriction of the problem. Here, the situation for edge or vertex faults differs significantly: In the case of an edge fault, the corresponding problem can be solved in linear time for all k, whereas the corresponding problem for a vertex fault remains $\mathcal{NP}$-complete.

Steiner Tree Spanners

Following the line of the previous paragraph where we have considered a relaxed version of tree k–spanners, in Chapter 5 we discuss a further variant: We introduce auxiliary vertices and edges, which may help in finding a tree k–spanner without imposing new distance constraints themselves. The corresponding concept is called *Steiner tree k–spanner*. As a starting point, we first study the problem of deciding whether such a Steiner tree k–spanner exists in a given graph. Due to our general interest in sparseness, our second objective is to find a smallest Steiner tree k–spanner (if one exists at all).

In order to answer these questions for the case $k = 2$, we develop a model in terms of an equivalent tree covering problem. Using this, we show that the problem of finding a Steiner tree 2–spanner is $\mathcal{NP}$-hard in general. Moreover, even if we know in advance that a given graph contains a Steiner tree k–spanner for any arbitrary $k \geq 2$, we prove that we cannot even hope to find efficiently a Steiner tree k–spanner that is closer to the optimal one than within a logarithmic factor. We conclude the reflections on this topic by discussing some problems related to the model for the case $k = 2$.

1.2.2 Spanners for Network Construction

Until now, we have considered k–spanners within the context of the analysis of a given graph: a k–spanner is a *subgraph* of a given graph G and thus can inherit edges only from G. We now follow a different course: We are looking at the problem of constructing and characterizing graphs that exhibit certain specified properties.

Survivable Networks with Bounded Delay

Following the arguments of the motivation, we turn our attention again towards fault-tolerance and the maintenance of short distances. Our goal in Chapter 6 is to describe networks that are reliable with respect to these: We introduce and characterize new classes of graphs that guarantee constant delay factors even if a multiple number of edges fail. In particular, we consider two cases: Graphs that guarantee constant delays

- even if an *unlimited* number of edges fails, or
- if the number of edge faults is *limited* to a fixed number ℓ.

The graph classes that model the respective properties are called *k–self-spanners* and *(k, ℓ)–self-spanners*, respectively. In both cases, the name is motivated by strong relationships with the concept of *k–spanners*.

For both graph classes, we show that the problem of deciding whether a graph belongs to the respective graph class is $\mathcal{NP}$-hard for general k (and ℓ). On the other hand, we give strict (but non-constructive) characterizations. For the more realistic approach of limited edge faults, we furthermore study how several popular graph classes, graph operations and network topologies fit into the context of self-spanners.

1.3 Prerequisites

In general, we assume that the reader is familiar with the usual notations and definitions from graph theory and complexity theory. To avoid ambiguities, we have compiled short overviews.

1.3.1 Graph Theory

Apart from some parts of Chapter 3, we deal with *unweighted* and *undirected* graphs. The following contains just a short introduction to the terms of graph

theory as they are used throughout this thesis. All non-standard terminology is explained at the places where it appears. More details are provided by a comprehensive glossary of graph-theoretic terms in Appendix B, and Appendix A comprises a list of notations. For further details on graph theory see for example Harary (1969); Bondy and Murty (1976); Even (1979); Golumbic (1980). A survey on planar graphs can be found in Nishizeki and Chiba (1988).

In what follows, $G = (V, E)$ denotes an *unweighted* and *undirected* graph (unless stated otherwise) with finite vertex set V and finite edge set E. $V(G)$ (and $E(G)$, respectively), denotes the vertex set (and edge set, respectively) of G. We do not allow loops or multiple edges. In an *edge weighted graph* (or simply *weighted graph*), we associate a non-negative weight $w : E \to \mathbb{R}^+$ with every edge e.

A *subgraph* $H = (V', E')$ of $G = (V, E)$ (with $V' \subseteq V$ and $E' \subseteq E$) is called *spanning* if $V = V'$. If R is a subset of V, then $G[R]$ represents the subgraph of G that is *induced* by R; i.e., $G[R]$ consists of the vertices of R and contains all edges of G that are incident only to vertices of R. $G - E'$, where $E' \subseteq E$, is the graph obtained from G by deleting all edges in E'. In the case of the deletion of a single edge we also write $G - e$.

When considering paths in a graph, we always assume that they are simple, i.e., neither edges nor vertices are repeated within the path. The *length* of a path P is the number of its edges. In the case of weighted graphs, the length is the total sum of the edge weights. An ℓ–path is a path of length ℓ. The *distance* between two vertices u and v in G, i.e., the length of a shortest path, is denoted by $d_G(u, v)$. The *diameter* of G is the length of a longest shortest path in G, i.e., $\max_{u,v \in V} d_G(u, v)$.

A closed path is called a *cycle*. A *chord* in a path (or cycle) P is an edge $e \notin P$ that is incident to two vertices that occur in P but that are not adjacent in P. P is called *induced* in G if $G[V(P)] = P$, i.e., if P has no chord.

For a vertex v, denote by $N(v)$ the set of all *neighbors* of v, i.e., vertices that are adjacent to v. The number of neighbors of a vertex is called its *degree*. A vertex v of a graph G is called *universal w.r.t.* $V(G)$ if $N(v) \cup \{v\} = V(G)$. Let v be universal w.r.t. $V(G)$, then the *star centered at* v is the graph consisting of all vertices of G and all edges incident to v.

A *tree* is a connected graph that does not contain a cycle. A tree with n vertices contains $|V| - 1$ edges, and is a minimal connected graph. Paths between any pair of vertices within a tree are unique.

We denote by P_n the graph consisting of a simple induced path on n vertices and $n - 1$ edges, and by C_n the graph consisting of a simple induced cycle on n vertices. $\mathscr{C}_n$ denotes a simple cycle on n vertices that is not nec-

essarily induced. K_n denotes the complete graph on n vertices, $K_{n,m}$ is the complete bipartite graph on n and m vertices.

For a connected graph, an *ℓ–separator* is a set of ℓ vertices whose deletion disconnects the graph. A graph is *biconnected* (or *2–vertex-connected*) if it has no 1–separator, and *ℓ–vertex-connected* if it has no $(\ell-1)$–separator. A 1–separator is also called *articulation vertex*. A *block* of a graph is a maximal biconnected subgraph. A graph is *ℓ–edge-connected* if no deletion of $\ell-1$ edges disconnects it.

1.3.2 Complexity Theory

For a comprehensive discussion and a formal treatment of complexity theory we refer the reader for example to Aho et al. (1974); Garey and Johnson (1979); Papadimitriou (1994). Here, we only survey the notation used within this thesis. More details are provided by a glossary on complexity-theoretic terms in Appendix C.

We distinguish between two types of problems: *decision problems* and *optimization problems*. In the first case, an answer to such a problem is either 'yes' or 'no'. In the second case, there (usually) exist many feasible solutions which all have an assigned value, and we are looking for a solution of optimal value with respect to the given requirements (i.e., a solution with either maximal or minimal value).

Within classical complexity theory, the running time of an algorithm is measured by its worst-case complexity in terms of the size of the input (also called instance). In the following, we always assume that the input is encoded in a suitable way to fully describe the data that is necessary to solve the problem. Let $|I|$ describe the size of (the encoding of) an instance I. We consider an algorithm *efficient* if its running time is bounded by a polynomial in $|I|$.

Throughout this thesis, we use the $\mathcal{O}$ notation to bound the running time of an algorithm from above: A function $f : \mathbb{N} \to \mathbb{R}^+$ belongs to the class $\mathcal{O}(g(n))$ for a function $g : \mathbb{N} \to \mathbb{R}^+$ if there are constants $c \in \mathbb{R}$ and $n_0 \in \mathbb{N}$ such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$. Informally, $f \in \mathcal{O}(g(n))$, if f grows at most as fast as g. Furthermore, we write $f \in o(g(n))$, if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, i.e., f grows more slowly than g.

Complexity classes. The most important complexity classes that are treated within this thesis are the classes $\mathcal{P}$ and $\mathcal{NP}$. These are the classes of decision problems that can be solved in polynomial time by a *deterministic* (and *non-deterministic*, respectively) Turing machine. The class *co-$\mathcal{NP}$*

consists of those decision problems for which its complement is in $\mathcal{NP}$. It is still an open problem if $\mathcal{P} = \mathcal{NP}$ or $\mathcal{NP} = co-\mathcal{NP}$, but it is widely believed that both is not the case.

Given two decision problems Π and Π', denote by I (and I', respectively) an encoded instance of Π (and Π', respectively). A *polynomial transformation* (also called *reduction*) from Π' to Π is a function f that transforms an instance of Π' into an instance of Π such that f can be computed in polynomial time, and the answer for I' is 'yes' in Π' if and only if the answer for $I = f(I')$ is 'yes' in Π. A decision problem Π is called *$\mathcal{NP}$-complete* if $\Pi \in \mathcal{NP}$ and if, for every decision problem $\Pi' \in \mathcal{NP}$, there is a polynomial transformation from Π' to Π. The term *co-$\mathcal{NP}$-complete* is defined analogously. Observe that $\mathcal{NP} = co-\mathcal{NP}$ if there is a co-$\mathcal{NP}$-complete problem that is also in $\mathcal{NP}$. Therefore, co-$\mathcal{NP}$-complete problems can be considered even harder than problems in $\mathcal{NP}$. We say that a problem is $\mathcal{NP}$-hard if it is at least as difficult as any $\mathcal{NP}$-complete problem.

The notion of polynomial transformation can be extended to optimization problems by reformulating the optimization problem in terms of a decision problem by introducing a new parameter to bound the quality of a feasible solution. See Garey and Johnson (1979) for details. We can hence use the concept of $\mathcal{NP}$-completeness for decision problems for these modified optimization problems.

It is well-known that an $\mathcal{NP}$-hard problem cannot be solved by a polynomial algorithm unless $\mathcal{P} = \mathcal{NP}$. Accordingly, if we have shown that a problem is $\mathcal{NP}$-complete or $\mathcal{NP}$-hard we cannot hope for finding an efficient algorithm to solve it.

Approximation. One approach to deal with $\mathcal{NP}$-complete or $\mathcal{NP}$-hard problems is searching for algorithms that yield a solution which is maybe not optimal, but for which at least some guarantee on its quality can be given.

Let $\mathcal{A}$ be a polynomial-time algorithm that produces a feasible (though not necessarily optimal) solution for an optimization problem Π, and denote by $\mathcal{A}(I)$ the value of the solution that is achieved by $\mathcal{A}$ for the instance I. $\mathcal{A}$ is called a *δ–approximation algorithm* (where $\delta > 1$) for Π if for every instance I of Π,

$$\max\left\{\frac{\mathcal{A}(I)}{\mathrm{OPT}(I)}, \frac{\mathrm{OPT}(I)}{\mathcal{A}(I)}\right\} \leq \delta,$$

where $\mathrm{OPT}(I)$ is the value of an optimal solution for I in Π. δ is called *approximation ratio* of $\mathcal{A}$. Note that δ may be a function in the size of the instances. A family of algorithms that contains δ–approximation algorithms for *all* $\delta > 1$ is called a *polynomial approximation scheme*. Given such an

approximation scheme, it is possible to achieve a polynomial algorithm such that the result is as close to the optimal as desired. We say that a problem is *inapproximable* (or *hard to approximate*) within some ratio δ if it is $\mathcal{NP}$-hard to find a δ–approximation algorithm for this problem. See for example Hochbaum (1997); Crescenzi and Kann (1995) for a survey on approximation algorithms and inapproximability results.

A list of some of the $\mathcal{NP}$-complete problems that appear in this thesis, together with a compilation of the respective complexity results and references can be found in Appendix D.

1.4 Roadmap

The basic structure of this thesis mirrors the topics as announced in Section 1.2. We first recapitulate some previous results from the context of k–spanners, and then start the discussion of the four subjects mentioned above. Each of these is treated in its own chapter and is self-contained. In order to maintain a clear structure, in each of these chapters, we first give a motivation and present related concepts, and then elaborate on the respective main results.

Following this line, Chapter 2 gives a detailed introduction to the concept of k–spanners. Chapter 3 comprises the discussion of minimum k–spanners within the context of planarity. In Chapter 4, we discuss our approach to fault-tolerant tree k–spanners. Chapter 5 is devoted to Steiner tree k–spanners. Finally, Chapter 6 contains our results on the construction of survivable networks with bounded delay. As a last chapter, we close the body of this thesis by a concluding review.

The appendix contains a list of notational conventions, glossaries of graph-theoretic and complexity-theoretic terms, and a list of $\mathcal{NP}$-complete problems that are mentioned within this thesis.

Chapter 2

Previous Results on Spanners

The concept of k–spanners has been introduced by Peleg and Ullman (1987), and has been studied widely since. In this chapter, we present an overview on previous results on k–spanners and tree k–spanner that form the base for our research, and which are used in the sequel. In order to emphasize the relevance of this concept, we also discuss some variants and applications.

Observe that we concentrate on the case of *unweighted* and *undirected* graphs. Most definitions easily generalize to *edge weighted* and/or *directed* graphs in a straightforward way. In case of ambiguities, we mention the differences. For extensive surveys, we refer to Peleg and Schaeffer (1989); Soares (1992).

2.1 k–Spanners

Definition 2.1 (k–spanner)
For any rational $k \geq 1$, a spanning subgraph $S = (V, E')$ is a k–spanner *of a graph $G = (V, E)$, if*

$$d_S(u, v) \leq k \cdot d_G(u, v) \quad \text{for all } u, v \in V.$$

The parameter k is called stretch factor.

Observe that we only consider *constant* stretch factors, i.e., stretch factors that are independent of $|V|$ and $|E|$. In most cases, we assume that k is chosen to be the minimal stretch factor that can be achieved for the given subgraph S. Clearly, S is also a k'–spanner of a graph G if S is a k–spanner of G and $k \leq k'$. Since G itself is a 1–spanner of G, it is clear that every graph contains a k–spanner for any k. Note that a (minimum) spanning tree does not necessarily meet this specification, see again Figure 1.2.

We say that an edge e that does not belong to a subgraph S, i.e., an edge e of $E \backslash E(S)$, is *covered* (by S) if there exists a path of length at most k (in S) connecting the end-vertices of e. Such a path is called a *covering path.*

In order to prove that a given spanning subgraph is a k–spanner, we do not have to consider all pairwise distances of the vertices. It suffices to look only at edges of the graph that are not part of the spanning subgraph:

Lemma 2.2 (Peleg and Schaeffer (1989)) *A subgraph $S = (V, E')$ of a graph $G = (V, E)$ is a k–spanner if and only if all edges that do not belong to S are covered, i.e.,*

$$d_S(u, v) \leq k \quad \text{for every edge } e = \{u, v\} \in E \backslash E'.$$

Remark 2.3 *Since distances in unweighted graphs are integral, it follows that S is a k–spanner of G if and only if S is a $\lfloor k \rfloor$–spanner of G. Thus, it suffices to consider* integer *stretch factors.*

In the case of *weighted* graphs, however, the smallest stretch factor may be rational. In the light of computability, we restrict ourselves to rational values instead of real values to enable a reasonable encoding of the instances.

If we look for a k–spanner in a given graph G, it is clear that we only have to consider the *connected* components of G. Furthermore, it even suffices to look at *biconnected* components: Let v be an articulation vertex of G, and G_1, G_2 be two distinct biconnected components of G. Then, no edge in G_1 can be covered by using edges from G_2 and vice versa. Therefore, it suffices to find a k–spanner for every biconnected component separately.

Minimum k–Spanners

A k–spanner is called a *minimum* k–spanner of G if it has a minimum number of edges among all k–spanners of G. We are interested in the corresponding optimization problem, Minimum k–Spanner, and formulate it in terms of the corresponding decision problem:

Problem 2.4 Minimum k–Spanner

Given: A graph G, and a positive integer K.

Problem: Does G contain a k–spanner having at most K edges?

Observe that the stretch factor k is not part of the input; it is a fixed parameter of the problem. In the case of weighted graphs, the quality of a k–spanner

is measured by its total edge weight. In MINIMUM k–SPANNER, we are then searching for a k–spanner with a smallest sum of all edge weights.

Obviously, for an unweighted graph G, the only 1–spanner of G is G itself. Thus, MINIMUM 1–SPANNER is trivial. Regarding weighted graphs, Hakimi and Yau (1964) proved that there is a unique 1–spanner with a minimal number of edges. Moreover, Cai and Corneil (1995) concluded that this must also be the unique 1–spanner with the minimum total edge weight, and that it can be determined in polynomial time. Consequently, MINIMUM 1–SPANNER is in $\mathcal{P}$ also for weighted graphs. For the remaining stretch factors (in both cases), however, MINIMUM k–SPANNER is $\mathcal{NP}$-complete as has been shown in Cai (1994) and Cai and Corneil (1995). To summarize, we have the following theorem for unweighted graphs:

Theorem 2.5 (Peleg and Schaeffer (1989); Cai (1994))

1. MINIMUM 1–SPANNER *is in* $\mathcal{P}$.
2. MINIMUM k–SPANNER *is* $\mathcal{NP}$*-complete for all* $k \geq 2$.

Therefore, subsequent efforts have concentrated on finding spanners that are maybe not minimum, but sufficiently sparse (see, e.g., Althöfer et al. (1993)). For example, Kortsarz and Peleg (1992) give a $(\log \frac{|E|}{|V|})$–approximation algorithm for MINIMUM 2–SPANNER on undirected, unweighted graphs.

Very recently, Kortsarz (1998) has shown that, for unweighted graphs and every $k \geq 2$, MINIMUM k–SPANNER is as hard to approximate as MINIMUM SET COVER (Problem D.8 in Appendix D). Accordingly, there is a constant $c < 1$ such that it is $\mathcal{NP}$-hard to approximate MINIMUM k–SPANNER within ratio $c \cdot \log |V|$.

Venkatesan et al. (1997) prove that MINIMUM k–SPANNER remains $\mathcal{NP}$-hard (for most stretch factors k) even for restricted graph classes such as *chordal*, *bipartite*, or graphs with *bounded degree*. Apart from these hardness results, they also give polynomial algorithms for *split* and *interval* graphs.

For further results on k–spanners on restricted graph classes see for example in Peleg and Ullman (1987); Peleg and Schaeffer (1989); Liestman and Shermer (1993b); Cai and Keil (1994); Cai (1994); Heydemann et al. (1994).

2.2 Tree k–Spanners

Another approach for finding sparse k–spanners is as follows: Instead of searching for a k–spanner that has the fewest edges among all k–spanners, we select spanning subgraphs that have a specified structure, which guarantees sparseness. Now, a new question arises: If we restrict on the structure of

the subgraph, it is not any more guaranteed that a given graph contains a subgraph such that both the distance constraint, and the structure can be achieved. Consequently, if we follow this course, we have to consider the existence problem, too.

As trees are the graphs that have the fewest edges among all connected graphs, it is a natural approach, to consider *tree k–spanner*s, i.e., k–spanners that are trees. Apart from their sparseness, the tree property is of particular interest in several applications. As an example consider communication networks where trees result in small sized routing tables; see e.g., Santoro and Khatib (1985).

When dealing with tree k–spanners, the remarks on integer stretch factors and the restriction to biconnected components apply as in the case of general k–spanners. As opposed to k–spanners (and as indicated above), now there are graphs that, for a fixed stretch factor k, do not contain a tree k–spanner as a subgraph. We say that such a graph does not *admit* a tree k–spanner. For example, $K_{3,3}$ does not admit a tree 2–spanner. We are hence interested in the following decision problem:

Problem 2.6 TREE k–SPANNER

Given: A graph G.

Problem: Does G admit a tree k–spanner?

Again, k is a fixed parameter, and it is not part of the input. Observe, that in the case of tree k–spanners and *unweighted* graphs, we do not have a corresponding optimization problem: Every tree k–spanner of a given graph $G = (V, E)$ contains exactly $|V| - 1$ edges. In the case of *weighted* graphs, however, it is an interesting problem to find a minimum tree k–spanner, i.e., a tree k–spanner with the minimum total edge weight, if one exists at all. But, throughout this thesis we do not elaborate on this.

Regarding the complexity status of TREE k–SPANNER for *unweighted* graphs, we get the following results. As mentioned in the previous subsection, the only 1–spanner of an unweighted graph G is G itself. Thus, G may only admit a tree 1–spanner if G is a tree itself.

For the case $k = 2$, Cai and Corneil (1995) give a linear-time algorithm to find a tree 2–spanner, if one exists. This algorithm is based on the following lemma:

Lemma 2.7 (Bondy (1989))

1. *Let G be a biconnected graph and T an arbitrary tree 2–spanner of G. Then for every 2–separator $\{x, y\}$ of G, edge $\{x, y\}$ is in T.*

2. *An unweighted graph G admits a tree 2–spanner if and only if either*

   (a) *G has a universal vertex, or*

   (b) *every block of G admits a tree 2–spanner, or*

   (c) *$G = G_1 \cup G_2$ such that G_1 and G_2 are two graphs that have exactly one edge e in common, and G_1 and G_2 each admit a tree 2–spanner, where e is an edge in both tree 2–spanners.*

Furthermore, Cai and Corneil (1995) show that TREE k–SPANNER is $\mathcal{NP}$-complete for all $k \geq 4$. The case $k = 3$ is still open. To summarize, we get the following theorem:

Theorem 2.8 (Cai and Corneil (1995))

1. TREE 1–SPANNER *and* TREE 2–SPANNER *are in* $\mathcal{P}$.

2. TREE k–SPANNER *is* $\mathcal{NP}$*-complete for all* $k \geq 4$.

In the same paper, Cai and Corneil also show that TREE 1–SPANNER can be solved in polynomial time for *weighted* graphs, whereas the problem for all remaining stretch factors is $\mathcal{NP}$-complete.

Observe that, whenever MINIMUM k–SPANNER is polynomial solvable for a specific graph class, then also TREE k–SPANNER is polynomial solvable for this graph class (though, of course, the converse is not true). Thus, in particular the positive results of Venkatesan et al. (1997) for MINIMUM k–SPANNER as mentioned above can be applied here. See also Madanlal et al. (1997); Brandstädt et al. (1999a) for results specifically for TREE k–SPANNER.

Recently, and partially based on our approach in Chapter 3.2, Fekete and Kremer (1998) have shown that it is $\mathcal{NP}$-hard to find a minimum k such that a given *planar* graph admits a tree k–spanner. On the other hand, they give a polynomial algorithm to decide (and to construct) whether a *planar* graph admits a tree 3–spanner. For other fixed stretch factors of $k \geq 4$, TREE k–SPANNER for planar graphs remains open.

Very recently in his Masters' thesis, Papoutsakis (1999) has shown some nice properties of graphs that admit a tree k–spanner in terms of decomposition, and on the number of edges that are not included in a tree k–spanner.

2.3 Variants and Applications of k–Spanners

In this section, we mention some of the variants and applications that have appeared within the context of k–spanners. Note however that we do not intend to give a complete overview.

Variants of k–Spanners

Instead of constant stretch factors, some authors have also considered stretch factors that are functions on the size of the graph. More formally, for a function $f : \mathbb{N} \to \mathbb{R}^{\geq 1}$, a spanning subgraph $S = (V, E')$ is an $f(|V|)$–*spanner* of a graph $G = (V, E)$, if

$$\frac{d_S(u,v)}{d_G(u,v)} \leq f(|V|) \quad \text{for all } u, v \in V.$$

Peleg and Schaeffer (1989), show for example that every graph $G = (V, E)$ contains an $\mathcal{O}(\log |V|)$–spanner with $\mathcal{O}(|V|)$ edges. Moreover, this spanner can be constructed in polynomial time.

Another significant parameter in network design, apart from sparseness, is for example the maximum degree of a vertex. See for example Kortsarz and Peleg (1994) for an extensive study of *low-degree* 2–spanners.

Liestman and Shermer (1991, 1993a) introduce *additive spanners*, a variant of k–spanners: instead of considering multiplicative distance guarantees, they consider additive delays: A spanning subgraph $S = (V, E')$ is an *additive* k–*spanner* of a graph $G = (V, E)$, if

$$d_S(u,v) \leq d_G(u,v) + k \quad \text{for all } u, v \in V.$$

Observe that the concept of additive k–spanners is more rigid than that of multiplicative k–spanners in the following sense: whenever a subgraph S is an additive k–spanner for some fixed k, then S is a also a multiplicative k–spanner, but not vice versa. In particular, an additive 0–spanner is also a multiplicative 1–spanner. See also Farley and Proskurowski (1993); Kratsch et al. (1998) for results on additive (tree) spanners.

Cai (1992) combines the two concepts of additive and multiplicative spanners, and strengthens the distance constraint of k–spanners as follows: A spanning subgraph $S = (V, E')$ is an (ℓ, k)–*spanner* of a graph $G = (V, E)$, if

$$d_S(u,v) \leq \begin{cases} k & \text{for all edges } \{u,v\} \in E \text{ and} \\ d_G(u,v) + \ell & \text{otherwise} \end{cases}$$

for all $u, v \in V$.

Very recently, Brandstädt et al. (1999a) discuss *distance* (t, r)–*approximating graphs*, which generalize on the concept of additive and multiplicative spanners: For two real numbers t and r, a graph $H = (V, E')$ is a *distance* (t, r)–*approximating graph* of a graph $G = (V, E)$ if

$$\begin{aligned} d_H(u,v) &\leq t \cdot d_G(u,v) + r \quad \text{and} \\ d_G(u,v) &\leq t \cdot d_H(u,v) + r \quad \text{and} \end{aligned}$$

for all $u, v \in V$. Observe that H is not necessarily a subgraph of G, but any multiplicative k–spanner is a distance $(k, 0)$–approximating graph, and any additive k–spanner is a distance $(1, k)$–approximating graph.

Throughout this thesis, we always deal with multiplicative spanners and constant stretch factors, and one of our objectives is sparseness, i.e., we want to find k–spanners with a small number of edges.

Applications of k–Spanners

The concept of k–spanners has been introduced by Peleg and Ullman (1987), where they used spanners to synchronize asynchronous networks. A similar notion appeared in Chew (1986) in the study of an approximation of complete Euclidean graphs.

Since then, k–spanners have been used in many different areas, ranging from communication networks, distributed systems, computational geometry, robotics to computational biology, just to mention a few (Peleg and Upfal (1988); Kortsarz and Peleg (1994); Venkatesan et al. (1997); Bandelt and Dress (1986)).

As an example, k–spanners can be used in the design of networks for parallel computation (Liestman and Shermer (1993b); Heydemann et al. (1994); Richards and Liestman (1995)). In the context of weighted graphs, research has concentrated in particular on k–spanners in geometric context, where the edge weights are determined by the Euclidean distance between the endvertices. Here, we refer to Chew (1986); Chandra et al. (1992); Althöfer et al. (1993); Arya et al. (1995); Levcopoulos et al. (1998) and the references there-in. Extensive surveys on further topics on k–spanners can be found in Peleg and Schaeffer (1989); Soares (1992).

Chapter 3

Minimum Spanners and Planarity

As stated in Section 2.1, Minimum k–Spanner is $\mathcal{NP}$-complete for all stretch factors k but the trivial one. It is thus reasonable to examine the complexity status of this problem for restricted graph classes or modified problem descriptions.

3.1 Introducing Planarity to the Concept of Spanners

In the light of the hardness of Minimum k–Spanner in general, we now consider k–spanners within the context of planarity. Recall that a graph is planar if it can be embedded in the plane without edge crossings. We study three different topics as outlined in the sequel.

Restriction to Planar Instances

Many graph problems that are $\mathcal{NP}$-complete for general graphs remain $\mathcal{NP}$-complete when restricted to planar graphs. As examples consider Chromatic Number (Problem D.2 in Appendix D) or Minimum Steiner Tree (Problem D.4). On the other hand, for example Maximum Cut (Problem D.5) or Maximum Clique (Problem D.3) are polynomially solvable for planar graphs. See for example Johnson (1985) for a more detailed discussion. In Section 3.2, we examine Minimum k–Spanner when restricted to *planar* graphs, and it turns out that the complexity status remains hopeless for planar instances for most stretch factors, be it on unweighted, weighted or directed graphs.

Outerplanar Instances

Motivated by the hardness of MINIMUM k–SPANNER on planar graphs, we turn our attention to further subclasses of planar graphs, namely to *ℓ–outerplanar*, and in particular, outerplanar graphs. Informally, a graph is called *ℓ–outerplanar* if it can be embedded in the plane without edge crossings such that it consists of at most ℓ disjoint cycles that are properly nested. A formal definition is given in Section 3.3 below. As indicated in Johnson (1985), many problems that are $\mathcal{NP}$-complete on planar graphs turn polynomially solvable for ℓ–outerplanar graphs (e.g., CHROMATIC NUMBER, Problem D.2). We will see in Section 3.3 that also MINIMUM k–SPANNER can be solved efficiently for these graphs.

Planar Spanners

As a third topic, we study an approach that is motivated by similar arguments as in the case of tree k–spanners (Section 2.2): Instead of searching for a k–spanner with the fewest edges among all k–spanners, we restrict the choice of the spanning subgraph, such that it is guaranteed to be sparse. For this, we examine *planar k–spanner*s, i.e., k–spanners that are planar.

Compared with tree k–spanners, planar k–spanners usually contain more edges. But they are still sparse since the number of edges is linear in the number of vertices. This holds in particular, when minimum planar k–spanners are considered, i.e., planar k–spanners with the fewest edges among all planar k–spanners.

Observe that the structure of a planar k–spanner is not as simple as that of a tree k–spanner. But on the other hand, the class of graphs that admit a planar k–spanner for a fixed k is larger than the class of graphs that admit a tree k–spanner for the same k. Therefore, the approach of searching for planar k–spanners is suitable for more graphs.

Furthermore, a (minimum) planar k–spanner may serve as a sparse planar representative of a non-planar graph, such that distances within the representative do not increase too much. This can be used for example within the context of problems that are $\mathcal{NP}$-complete for general graphs, but that are polynomially solvable for planar graphs (e.g., Problem D.5 MAXIMUM CUT or Problem D.3 MAXIMUM CLIQUE), or for problems that can be solved much more efficiently when restricted to planar graphs (e.g., finding shortest paths, see Klein et al. (1994)).

Unfortunately, it turns out in Section 3.4, as a consequence from results on tree k–spanners, that the problem of finding minimum planar k–spanners is $\mathcal{NP}$-hard for most values of k.

Some Notation for Planar Graphs

Before going into detail, we first recall some more notation on planar graphs. A planar embedding of a planar graph G separates the plane into regions called *faces*. The unbounded face is the *outer face*, all others are *inner faces*. Every edge is incident to one or two faces; in biconnected planar graphs, every edge is incident to exactly two faces. Two faces are *adjacent* if there is an edge that is incident to both of them. Let f be an inner face; we denote by $|f|$ the *size* of f, i.e., the number of edges that are incident to f. The number of edges in a planar graph $G = (V, E)$ is bounded by $|E| \leq 3|V| - 6$ for $|V| \geq 3$.

3.2 Minimum k–Spanners in Planar Graphs

In this section, we show that MINIMUM k–SPANNER remains $\mathcal{NP}$-complete for most values of k, even when G is restricted to be planar. In particular, we prove the following theorem. Recall that a directed graph is called *oriented* if it does not contain a pair of anti-parallel arcs.

Theorem 3.1

1. *For any fixed integer $k \geq 5$,* MINIMUM k–SPANNER *is $\mathcal{NP}$-complete for undirected, unweighted, planar, biconnected graphs.*
2. *For any fixed integer $k \geq 3$,* MINIMUM k–SPANNER *is $\mathcal{NP}$-complete for undirected, weighted, planar, biconnected graphs with edge weights equal to 1 or 2.*
3. *For any fixed integer $k \geq 5$ ($k \geq 3$),* MINIMUM k–SPANNER *is $\mathcal{NP}$-complete for unweighted (weighted) planar oriented graphs.*

The proofs of the three parts of the theorem are given in the next three subsections. All three are transformations from PLANAR 3–SATISFIABILITY, and they follow the same lines. We thus describe the proof technique in detail only for Part 1 of Theorem 3.1, the case of unweighted graphs. For the remaining two parts, we restrict ourselves to the necessary modifications.

Note that, as stated in Remark 2.3, in an unweighted graph G, a k–spanner of G is also a $\lfloor k \rfloor$–spanner of G. But there is no such correspondence in weighted graphs, even if all edge weights are integer. Accordingly, in the case of *weighted* graphs *rational* stretch factors do matter. Theorem 3.1, however, is only stated for integer stretch factors. But at the end of Section 3.2.2, it is easy to see how our construction can be adjusted to allow arbitrary rational values for $k \geq 3$ in the weighted case. Thus, Theorem 3.1 also holds for rational stretch factors.

3.2.1 Unweighted Planar Graphs

In this section, we prove Part 1 of Theorem 3.1, so all graphs are unweighted and planar. A part of the proof is based on the ideas of Cai (1994).

Let $k \geq 5$ be an arbitrary fixed integer. Clearly, Minimum k–Spanner is in $\mathcal{NP}$, because the test whether a spanning subgraph S is a k–spanner can be done in polynomial time. Following Lemma 2.2, we just have to check the (at most linear number of) edges of G that do not belong to S.

To show the $\mathcal{NP}$-completeness, we transform Planar 3–Satisfiability to Minimum k–Spanner. Planar 3–Satisfiability (Problem D.10 in Appendix D) is a variant of 3–Satisfiability (Problem D.10), which is defined as follows: An instance (U, C) consists of a set of Boolean variables U and a set C of clauses over literals of U, such that every clause contains exactly three literals. The problem is to decide whether there is a truth assignment for U such that every clause in C is fulfilled. In Planar 3–Satisfiability, we have the additional restriction that the underlying bipartite graph consisting of clause vertices (corresponding to the clauses) and variable vertices (corresponding to the variables), such that clause vertices are connected to variable vertices if the corresponding variables appear within the clause, is planar.

We use the planarity of the underlying graph of Planar 3–Satisfiability to construct a planar graph in which a minimum k–spanner can be determined easily.

Forcing Edges into a Minimum k–Spanner

For the construction of the instance of Minimum k–Spanner from the instance of Planar 3–Satisfiability, we use the fact that we can force edges to be in every minimum k–spanner:

Lemma 3.2 (Cai (1994)) *Let e be an arbitrary edge of a graph G, and let G' be the graph constructed from G by adding two disjoint paths P_1 and P_2 of length k (all internal vertices of P_1 and P_2 are new vertices) between the end-vertices of e. Then for any minimum k–spanner S of G', edge e belongs to S.*

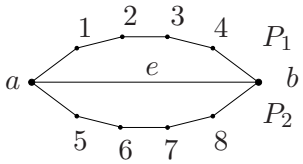

Figure 3.1: Forcing $\{a, b\}$ into a minimum 5–spanner.

The two auxiliary paths P_1 and P_2 are called *forcing paths*, edge e is called *forced edge*. A *forced ℓ–component* is a simple path of length ℓ consisting of ℓ forced edges together with their forcing paths. For an example of a forced edge $e = \{a, b\}$ (and

thus a forced 1–component) for $k = 5$, see Figure 3.1. A minimum k–spanner of a forced ℓ–component contains exactly $\ell \cdot (2 \cdot (k-1)+1) = \ell(2k-1)$ edges: the ℓ forced edges and $k-1$ edges from each of the forcing paths.

Construction of the Instance

We start from the planar, embedded graph underlying the given instance (U, C) of Planar 3–Satisfiability, and extend the variable and clause vertices to form *variable components* and *clause components*. These components are combined to form *truth assignment testing components*, which reflect the relationship between the satisfiability of a clause and the existence of a minimum k–spanner. Finally, we compute a bound on the number of edges for Minimum k–Spanner.

Variable components. Our goal is to build the variable components such that a minimum k–spanner of such a component reflects a truth assignment for the corresponding variable. For every variable $x \in U$, construct a variable component T_x as follows. Let ℓ be the number of (positive and negative) occurrences of the variable x in all clauses.

- Create a central vertex x^*.
- For each occurrence of x in a clause c, create a block of four new vertices $x_1^{(c)}$, $\overline{x}_1^{(c)}$, $x_2^{(c)}$, and $\overline{x}_2^{(c)}$ (in this order). The resulting 4ℓ new vertices are called *literal vertices*. The blocks are arranged circularly around x^* according to the embedding of the underlying graph of the instance of Planar 3–Satisfiability.
- Connect every pair of neighboring literal vertices by a forced $(k-1)$–component, resulting in a circle of 4ℓ forced $(k-1)$–components altogether.
- Connect x^* to all literal vertices by an edge, called *literal edge*. An edge $\{x_i^{(c)}, x^*\}$ is called *positive* literal edge, an edge $\{\overline{x}_i^{(c)}, x^*\}$ is called *negative* literal edge.
- Create 4ℓ new auxiliary vertices, one between every pair of neighboring literal vertices. Connect each of these by an *auxiliary* edge with x^* and by two distinct forced $(k-1)$–components with the neighboring literal vertices. The literal edges of the neighboring literal vertices are called *associated* literal edges of the auxiliary edge and vice versa.

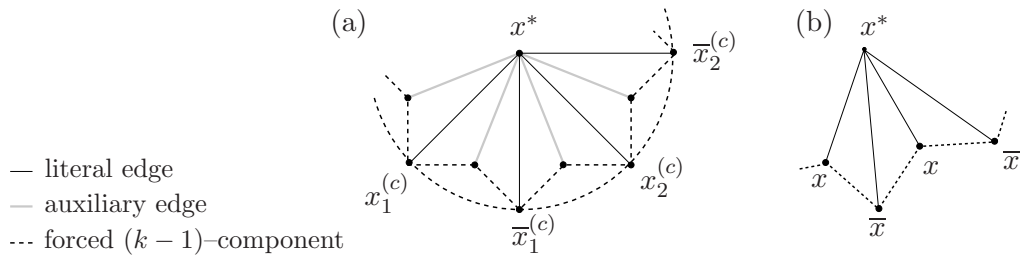

Figure 3.2: (a) Part of the variable component T_x for the variable x occurring in clause c, (b) its symbolic representation.

Figure 3.2 illustrates this construction. For readability, the symbolic representation in Figure 3.2(b) is used later on when larger portions of the graph are drawn. The following lemma shows that the literal edges in a minimum k–spanner are *consistent*:

Lemma 3.3 *Any minimum k–spanner of a variable component T_x contains either all positive or all negative literal edges.*

Proof Let S be an arbitrary minimum k–spanner of T_x. Then S contains all forced edges and $k-1$ edges from each forcing path. Observe that these edges together with either all 2ℓ positive or all 2ℓ negative literal edges form a k–spanner. Therefore, S can contain at most 2ℓ edges out of the 8ℓ existing literal and auxiliary edges.

Furthermore note the following: A literal edge (together with the forced edges) covers its two associated auxiliary edges and its two neighboring literal edges. An auxiliary edge, however, covers only its associated literal edges, but no further auxiliary edge.

First, we show that S does not contain an auxiliary edge by contradiction: If S contains an auxiliary edge, then S also contains either the next auxiliary edge or the next non-associated literal edge. In total, this leads to more than 2ℓ additional edges and hence contradicts the optimality of S.

Similarly, if S contains two inconsistent literal edges, at least one auxiliary edge or more than 2ℓ literal edges are needed to cover all other edges. Again, this contradicts the optimality of S. Consequently, S contains only consistent literal edges, i.e., every other literal edge. □

Clause components. The clause component for each clause $c \in C$ is basically a quadrilateral consisting of four *clause vertices* 1, 2, 3, and 4, where the sides are formed by distinct forced $(k-2)$–components. Vertices 1 and 3 are connected by an additional edge, called the *clause edge*, see Figure 3.3(a).

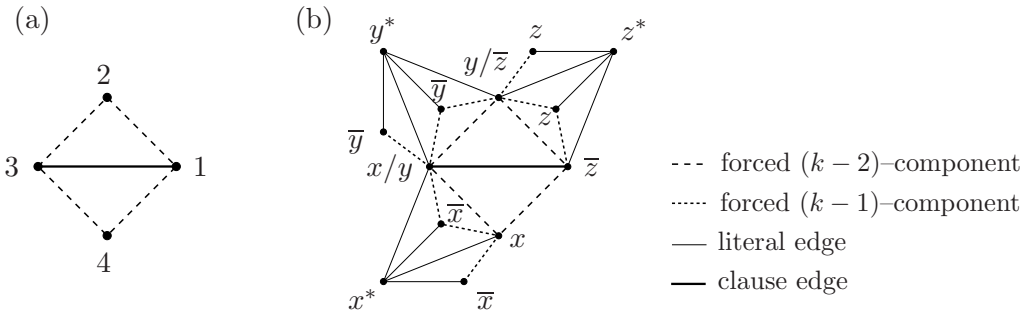

Figure 3.3: (a)A clause component, and (b) the truth assignment testing component for clause $c = x \vee y \vee \overline{z}$ using the symbolic representation for relevant blocks of the variable components.

Observe that any minimum k–spanner for $k \geq 5$ of such an isolated clause component must contain the clause edge. Moreover, we should note that our construction is a bit more complex than actually needed in the unweighted case, but does not have to be changed much when being modified for the weighted and the directed case.

Truth assignment testing components. We combine the variable components with the clause components according to the given clauses by identifying vertices. In particular, each of the three literals in a clause corresponds to one side of the quadrilateral in the clause component (the fourth side is used to make the arguments symmetric). Now, if a clause c contains literal x (or $\overline{x}$, respectively) we melt together literal vertex $x_1^{(c)}$ (or $\overline{x}_1^{(c)}$) with the first end-vertex, and literal vertex $x_2^{(c)}$ (or $\overline{x}_2^{(c)}$) with the second end-vertex of the corresponding side of the clause component, see Figure 3.3(b).

Note that the combination of the variable components with the clause components does not affect the validity of Lemma 3.3. Consequently, the number of edges in a minimum k–spanner of such a truth assignment testing component reflects the truth value of the corresponding clause:

Lemma 3.4 *For any fixed integer $k \geq 5$, a minimum k–spanner S of a truth assignment testing component contains the clause edge if and only if S contains no pair of consistent literal edges that is adjacent to the clause edge.*

Proof If S does not contain a pair of literal edges that is adjacent to the clause edge then every path connecting the end-vertices of the clause edge in S either uses the clause edge or has length at least $2(k-2) > k$, if $k \geq 5$.

For the other direction, assume that S contains a pair of adjacent consistent literal edges. Then this provides a shortcut for one of the forced

$(k-2)$–components, and thus there is a path of length $2+(k-2)=k$ in S connecting the end-vertices of the clause edge. Hence, the clause edge is covered. □

This completes the construction of the graph. All isolated components are planar, and since we start from an instance of PLANAR 3–SATISFIABILITY, the whole graph is planar. It is also easily seen that the instance is biconnected, and can be constructed in polynomial time.

Choice of K. Let m be the number of clauses in the instance of PLANAR 3–SATISFIABILITY. According to Lemmas 3.3 and 3.4, we set K, the bound on the number of edges in a k–spanner, to

$$K = 6m + 36m(k-1)(2k-1) + 4m(k-2)(2k-1).$$

Equivalence of the Problems

In this subsection, let (U, C) be an instance of PLANAR 3–SATISFIABILITY, and (G, K) the instance for MINIMUM k–SPANNER constructed as described above. We show that there is a satisfying truth assignment for (U, C), if and only if G has a k–spanner with at most K edges.

Lemma 3.5 *If the set of clauses C is satisfiable, then there exists a planar k–spanner of G with at most K edges.*

Proof Suppose that C is satisfiable, and let θ be a satisfying truth assignment. From this we construct the subgraph S of G as follows:

- S contains all forced edges.
- S contains $k-1$ arbitrarily chosen edges from every forcing path.
- For every variable $x \in U$, if $\theta(x) = \texttt{true}$ then S contains all positive literal edges. Otherwise, S contains all negative literal edges.

Observe that the total number of variable occurrences is $3m$. By construction, S is a spanning subgraph. The number of edges in S computes as:

- $3m \cdot 3 \cdot 4(k-1)(2k-1)$ (forced $(k-1)$–components of variable comp.)
- $m \cdot 4(k-2)(2k-1)$ (forced $(k-2)$–components of clause comp.)
- $3m \cdot 2$ (literal edges)

Altogether the number of edges in S is exactly K. It remains to prove that S is a k–spanner of G:

According to Lemma 2.2, we have to show that for every edge that is not contained in S, there exists a path of length at most k connecting the end-vertices of that edge. This is obvious for the variable components. For the clause edges observe that, since θ is a satisfying truth assignment, there is at least one literal in every clause that is true. Due to the construction of S we have at least one adjacent pair of literal edges in every clause component. As a consequence of Lemma 3.4, S is a k–spanner. □

Observe that any minimum k–spanner S of G contains at least K edges. This can be seen as follows: Any k–spanner S of G must contain all forced edges and $k-1$ edges from every forcing path. By Lemma 3.3, S contains at least either all positive or all negative literal edges for every variable component. Altogether, this sums up to K.

Finally, Lemma 3.6 completes the proof of Part 1 of Theorem 3.1.

Lemma 3.6 *If G has a k–spanner with at most K edges, then there exists a satisfying truth assignment for (U, C).*

Proof If S is a k–spanner of G with at most K edges, then, by the previous observation, S is minimum and has exactly K edges. All forced edges and the edges from the corresponding forcing paths must be in S. Hence, there remain only $6m$ further edges, which can only be consistent literal edges (by Lemma 3.3). Consequently, we can uniquely define a truth assignment θ by setting, for every $x \in U$, $\theta(x) = \texttt{true}$, if S contains the positive literal edges of T_x, and $\theta(x) = \texttt{false}$ otherwise.

Since S is a k–spanner and S contains no clause edge it follows from Lemma 3.4 that there is at least one adjacent pair of literal edges for every clause edge. Hence, θ satisfies all clauses. □

3.2.2 Weighted Planar Graphs

We now turn from unweighted to edge weighted graphs, and the goal is to find a k–spanner with minimal total edge weight. For the proof of the second part of Theorem 3.1, we again transform an instance of Planar 3–Satisfiability to an instance of Minimum k–Spanner by extending variable and clause vertices to appropriate components.

The variable components are the same with all edges having unit edge weight, and the results about minimum k–spanners for these components

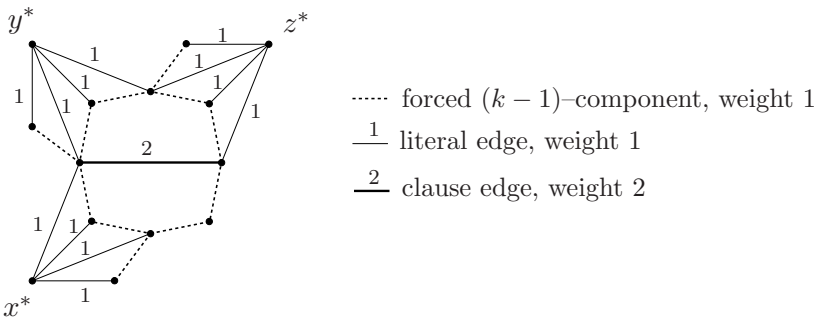

Figure 3.4: The truth assignment testing component in the weighted case.

remain valid (Lemma 3.3). The clause components again consist of four clause vertices, but now three sides of the quadrilateral remain unconnected. Only one side is connected by two consecutive forced $(k-1)$–components with unit edge weights. As before, we have one clause edge, now having edge weight 2. We combine the components to form the truth assignment testing components as we did in the unweighted case by identifying the corresponding vertices (see Figure 3.4). Using similar arguments as in Section 3.2.1, we get an equivalent of Lemma 3.4 now for all values of $k \geq 3$.

The constructed graph is again planar and biconnected. By choosing $K = 6m + 38m(k-1)(2k-1)$, and using similar arguments as before, the proof of Part 2 of Theorem 3.1 is complete.

Theorem 3.1 can be generalized to allow rational stretch factors $k \geq 3$ by using forced $(\lfloor k \rfloor - 1)$–components in the construction described above. All results about minimum k–spanners then keep valid. Consequently, Theorem 3.1 also holds for rational stretch factors.

Observe that the assignment of edge weight 2 in the construction has helped to lower the bound on k, thus yielding stronger results than in the unweighted case. But we cannot expect to exploit this even further.

3.2.3 Planar Digraphs

In this section, we prove Part 3 of Theorem 3.1, and consider the case of directed graphs (digraphs). For the definition of a k–spanner S in a digraph G, note that S has to contain an oriented path of length at most k for any arc of G, i.e., covering paths have to be oriented (see, e.g., Figure 3.5, where arc e is not covered by any of the two paths, but arc (c, b) is).

Here, we only give the details for the case of unweighted digraphs. In the weighted case, the same variable components as described below can be used (with unit arc weights). The clause components are the ones from the weighted, undirected case, and orientations are determined analogously.

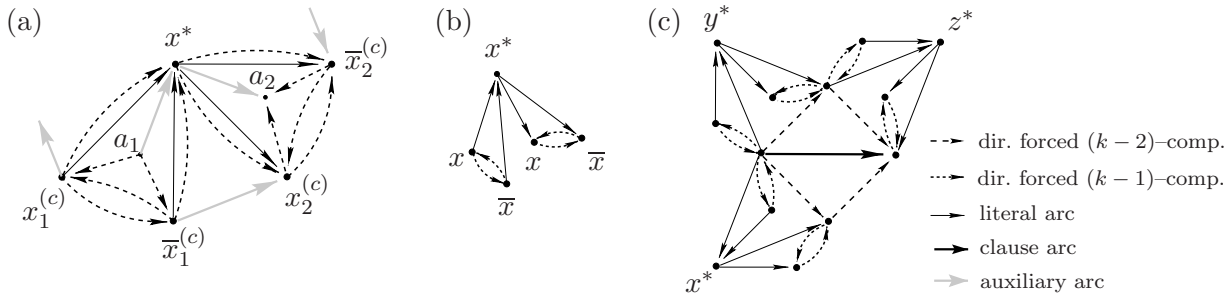

Figure 3.6: (a) Part of the variable component for the variable x occurring in clause c, (b) its symbolic representation, and (c) the truth assignment testing component for unweighted digraphs.

Forcing arcs into a minimum k–spanner. Similar to the undirected case, an arc (a, b) of a digraph can be forced to be in every minimum k–spanner as described in Cai (1994). For this purpose we create two new vertices c and d, add two arcs (c, b) and (d, b), and then add two distinct directed paths (called *covering paths*) of length $k-1$ from c to a and from d to a, respectively. Then, a minimum k–spanner of this component consists of arc (a, b) and all arcs of the paths of length $k-1$. Figure 3.5 shows an example for $k = 5$. The definition of forced ℓ–components carries over from the undirected case.

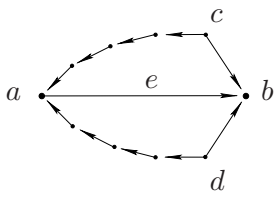

Figure 3.5: Forcing (a, b) into a 5–spanner.

Construction of the instance. For the construction of the instance from Planar 3–Satisfiability see Figure 3.6. The *variable components* (see Figure 3.6(a)) again consist of literal and auxiliary vertices, as well as literal and auxiliary arcs. The orientation of these arcs depends on the orientation of the corresponding clause arc. As in the undirected case this construction guarantees that every minimum k–spanner of a variable component only contains consistent literal arcs (cf. Lemma 3.3). This follows from the following observations: All positive (or negative, respectively) literal arcs together with the appropriate arcs from the forced components form a k–spanner. By construction of the auxiliary arcs, at least every other literal arc has to be in a k–spanner. Moreover, no auxiliary arc is covered by an other auxiliary arc.

The *clause components* are analogous to the undirected case, where the clause arc and the forced $(k-2)$–components are oriented such that they start and end at the same vertices of the quadrilateral. Figure 3.6(c) shows an example of a directed truth assignment testing component.

The constructed graph is planar and oriented. Choosing K as in the undirected case, $K = 6m + 36m(k-1)(2k-1) + 4m(k-2)(2k-1)$, the proof of the equivalence of Planar 3–Satisfiability and Minimum k–Spanner is straightforward as before. This completes the proof of Theorem 3.1.

3.3 Minimum k–Spanners in ℓ–Outerplanar Graphs

In the previous section, we have seen that Minimum k–Spanner is hard for most stretch factors, even when restricted to planar graphs. We now study ℓ–*outerplanar* and *outerplanar* graphs, i.e., proper subclasses of planar graphs.

Formally, ℓ–outerplanar graphs are defined as follows: Given a planar embedding of a planar graph G, we label the vertices as follows: Vertices on the outer face are labeled 1. A vertex v has label i if it is on the outer face of the embedded subgraph in which all vertices labeled $i-1$ and lower have been deleted. The labels of the vertices can be computed in linear time using the algorithm of Lipton and Tarjan (1979). A graph is called ℓ–*outerplanar* if it has a planar embedding such that ℓ is the maximum label of a vertex.

The term *outerplanar* simply means 1–outerplanar, and it describes the class of graphs that can be drawn in the plane without edge crossings such that all vertices are incident to the outer face.

Given a planar graph $G = (V, E)$, an ℓ–outerplanar embedding of G such that ℓ is minimal can be found in polynomial time, see Bienstock and Monma (1990). But observe that the degree of outerplanarity of a planar graph G, i.e., the smallest ℓ such that there is an ℓ–outerplanar embedding for G, is in the order of the number of the vertices. When dealing with ℓ–outerplanar graphs here, we assume ℓ to be a constant parameter. Furthermore, we consider an ℓ–outerplanar graph together with an ℓ–outerplanar embedding.

In the following subsection, we discuss the fact that we can use the general framework of extended monadic second order logic to solve Minimum k–Spanner for ℓ–outerplanar graphs. But since this general approach turns out to be impractical, in the subsequent subsection, we show how to construct a minimum k–spanner directly at least for the case of outerplanar graphs.

3.3.1 Minimum k–Spanners Via Monadic Second Order Logic

ℓ–outerplanar graphs, and in particular outerplanar graphs, have strong structural properties, which make them much easier to handle than planar

graphs. For example, ℓ–*outerplanar* graphs have bounded *treewidth*, which is defined as follows:

A *tree decomposition* of a graph $G = (V, E)$ is a pair (X, T) in which $T = (I, F)$ is a tree and $X = \{X_i \mid i \in I\}$ is a family of subsets of V, one for each vertex $i \in I$ of T, such that

- $\bigcup_{i \in I} X_i = V$;
- for all edges $\{u, v\} \in E$, there is an $i \in I$ with $u \in X_i$ and $v \in X_i$;
- for all $i, j, k \in I$: if j is on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$.

The *width* of a tree decomposition (X, T) is $(\max_{i \in I} |X_i|) - 1$. The *treewidth of* G is the minimum width among all tree decompositions of G. The notion of treewidth has been introduced by Robertson and Seymour (1983, 1986). See also Bodlaender (1993, 1997, 1998) for further results on the treewidth of graphs.

Lemma 3.7 (Bodlaender (1998))

1. *Every outerplanar graph has treewidth at most* 2.
2. *Every ℓ–outerplanar graph has treewidth at most* $3\ell - 1$.

We can use the bounded treewidth of ℓ–outerplanar graphs in the following context: As shown in Rotics (1998), MINIMUM k–SPANNER (like many other decision and optimization problems on graphs) can be formulated within *extended monadic second order logic*, i.e., by using the following constructions:

- quantification over vertices, edges, sets of vertices, sets of edges;
- membership and adjacency tests; and
- logic operations.

Extensions allow for example to optimize over the size of a free set variable. More details and formal definitions of this can be found for example in Courcelle (1997); Papadimitriou (1994).

Every problem that can be defined in terms of extended monadic second order logic can be solved in linear time on graphs of bounded treewidth (see e.g., Courcelle (1990); Arnborg et al. (1991); Courcelle and Mosbah (1993); Bodlaender (1996)). As a consequence, MINIMUM k–SPANNER can be solved in linear time on ℓ–outerplanar graphs.

Unfortunately, the constants used in this general framework are so exorbitant that the algorithms derived by that approach are not practical. In the following section, we give a direct, linear algorithm for finding a minimum k–spanner in outerplanar graphs.

3.3.2 An Algorithm for Outerplanar Graphs

We now turn back to the case of *outerplanar* (i.e., 1–outerplanar) graphs, and show how to find a minimum k–spanner in such graphs. By this, we also solve Tree k–Spanner for outerplanar graphs, because we only have to check whether the minimum k–spanner found by the algorithm is a tree.

Within this section, we assume that we are given an outerplanar graph G together with an outerplanar embedding. As in the general case, it suffices to consider biconnected components.

Before going into details, we compile some more properties of *outerplanar* graphs $G = (V, E)$ (see Sysło (1979) for a survey): The number of edges is at most $2|V| - 3$, and the number of inner faces is at most $|V| - 2$. An outerplanar graph is at most biconnected, every block of an outerplanar graph is again outerplanar, and every biconnected outerplanar graph has a unique outerplanar embedding.

Given a biconnected outerplanar graph G, every edge of G is either

- a *chord*, which is incident to two inner faces, or
- a *boundary edge*, which is incident to one inner face and the outer face.

For a biconnected outerplanar graph G and a set F of inner faces of G, denote by $G - F$ the graph obtained from G by deleting all faces of F, i.e., by deleting all edges that are only incident with faces of F or the outer face. Observe that an edge that is a chord between a face of F and another inner face that is not in F becomes a boundary edge in $G - F$.

We make use of the internal dual graph: The *internal dual*, denoted by T, is constructed as the usual geometric dual graph (cf. Appendix B), but now we only consider inner faces: For each inner face of G create a node in T; two nodes are adjacent in T if the corresponding faces are adjacent in G. By this, every chord of G corresponds to an edge of T. Observe that if G is biconnected and outerplanar, then T is a tree, and hence paths within T are unique. We abuse notation f to denote either a face in G or a node in T.

In the following, we sometimes consider T to be rooted at a specified root vertex r. For a vertex v, let w be the next vertex on the path from v to r. Then w is called *ancestor* of v, and v is called *successor* of w.

Before we deal with the general case, we first discuss two special cases that allow for breaking down the given instance into smaller pieces: First we consider *big faces*, then *chains*, and finally turn to general outerplanar graphs that do not contain both.

Big Faces

Given a biconnected outerplanar graph, the next lemma states the following observation: The number of boundary edges that may miss in a k–spanner is bounded, in particular for faces with more than $k+1$ incident edges.

Lemma 3.8 *Let S be a k–spanner of a biconnected outerplanar graph G. Then, for every inner face f of G, the following holds:*

1. *At most one boundary edge of f does not belong to S.*
2. *If $|f| > k+1$ then all boundary edges of f belong to S.*

Proof (By contradiction.)

1. Suppose that f has at least two boundary edges and that two of these, say e_1 and e_2, do not belong to S. Since G is biconnected and outerplanar, $G - \{e_1, e_2\}$ is disconnected and so is S, a contradiction to S being connected.
2. Let f be a face with $|f| > k+1$ and suppose that there is a boundary edge $e = \{u, v\}$ that does not belong to S. As e is a boundary edge, it is incident only to f and the outer face. A shortest path connecting u and v that does not use e hence follows the boundary of f. Then $d_{G-e}(u, v) > k$ because $|f| > k+1$, and thus, e cannot be covered in $G - e$, a contradiction to S being a k–spanner. □

Observe that the number of chords that are contained in a k–spanner may vary for each face. In the sequel, we call a face f *big* if $|f| > k+1$. Otherwise f is called *small*. Part 2 of the previous lemma justifies the splitting process Split-Big-Face for a big face f:

Procedure Split-Big-Face(f)

Input: An outerplanar graph G, and a face f with $|f| > k+1$.

Output: A collection of disconnected components of G obtained by splitting G along f.

Algorithm:

1. Delete all boundary edges of f from G.
2. Split G along f:
   For every vertex u that is incident to two chords of f do the following: split u into two non-adjacent, new vertices u_1 and u_2, such that G disintegrates into biconnected components.

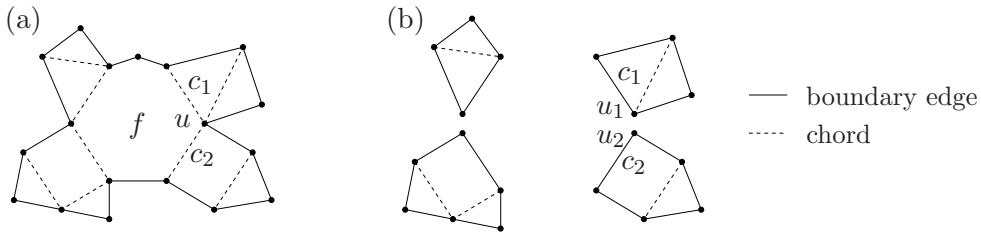

Figure 3.7: (a) G contains a big face f for $k = 5$; (b) the components of G that arise by splitting along f.

See Figure 3.7 for an illustration of the splitting process for the split of G along f. Observe that chords of f become boundary edges in their respective component. We use this procedure to construct a minimum k–spanner for G from minimum k–spanners of the new components:

Lemma 3.9 *Let G be a biconnected outerplanar graph, f an inner face with $|f| > k + 1$, and let $G_1, \ldots, G_\ell$, where $\ell \leq |f|$, be the components obtained by splitting G along f as in* Split-Big-Face(f). *Let S_i, where $1 \leq i \leq \ell$, be a minimum k–spanner of G_i. Then, the subgraph S of G that is obtained by recombining the S_i according to the splitting process and by adding all boundary edges of f is a minimum k–spanner of G.*

Proof By Part 2 of Lemma 3.8, the boundary edges of f belong to any k–spanner of G. Now consider two components G_1 and G_2 that are obtained in the splitting process, and let c_1 be the chord of f that is a boundary edge in G_1. As f is a big face, neither c_1 nor any other edge of G_1 can be used to cover any edge in G_2. The covering path would be far too long. Consequently, each of the new components can be treated independently of the others. □

Note that each of the new components is biconnected because all inner faces but the big face are retained. By using Split-Big-Face for all big faces, we end up with a biconnected outerplanar graph having only small faces.

Chains

In the sequel, we assume that G consists only of small faces, i.e., for all inner faces f, $3 \leq |f| \leq k + 1$. The next simplification deals with the special case of faces that are arranged linearly:

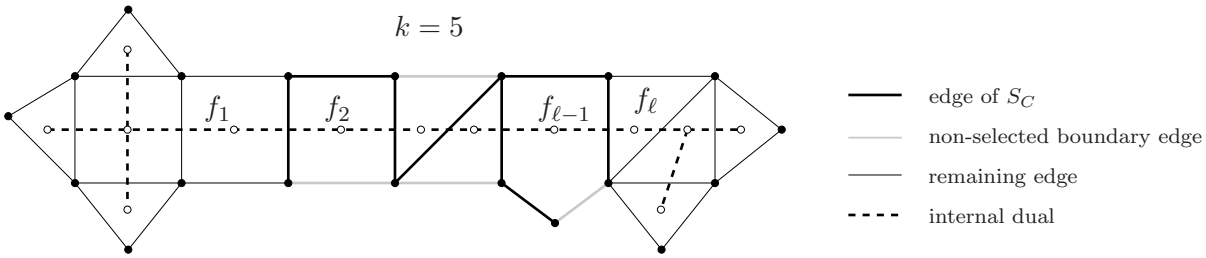

Figure 3.8: Chain $C = \{f_1, \ldots, f_\ell\}$ and a minimum 5–spanner S_C as constructed by Procedure CHAIN-SPANNER(C).

Definition 3.10 (chain)
Let T be the internal dual of G. A subgraph C of G is a chain *of G if*

- *C is biconnected and contains at least two inner faces;*
- *all nodes of T that correspond to faces of C have degree at most 2 in T;*
- *C is maximal with respect to this property.*

We denote C by $\{f_1, \ldots, f_\ell\}$, where f_i, for $1 \leq i \leq \ell$, is an inner face of C.

See Figure 3.8 for an example. Given a chain $C = \{f_1, \ldots, f_\ell\}$, we construct a minimum k–spanner S_C for the internal faces $\{f_2, \ldots, f_{\ell-1}\}$ of C as follows:

Procedure CHAIN-SPANNER(C)
Input: A chain $C = \{f_1, \ldots, f_\ell\}$ consisting of small faces.
Output: A minimum k–spanner S_C for $\{f_2, \ldots, f_{\ell-1}\}$.
Algorithm:

1. Put all chords of $\{f_2, \ldots, f_{\ell-1}\}$ (including those between f_1, f_2 and $f_{\ell-1}, f_\ell$) into S_C.
2. For all $f_2, \ldots, f_{\ell-1}$, put all but one bounding edges into S_C.

Lemma 3.11 *S_C as constructed by* CHAIN-SPANNER(C) *is a minimum k–spanner of $\{f_2, \ldots, f_{\ell-1}\}$.*

Proof By assumption, all faces in $\{f_2, \ldots, f_{\ell-1}\}$ are small. Thus, each boundary edge that is not in S_C is covered by a path along its incident face. Since S_C is a tree, it is also a minimum k–spanner of $\{f_2, \ldots, f_{\ell-1}\}$. □

If G is a chain, it remains to treat the faces f_1 and f_ℓ, which is trivial since both faces are small. Otherwise we use Chain-Spanner to obtain smaller graphs by cutting G in C as follows: We set $G' = G - \{f_2, \dots, f_{\ell-1}\}$, i.e., we cut G along the chords between f_1 and f_2 as well as between $f_{\ell-1}$ and f_ℓ. By this, G' consists of two components G'_1 and G'_2, which are both biconnected.

A minimum k–spanner of G can then be constructed from minimum k–spanners of $\{f_2, \dots, f_{\ell-1}\}$, G'_1 and G'_2 as indicated in the following lemma:

Lemma 3.12 *Let $C = \{f_1, \dots, f_\ell\}$ be a chain in G, and S_C be a minimum k–spanner of $\{f_2, \dots, f_{\ell-1}\}$ as constructed by* Chain-Spanner(C). *Let G'_1 and G'_2 be the components of G' that arise by cutting G in C, and let S'_1 and S'_2 be minimum k–spanners of G'_1 and G'_2, respectively, such that S'_1 contains the chord between f_1 and f_2 and S'_2 contains the chord between $f_{\ell-1}$ and f_ℓ. Then, the subgraph S obtained by recombining S_C, S'_1, and S'_2 according to the cut in C is a minimum k–spanner of G.*

Proof It is clear that the subgraph described in the lemma is a k–spanner of G. To see the optimality, first observe that S_C is optimal for $\{f_2, \dots, f_{\ell-1}\}$ and that it contains in particular the chord between f_1 and f_2 as well as the chord between $f_{\ell-1}$ and f_ℓ. This implies that the choice of edges for S_C within $f_2, \dots, f_{\ell-1}$ does not affect the choice of edges that are necessary to cover edges in f_1, f_ℓ, or $G - C$, since the vertices of $f_2, \dots, f_{\ell-1}$ are only incident to edges within C.

W.l.o.g, consider G'_1 and S'_1. f_1 is a face in G'_1, and the chord between f_1 and f_2 is now a boundary edge. This edge is contained in S_C, and it is preselected in S'_1. We have to show that this preselection does not interfere with the choice of optimal edges within G'_1. This can be seen as follows.

By Lemma 3.8, we know that in every k–spanner of G'_1, and in particular in a minimum k–spanner, at most one boundary edge of f_1 may be missing. Since f_1 belongs to the chain C and $|f_1| \geq 3$, we know that f_1 contains at least one boundary edge w.r.t. G. This boundary edge is not included in S_C, and remains a boundary edge w.r.t. G'_1. This edge may thus be chosen to be missing in a minimum k–spanner. □

Observe that the case where $\ell = 2$ is included in the above argumentation. Thus in the following, we can assume that G consists of small faces and does not contain any chain.

Small Faces without Chains

To deal with the general case, we make use of a *weighted* version of the *internal dual* T, where in each node of T the size $|f|$ of the corresponding

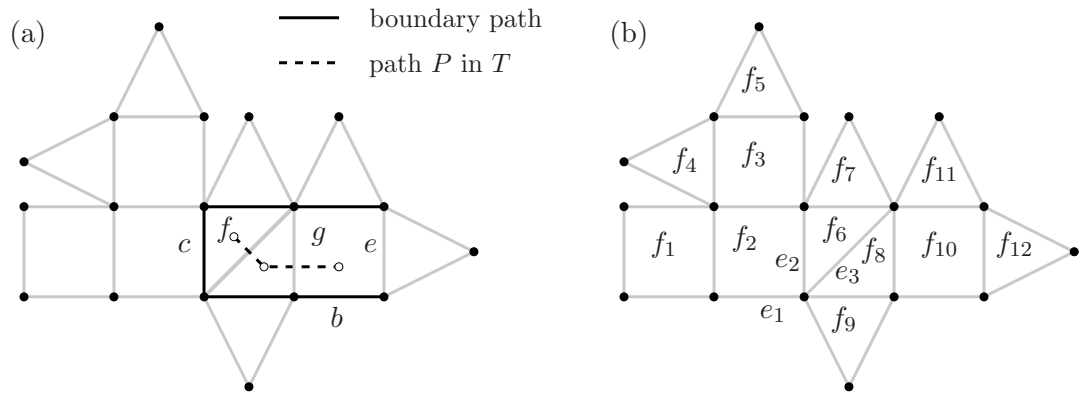

Figure 3.9: (a) Boundary path for e up to f, (b) example for influence regions, e.g., $f_{10} \in IR(f_6)$ for $k = 5$.

face f of G, i.e., the number of incident edges, is stored. In the sequel, T denotes the weighted internal dual. Observe that every edge of T corresponds to a chord in G. Furthermore, every face f that corresponds to a leaf in T contains at least two boundary edges. T can be constructed in linear time. In the following, we consider G together with its weighted internal dual T, denoted by (G, T).

Throughout this section, we often use paths that follow the boundary of adjacent faces. The *boundary path* is defined as follows: Let $e = \{u, v\}$ be an edge in G and f be a face of G. Moreover, let g be the face of G that is incident to e and that is closer to the face f in T (note that g is unique since T is a tree). Denote by P the path from f to g in T. The *boundary path* for e up to f is the path from u to v that follows the boundary of all faces of P. See Figure 3.9(a) for an illustration.

Influence regions. The key observation for the construction of a minimum k–spanner is as follows: The influence of a chord with respect to covering other edges is bigger than that of a boundary edge. We quantify this by the following definition:

Definition 3.13 (influence region)

1. *The* influence region of an edge e, $IR(e)$, *is the set of all faces f of G for which the boundary path for e up to f has length at most k.*

2. *The* influence region of a face g *is the set of all faces f of G that fulfill Part* 1 *for every edge incident to g, i.e.,* $IR(g) = \bigcap_{e \text{ incident to } g} IR(e)$.

Thus, the boundary paths up to the faces of the influence region of an edge are covering paths of that edge. In Figure 3.9(b) we set $k = 5$, and we have

$$\begin{aligned} IR(e_1) &= \{f_1, f_2, f_3, f_6, f_7, f_8\} \\ IR(e_2) &= \{f_1, f_2, f_3, f_6, f_7, f_8, f_9, f_{10}\} \\ IR(e_3) &= \{f_2, f_6, f_7, f_8, f_9, f_{10}, f_{11}, f_{12}\}, \text{ and} \\ IR(f_6) &= \{f_2, f_6, f_7, f_8, f_9, f_{10}\}. \end{aligned}$$

As G consists only of small faces, every face belongs to the influence region of itself. Furthermore, influence regions are symmetric, i.e., $g \in IR(f)$ if and only if $f \in IR(g)$. Consider the influence regions of a chord c and a boundary edge b of one particular face. It is easy to see that $IR(b) \subseteq IR(c)$. Moreover, the influence region of a face f only depends on the size of the faces that are close to f with respect to weighted distances in T:

Lemma 3.14 *Let f, g be two inner faces of G and P be the path from f to g in T. Then, $f \in IR(g)$ if and only if* $1 + \sum_{h \in P} (|h| - 2) \leq k$.

Proof We count all but two edges from each face on P and one further edge from f. This is the length of a boundary path of an edge incident to g up to f. □

Note that all edges that are incident to faces of P either belong to this boundary or are covered by a path of length at most k along the boundary. We can use this to construct a tree k–spanner within a given influence region:

Lemma 3.15 *Let f be a face in G. Then, the subgraph G_f of G that consists of the faces of the influence region $IR(f)$ of f admits a tree k–spanner.*

Proof We consider T_f, the subgraph of T consisting of the nodes of $IR(f)$, rooted at f. Furthermore, we classify the edges of G_f into chords and boundary edges with respect to G_f; observe that an edge that is a chord in G may be a boundary edge in G_f. Let g be a leaf in T_f and P be the path in T_f from g to f. Since g is a leaf, it contains at least two boundary edges. We construct a tree k–spanner S_f for (G_f, T_f) as follows:

- Take the boundary path of a boundary edge of g up to f into S_f.
- By Step 1, S_f contains the following edges: all boundary edges of the faces on P but g (e.g., b in Figure 3.9(a)), and also those chords that are incident to one face from P and one face outside P (e.g., c in

Figure 3.9(a)). Consider such a chord c, and let f_c be the face that is not in P and that is incident to c. Denote by T_c the subtree of T_f rooted at f_c. We continue the construction as in Step 1, but now starting with T_c rooted at f_c, and repeat this for all choices of c.

By construction, S_f is a k–spanner and a tree. □

By the previous lemma, we know that we can treat a certain connected portion (i.e., a set of faces) of G by constructing a tree k–spanner, which is clearly also minimum for this portion.

Overall strategy. We use this to construct a minimum k–spanner of G: We search for a suitable portion, construct a minimum k–spanner for it, cut it off, and repeat this process with the remainder of G. In the end, we put together all the k–spanners of these portions to form a subgraph S of G. It then remains to show that S is a minimum k–spanner of G.

The selection of the actual portion in each iteration step is dominated by the following requirements:

- The remainder of G after cutting off the portion is connected;
- the portion admits a tree k–spanner;
- the portion is maximal to that respect.

A suitable strategy for a selection of the portion is to proceed from 'outside inwards': starting with a leaf in T and proceeding towards a central node (to be defined below). For this, we consider T rooted at the central node, and the selected portions correspond to rooted subtrees of T.

Let us now discuss the notion of centrality, which is inevitable in this process. We use again weighted distances in T: The central node is a node that has the smallest weighted distance to all leaves.

Definition 3.16 (central node)
Let T be a node weighted tree, where $|h|$ denotes the weight of a node h. A node r of T is a central node *of T if the weighted maximal distance from r to all leaves of T is minimal; i.e., select r such that the following is minimal:*

$$\max_{\ell \text{ leaf in } T} \left\{ \sum_{h \text{ node on } P} |h| \right\}, \text{ where } P \text{ is the path from } r \text{ to } \ell \text{ in } T.$$

Observe that, by definition of T, a central node of T is never a leaf (w.l.o.g., assume that T contains at least three nodes).

Selection of the considered portion. We now describe the process of finding the actually considered portion of G in more detail. Suppose we are given T together with the central node r and we consider T rooted at r. A *face of maximal influence*, is a face f such that

- all leaves of the subtree T_f rooted at f are contained in the influence region of f, and
- f is chosen maximally, i.e., either $f = r$ or there is a leaf $\ell \in T_f$ such that $\ell \in IR(f)$ but $\ell \notin IR(g)$, where g is the direct ancestor of f in T. In this case, we say that ℓ *determines* f.

Observe the following: If f is a face of maximal influence then the graph corresponding to T_f admits a tree k–spanner since all faces of T_f belong to the influence region of f. Furthermore, if we cut off T_f, the remaining graph remains connected.

The algorithm. We use this definition for the choice of the actual portion by choosing a face of maximal influence that is furthest away from r. We proceed as follows:

Procedure SMALL-FACES-SPANNER(G, T)

Input: A graph G consisting of small faces without chains together with its weighted internal dual T.

Output: A minimum k–spanner for G.

Algorithm:

1. Compute a central node r of T.
2. Search for a face f of maximal influence such that f has a longest distance from r. Construct a tree k–spanner similar as in Lemma 3.15 starting from a leaf that determines f.
3. Remove the faces corresponding to T_f from (G, T), and repeat.

Note that we select not an arbitrary, but a particular face of maximal influence in Step 2. As we will see later, this choice is necessary to retain the optimality of the constructed k–spanner.

Furthermore, in Step 3, we remove exactly the faces of T_f. The chord that separates f from $T - T_f$ becomes a boundary edge in $T - T_f$. By the choice of the edges for the k–spanner of T_f in Step 2, this edge is preselected to be in the k–spanner of $T - T_f$ too. Thus some more care has to be taken

in Step 3 before continuing the process. We will flesh this out in the proof of the correctness lemma (Lemma 3.17) below.

Using the definition of centrality, we get the following correctness lemma:

Lemma 3.17 *Procedure* SMALL-FACES-SPANNER(G, T) *yields a minimum k–spanner for G.*

Proof Since at each step at least one face is removed, the procedure terminates. Moreover, it is easy to see that the produced subgraph S is a k–spanner of G. It remains to show that S is minimum. To see this, observe the following properties:

1. The procedure exploits the influence regions in a best possible way and constructs tree subgraphs whenever this is possible. Within these regions, those subgraphs are thus minimal.

2. The search for f never goes beyond r. That means, if all leaves of T are in the influence region of r then we choose $f = r$ in Step 2. It is clear that, in this case, G admits a tree k–spanner, and S as constructed in SMALL-FACES-SPANNER is a tree.

3. Suppose that there is a leaf ℓ in T such that ℓ is not in the influence region of r. In Step 2, it is guaranteed that a face $f \neq r$ is chosen. In this case, G may or may not admit a tree k–spanner.

4. The chord between f and its ancestor in T is in S. This edge can be used also to cover edges of faces beyond T_f.

Consider a particular iteration step of SMALL-FACES-SPANNER, where f is chosen as a face of maximal influence. Denote the ancestor of f by g. Let G' be the graph that remains after the removal of T_f in Step 3, and let T' be the weighted internal dual for G'.

The removal of the faces is carried out along an edge that is a chord in G and that becomes a boundary edge in G'. According to the construction in the actual iteration step, this boundary edge is preselected to be in S. As in the proof of Lemma 3.12, we have to make sure that this preselection does not destroy the optimality of the k–spanner which is constructed for G'. We do this by showing that we do not close an unnecessary cycle in S, but retain a tree substructure whenever this is possible.

The situation is easy if g contains a boundary edge that has not been selected yet; see Figure 3.10(a). Here, we can use Lemma 3.8 and choose this edge as the one missing edge.

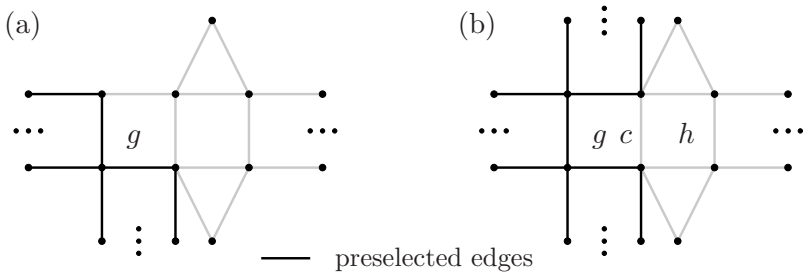

Figure 3.10: Removal of faces such that the new leaf g (a) contains a non-selected boundary edge, (b) contains only preselected boundary edges.

Now consider the case, where all boundary edges of g are preselected, see Figure 3.10(b). By construction, all these edges are needed to cover the removed faces in an optimal way, i.e., such that they are in a minimum k–spanner of the actual portion that corresponds to the removed subtree, and such that they can also be used for covering edges beyond that region (cf. Observation 4 above). Otherwise the previous step would have considered g too and would have removed also g.

Here, some extra action has to be taken in Step 3: Let h be the unique ancestor of g in G' and let c be the chord between g and h. Then, c is already covered in S by the path along the preselected boundary edges (because g is a small face). On the other hand, by the choice of the faces of maximal influence in the previous steps, c cannot be used to cover edges in the removed part of G. Furthermore, c does not help for covering edges in G' since the influence region of c within G' is contained in the influence region of any other edge of h. Thus, it is possible to remove c from G' and to melt together g and h to form a new, bigger face f'. If f' remains small (i.e., $|f'| \leq k+1$) we can proceed as before. Otherwise, f' becomes a big face, and we can use Procedure Split-Big-Face(f') and Lemma 3.9. Clearly, this may result in a cycle in S. But it is easy to see that this cycle is inevitable to cover all edges. □

Together with the arguments of the previous subsections and Lemmas 3.9 and 3.12, we get an algorithm to construct a minimum k–spanner in outerplanar graphs:

Theorem 3.18 Minimum k–Spanner *can be solved in linear time (and thus is in $\mathcal{P}$) for outerplanar graphs.*

Proof The algorithm basically consists of the three components Split-Big-Face, Chain-Spanner, and Small-Faces-Spanner. It remains to

show the time complexity of the algorithm given above. Given G, its weighted internal dual T can be constructed in linear time. Moreover, big faces and chains can be detected and treated in linear time.

Now consider SMALL-FACES-SPANNER(G, T). First observe that the weighted diameter of an influence region, and thus the distance from a leaf ℓ to its face of maximal influence f is bounded by $\mathcal{O}(k) = \mathcal{O}(1)$. Thus, given ℓ and f, the actual construction of the spanner can then be done in constant time. The critical operations that remain are the determination of the central node r and the actual choice of f.

In the beginning, a central node r of T and the faces of maximal influence for all leaves can be computed in linear time by starting from the leaves, labeling them, and successively deleting them (see, e.g., Brandstädt (1994), p. 28, for details on a similar problem). During the iteration, it is not necessary to compute a new central node in each step from scratch: As a second consequence of the bounded distance between ℓ and f, r can shift within T at most k faces. Thus the necessary updates for r can be done in constant time.

For the computation of the faces of maximal influence in each step, we can proceed as follows: In the beginning, start with all leaves in parallel and search for a face of maximal influence bottom-up, i.e., until the first such face is found. This is the face f that is considered in the actual iteration. After cutting off f in Step 3, the search for the next face of maximal influence can make use of the information obtained so far: only the direct successors of the ancestor g of f have to be reconsidered. But since G contains only small faces, the degree of g is bounded by a constant. Together with the argument of bounded diameter of influence regions as above, this re-computation can be done in constant time.

It remains to consider the case where a big face occurs. Let $G_0, G_1, \ldots, G_\ell$ be the components that arise from the call of SPLIT-BIG-FACE, and let G_0 be the component containing the central node. Since f has been chosen to be a face of maximal influence that has longest distance from r, the weighted diameters of $G_1, \ldots, G_\ell$ are also bounded by k. By this, the new central node for G_0 can also be recomputed in constant time. Altogether this results in a linear time algorithm. □

Observe that the arguments used in this section heavily rely on the implicit structure given by the outerplanarity. Hence this approach cannot be used for example for general ℓ–outerplanar or planar graphs.

3.4 Minimum Planar k–Spanners

Instead of restricting the input graph, we now consider restrictions on the spanning subgraph by introducing planar k–spanners.

Definition 3.19 (planar k–spanner)
For any rational $k \geq 1$, a spanning subgraph $S = (V, E')$ is a planar k–spanner *of a graph $G = (V, E)$, if S is a k–spanner, and S is planar.*

The remarks on integer stretch factors and the restriction of biconnected components apply as in the case of general k–spanners. Reconsider Figure 1.1(a), where a planar 2–spanner of a non-planar graph is given. As in the case of tree k–spanners, there are graphs that, for a fixed stretch factor k, do not admit a planar k–spanner. As mentioned before, $K_{3,3}$ does not admit a tree 2–spanner, but it neither admits a planar 2–spanner. On the other hand, C_4 does not admit a tree, but a planar 2–spanner. Clearly, a graph that admits a tree k–spanner also admits a planar k–spanner, but the reverse is not true.

A planar k–spanner contains at most $3|V| - 6$ edges and is therefore sparse. But, as opposed to the case of tree k–spanners, the number of edges may vary. We therefore consider minimum planar k–spanners and discuss the corresponding optimization problem (cf. Problem 2.4):

Problem 3.20 Minimum Planar k–Spanner

Given: A graph G, positive integer K.

Problem: Does G contain a planar k–spanner having at most K edges?

First, consider the *unweighted* case: As discussed in Section 2.1, the only 1–spanner of G is G itself. Using a linear time planarity test, Minimum Planar 1–Spanner is thus in $\mathcal{P}$. On the other hand, as stated in Section 2.2, it is $\mathcal{NP}$-complete to decide whether an unweighted graph contains a tree k–spanner if $k \geq 4$ (Cai and Corneil (1995)). Since spanning trees are planar with the least possible number of edges, the $\mathcal{NP}$-completeness for Minimum Planar k–Spanner for $k \geq 4$ follows immediately.

For *weighted* graphs the situation is different: As mentioned in Section 2.1, the unique 1–spanner with a minimal number of edges is also the unique minimum 1–spanner with respect to the total edge weight, and can be determined in polynomial time. Since we assume that all edge weights are positive, a minimum 1–spanner has a minimal number of edges. Therefore, if a planar 1–spanner exists, then the minimum planar 1–spanner is identical to the minimum 1–spanner, and we can conclude that Minimum Planar 1–Spanner is in $\mathcal{P}$ for weighted graphs by testing the minimum 1–spanner for planarity.

In Cai and Corneil (1995), the $\mathcal{NP}$-completeness of TREE k–SPANNER for $k > 1$ in weighted graphs is proved. By a close look at the transformation used there, and by an appropriate choice of the bound on the total weight of a planar k–spanner, the proof can be modified to show the $\mathcal{NP}$-completeness of MINIMUM PLANAR k–SPANNER for $k > 1$ in weighted, undirected graphs. Altogether, we get the following corollary:

Corollary 3.21

1. *For any fixed rational number $k \geq 4$,* MINIMUM PLANAR k–SPANNER *is $\mathcal{NP}$-complete for unweighted graphs.*

2. *For any fixed rational number $k > 1$,* MINIMUM PLANAR k–SPANNER *is $\mathcal{NP}$-complete for weighted graphs.*

Observe that MINIMUM k–SPANNER and MINIMUM PLANAR k–SPANNER are the same for planar instances.

3.5 Further Remarks

In this chapter, we have examined the problem of finding minimum k–spanners (MINIMUM k–SPANNER) in planar and outerplanar graphs, and the problem of finding minimum planar k–spanners (MINIMUM PLANAR k–SPANNER) in general graphs. Apart from some trivial cases and a restriction to ℓ–outerplanar graphs for a fixed ℓ, both problems have been shown to be $\mathcal{NP}$-complete for most stretch factors k.

It remains to fill the open cases, i.e., to establish the complexity status of

- MINIMUM k–SPANNER for unweighted, planar graphs for $k \in \{2, 3, 4\}$;
- MINIMUM k–SPANNER for weighted, planar graphs for $1 < k < 3$;
- MINIMUM PLANAR k–SPANNER for unweighted graphs for $k \in \{2, 3\}$.

Furthermore, the complexity status of the decision problem for planar k–spanner is still open:

Problem 3.22 PLANAR k–SPANNER

Given: A graph G.

Problem: Does G admit a planar k–spanner?

Related Work

As stated in Section 2.1, Kortsarz (1998) has shown that, for unweighted graphs and every $k \geq 2$, MINIMUM k–SPANNER is hard to approximate within logarithmic ratio. The construction used there uses highly non-planar graph components such that the approach does not apply to the case of planar instances. In fact, the approximability status of MINIMUM k–SPANNER is still open, when restricted to planar graphs.

An upper bound on the approximability for the MINIMUM 2–SPANNER on planar graphs is given by the approximation algorithm of Kortsarz and Peleg (1992) for general undirected, unweighted graphs: this algorithm results in a constant approximation for planar graphs.

Very recently, Fekete and Kremer (1998) have used a proof that is partially based on our approach in Section 3.2, to show that it is $\mathcal{NP}$-hard to find a minimum k such that a given *planar* graph admits a tree k–spanner.

ℓ–Outerplanar Graphs Revisited

A further motivation for studying ℓ–*outerplanar* graphs results from the work of Baker (1994). In this article, Baker examines some optimization problems that are $\mathcal{NP}$-complete on planar graphs. In particular, she develops a concept how exact algorithms that work for ℓ–*outerplanar* graphs can be used to derive a polynomial approximation scheme for general *planar* graphs. By this, a polynomial algorithm is achieved such that the result is as close to the optimal as desired. Since we cannot hope for finding an efficient exact algorithm (unless $\mathcal{P} = \mathcal{NP}$), the given algorithms are optimal with respect to the quality of the result.

Unfortunately, it turns out that this approach cannot be used in the case of MINIMUM k–SPANNER. For Baker's approach to work, the considered problem has to exhibit strong local properties with respect to the deletion or insertion of vertices. This is not the case for MINIMUM k–SPANNER: whenever a vertex and some incident edges are added to a given graph, this may affect the whole structure of a k–spanner.

Chapter 4

Fault-Tolerant Tree Spanners

In the main part of the previous chapter, we have studied the problem of finding sparse k–spanners regardless of the actual structure of the k–spanner. Only in the last part, we have restricted ourselves to planar k–spanners. Now we turn back to the even more rigid concept of tree k–spanners. In particular, we consider a model for incorporating fault-tolerance, that is still maintaining distance guarantees, sparseness and structural simplicity.

4.1 Fault-Tolerant Spanning Trees with Distance Guarantees

Let us reconsider tree k–spanners. As mentioned before (see Section 2.2), if they exist, then these are the sparsest k–spanners. Furthermore, the tree property is of particular interest in several applications. One major drawback of tree subnetworks, however, is their vulnerability: A fault of one single link or site results in disconnecting the subnetwork.

In order to overcome this problem, we now introduce models of fault-tolerance without destroying the simple structure and sparseness too much. In particular, we discuss three topics which are all based on the concept of *independent trees*, i.e., pairs of trees in which the two unique paths from any vertex to a specified root vertex are edge disjoint or vertex disjoint, respectively. By this it is guaranteed, that even in the case of the fault of one edge or vertex, one tree path to the root remains. In all three topics, we examine the *edge* as well as the *vertex* disjoint case, thus guaranteeing fault-tolerance in either one link or one site.

A Model of Fault-Tolerance

As a first model, in Section 4.2, we impose the concept of independent trees directly on the concept of tree k–spanners, and consider *independent tree k–spanners*: pairs of tree k–spanners that are independent. We can thus guarantee fault-tolerance such that the distance is bounded by a constant factor for any pair of vertices, even if either one edge or one vertex fault occurs. Moreover, the subgraph consisting of the union of two independent tree k–spanners is also sparse and has good structural properties. Accordingly, this sort of subgraphs may serve as a concept for more reliable, but simply structured networks that still guarantee short delays.

In this chapter, we give a constructive, efficient characterization for graphs admitting a pair of independent tree k–spanners for $k \leq 2$. For $k \geq 4$, however, the corresponding decision problem is $\mathcal{NP}$-complete. Both results hold for the edge and vertex disjoint case.

A Relaxed Distance Constraint

Unfortunately, the concept of independent tree k–spanners turns out to be very restrictive in the sense that the class of graphs admitting a pair of independent tree k–spanners for a fixed k and a given root is quite small. In many settings in the design of subnetworks, however, it is not strictly necessary that the distance guarantee holds for all pairs of vertices, but only for the distances of all vertices from the specified root vertex. This leads in Section 4.3 to the definition of *k–root-spanners*, and in particular *independent tree k–root-spanners*, where we are interested in finding a pair of independent spanning trees such that the distance between any vertex and the root in every tree is at most k times the distance in the graph. Here, we show that the problem of deciding the existence of a pair of independent tree k–root-spanners is $\mathcal{NP}$-complete for all non-trivial values of k.

Direct Root Spanners

The general problem of deciding the existence of independent tree k–root-spanners is hard for both cases of edge and vertex faults. In Section 4.4, we consider a natural restriction of the problem: We examine independent tree k–root-spanners where paths in the root-spanner are not allowed to 'run uphill'. This means that, in such a root-spanner, on the path from a vertex to the root, we either go one step closer to the root or we stay at the same distance with respect to the original graph. We call these root-spanners *direct*. It turns out that, in this case, the situation for the edge and vertex

disjoint case differs significantly. The decision problem for such edge independent tree k–root-spanners can be solved in linear time for all k, whereas the corresponding problem for such vertex independent tree k–root-spanners remains $\mathcal{NP}$-complete.

Some Notation

Before going into details, we need some more notation. The *root* of a graph $G = (V, E)$ is a specified vertex $r \in V$. Given a root r, we partition V into *levels* $L_\ell := \{v \in V |\ d_G(v, r) = \ell\}$ for $0 \leq \ell \leq \max_v\{d_G(v, r)\}$. The *level index* of a vertex v is indicated by $l(v) := d_G(v, r)$. The level $L_{l(v)-1}$ is called *parent level* of vertex v. A vertex in $L_{l(v)-1} \cap N(v)$ is called a *parent vertex* of v, and a vertex in $L_{l(v)} \cap N(v)$ is called a *sibling vertex* of v. Note that every vertex $v \neq r$ has at least one parent vertex.

For a tree T rooted at r, denote the unique path from a vertex v to the root r (called *root path*) by $rp(v, T)$. Sometimes we abuse the notation $rp(v, T)$ to indicate the set of internal vertices of the root path.

A vertex v of a graph G is *universal* w.r.t. $V(G)$ (or *universal* for G) if is is adjacent to all other vertices of G. A *star centered at a vertex* v is a complete bipartite graph $K_{1,m}$, where v is the unique universal vertex.

4.2 Independent Tree k–Spanners

We now deal with a fault-tolerant concept of spanning trees with constant distance guarantees by examining *independent tree k–spanners*. After the formal definition and statement of the results, the proofs for the main theorem are given in the subsequent subsections.

4.2.1 Definitions and Results

Definitions

In the sequel, assume that $G = (V, E)$ is given together with a root $r \in V$.

Definition 4.1 (independent trees)
Two spanning trees T_1 and T_2 of a graph G, both rooted at r, are called edge independent *(or* vertex independent*, respectively)* w.r.t. r*, if for every vertex $v \in V(G)$ the unique root paths $rp(v, T_1)$ and $rp(v, T_2)$ are edge disjoint (or internally vertex disjoint, respectively).*

If the context is clear, in the sequel we often omit the term 'w.r.t. r'. Moreover, in most cases considered here, the situation for vertex independence and

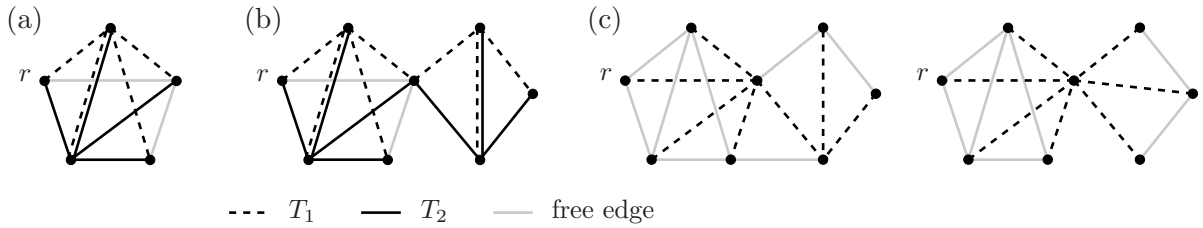

Figure 4.1: (a) A graph admitting a pair of vertex (and edge) independent tree 2–spanners, (b) a graph admitting a pair of edge (but not vertex) independent tree 2–spanners, and (c) two graphs admitting one tree 2–spanner, but no pair of independent tree 2–spanners.

edge independence is similar. Thus in the following, if not stated explicitly, the terms *disjointness* and *independence* always stand for both cases.

There is a strong relationship, though not equivalence, between the connectivity of a graph and the existence of independent trees. As shown in Itai and Rodeh (1988), every biconnected (or 2–edge-connected, respectively) graph admits a pair of vertex (or edge, respectively) independent trees w.r.t. any r. On the other hand, a graph admitting a pair of *edge* independent trees w.r.t. some r is certainly 2–edge-connected. But note that a graph admitting a pair of *vertex* independent trees w.r.t. some r does not necessarily have to be biconnected: r may be an articulation vertex (but no other vertex than r).

Observe that independent trees do not necessarily have to be edge disjoint. The two trees may share an edge as long as the root paths are disjoint. See for an example Figure 4.1(b).

In the rest of this section, we are interested in finding a pair of tree k–spanners that are independent:

Problem 4.2 Independent Tree k–Spanners

Given: A graph G and a root r.

Problem: Does G admit a pair of tree k–spanners that are independent w.r.t. r?

Figure 4.1 shows examples of graphs with different behavior concerning admissibility of independent tree 2–spanners. Observe that the number of edges of a pair of independent tree k–spanners cannot exceed $2 \cdot |V| - 2$. Clearly, for a fixed k, the class of graphs admitting a pair of *vertex* independent tree k–spanners is a proper subclass of the class of graphs admitting a pair of *edge* independent tree k–spanners, which is itself a proper subclass of the class of

graphs admitting one tree k–spanner. Note that, according to Remark 2.3, we only have to consider integer stretch factors in this section.

Related Concepts

Previous work has mainly concentrated on only one of the two aspects of fault-tolerance and distance guarantee at a time. Itai and Rodeh (1988), for example, deal with independent spanning trees in the context of fault-tolerant communication protocols, but they do not attempt to optimize the length of individual tree paths nor do they consider the resulting stretch factors. Furthermore, independent spanning trees are treated in Khuller and Schieber (1992); Zehavi and Itai (1989); Huck (1994), for example. See also Grötschel et al. (1995) for a survey on fault-tolerant subgraph problems.

In his PhD thesis, Cai (1992) considers *k–tree t–spanners*, i.e., t–spanners that are k–trees (note that here t denotes the stretch factor whereas k denotes the parameter for the construction of the k–tree). Recall that a graph G is a *k–tree* if it is either a k–clique, or there is a vertex v such that $G-\{v\}$ is a k–tree and the neighborhood of v in G induces a k–clique. Since k–trees are k–vertex-connected, a k–tree t–spanner thus guarantees fault-tolerance within the t–spanner. But, in contrast to our concept, the distance conditions may be heavily violated in case of an edge or vertex fault; just the connectivity is maintained.

Very recently, Levcopoulos et al. (1998) examined fault-tolerant spanners in geometric graphs (i.e., complete graphs with Euclidean edge weights). Apart from using this special sort of weighted graphs, their model differs from ours as follows: They do not relate the distances in the spanning subgraph to the distance in the underlying fault-free network, but to the faulty network. Thus in their model, there is no direct measure how much the distances are increasing because of faults of network components. Moreover, Levcopoulos et al. (1998) do not consider the tree structure.

Results

Let us now discuss the concept of independent tree k–spanners in detail. Considering either edge or vertex disjointness, the case for $k = 1$ is trivial: A pair of independent tree 1–spanners cannot exist, since for a vertex v incident to the root r edge $\{v, r\}$ has to be contained in both trees. Thus, the root paths from v are not disjoint. For the other values of k, in the following two subsections, we prove the following theorem:

Theorem 4.3

1. *A pair of independent tree 2–spanners can be found in linear time, if it exists. Thus,* Independent Tree 2–Spanners *is in* $\mathcal{P}$.
2. Independent Tree k–Spanners *is* $\mathcal{NP}$*-complete for* $k \geq 4$.

The case $k = 3$ remains open.

As we will see later, the main property of independent tree 2–spanners (see Lemma 4.5) can be extended to more than two independent tree 2–spanners. Hence, the first part of Theorem 4.3 also holds for ℓ independent tree 2–spanners for $\ell \geq 2$.

4.2.2 Finding Independent Tree 2–Spanners

In this subsection, we prove Part 1 of Theorem 4.3. We first consider the case of an edge fault, and describe the structure of the two edge independent tree 2–spanners. The strict characterization of graphs that admit a pair of *edge* independent tree 2–spanners and the algorithm to find such a pair then follow directly. The case for *vertex* independent tree 2–spanners is treated subsequently.

Edge Independent Tree 2–Spanners

In this subsection we deal with edge independent tree 2–spanners. As mentioned above, it suffices to consider 2–edge-connected graphs. Clearly, if G is 2–edge-connected then it is not necessarily the case that G is also biconnected. G may have articulation vertices and consist of several blocks (i.e., maximal biconnected components). Now observe the following: For every block B of G that does not contain r, there is a unique articulation vertex s such that all paths in G from vertices in B to r have to include s. This vertex s is called *root separator*, and takes over the role of r in B. Let r be the root separator of the blocks containing r. The following lemma describes the form that two edge independent tree 2–spanners may have:

Lemma 4.4 *If* G *admits a pair* T_1 *and* T_2 *of edge independent tree 2–spanners w.r.t.* r *then for every block* B *of* G, $T_1[B]$ *and* $T_2[B]$ *are stars.*

Proof Let T_1 and T_2 be two edge independent tree 2–spanners of G and consider a block B of G with root separator s. From Bondy (1989); Cai and Corneil (1995) we know that for $i \in \{1, 2\}$ either

1. B is 3–vertex-connected and $T_i[B]$ is a star centered at a vertex u_i, or
2. for every 2–separator $\{x, y\}$ of G, edge $\{x, y\}$ is in $T_i[B]$.

In the first case, we are done. Now consider the second case. Let $\{x, y\}$ be a 2–separator, and G_1 and G_2 be the two subgraphs of B that are separated by $\{x, y\}$. Then, by Part 2 above, edge $\{x, y\}$ is in T_1 and in T_2. Let s be the root separator of B and G_1 be the subgraph containing s; see Figure 4.2 for an illustration. Since T_1 and T_2 are edge independent, either $x \in rp(y, T_1)$ and $y \in rp(x, T_2)$ or vice versa.

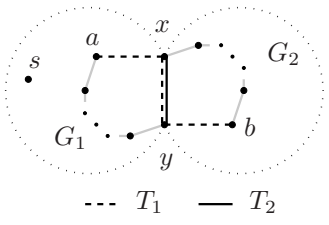

Figure 4.2: Illustration for the second case.

We show that $T_1[B]$ and $T_2[B]$ are stars by contradiction: W.l.o.g., assume that T_1 is not a star, and that the edges $\{a, x\}$ and $\{y, b\}$ belong to T_1. Since B is biconnected and $\{x, y\}$ is a 2–separator, there is a path from a to y in G_1 and a path from b to x in G_2. These paths cannot be covered by two edge independent tree 2–spanners because either the 2–spanner condition or the independence condition are violated; a contradiction to T_1 and T_2 being edge independent tree 2–spanners. □

This observation leads to the following strict characterization:

Lemma 4.5 *G admits a pair of edge independent tree 2–spanners w.r.t. r if and only if every block B of G contains two distinct universal vertices w.r.t. $V(B)$ that are not root separators of B.*

Proof For the if-part, observe that if G fulfills the given conditions then we can construct two edge independent tree 2–spanners by combining the stars centered at the universal vertices, i.e., by taking all edges that are incident to the two universal vertices.

To show the opposite direction, assume that G admits a pair of edge independent tree 2–spanners T_1 and T_2. Then by Lemma 4.4, for every block B, $T_1[B]$ (and $T_2[B]$, respectively) is a star centered at a vertex $u_1^{(B)}$ (and $u_2^{(B)}$, respectively). Since T_1 and T_2 are edge independent, $u_1^{(B)}$ and $u_2^{(B)}$ are disjoint and not root separators of B. □

Using the characterization, an algorithm to decide the existence of a pair of edge independent tree 2–spanners can then be implemented in linear time.

Vertex Independent Tree 2–Spanners

The situation for *vertex* independent tree 2–spanners is similar. We get the following characterization:

Lemma 4.6 *G admits a pair of vertex independent tree 2–spanners w.r.t. r if and only if G admits a pair of edge independent tree 2–spanners w.r.t. r and G is biconnected or 2–edge-connected and r is the only articulation vertex.*

Proof As stated in Section 4.2.1, if G admits a pair of vertex independent trees w.r.t. r then G is at least 2–edge-connected and r is the only articulation vertex. Accordingly, G consists only of blocks containing r. The proof is complete by observing that the edge independent tree 2–spanners found in Lemma 4.5 for blocks containing r are also vertex independent. □

4.2.3 Hardness for $k \geq 4$

In this subsection, we prove the second part of Theorem 4.3, the $\mathcal{NP}$-completeness of Independent Tree k–Spanners for $k \geq 4$. The membership of Independent Tree k–Spanners in $\mathcal{NP}$ is immediate. To show the $\mathcal{NP}$-completeness, we use a reduction from Not–All–Equal 3–Satisfiability (Problem D.12 in Appendix D): Given a set U of Boolean variables, a collection C of clauses over U such that every clause $c \in C$ consists of $|c| = 3$ literals, the problem is to find a truth assignment for U such that every clause in C has at least one true literal and at least one false literal. W.l.o.g., assume that there is no clause that contains one literal in both the positive and negative form.

Given an instance (U, C) of Not–All–Equal 3–Satisfiability, we construct the graph G for Independent Tree k–Spanners as follows:

- For every variable $x \in U$, construct an individual *variable component* (see Figure 4.3(a)):
  - Create a triangle consisting of a *root vertex* r_x and two *literal vertices* x and $\overline{x}$. The edges $\{r_x, x\}$ and $\{r_x, \overline{x}\}$ are called *literal edges*.
  - Create an *auxiliary vertex* w, connect it by an edge each to x and $\overline{x}$. Furthermore connect w to r_x by two simple, disjoint *auxiliary paths* of length $k-1$ that each use new internal vertices $u^{(1)}, \ldots, u^{(k-2)}$, and $v^{(1)}, \ldots, v^{(k-2)}$, respectively. The edges $\{r_x, u^{(1)}\}$ and $\{r_x, v^{(1)}\}$ are called *auxiliary root edges*.
- For every variable $x \in U$, melt together all root vertices r_x to form one distinguished root vertex r.

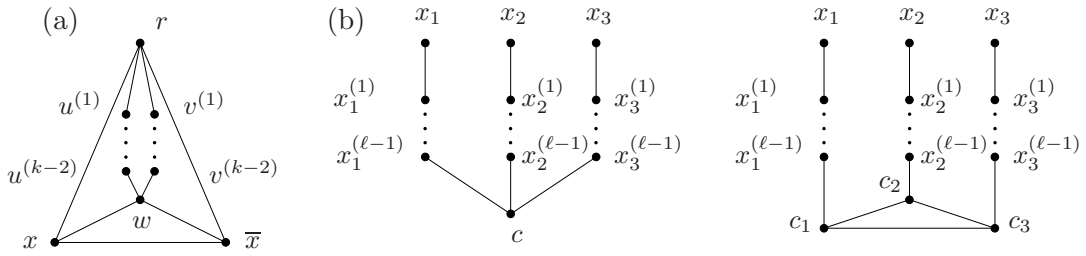

Figure 4.3: (a) A variable component for variable x, and (b) a clause component for even and odd k for the construction of an instance of INDEPENDENT TREE k–SPANNERS.

- For every clause $c = x_1 \vee x_2 \vee x_3 \in C$, construct an individual *clause component* as follows (see Figure 4.3(b)). Let $\ell = \left\lfloor \frac{k}{2} \right\rfloor - 1$.
  - Create *clause vertices* x_1, x_2, and x_3, one for each literal in c.
  - If k is even, then create a *central vertex* c and connect it to all clause vertices by simple, disjoint paths of length ℓ using new internal vertices $x_i^{(1)}, \ldots, x_i^{(\ell-1)}, i \in \{1, 2, 3\}$.
  - If k is odd, then create three *central vertices* c_1, c_2, and c_3, each connected with each other by an edge. Connect x_i with c_i by a simple path of length ℓ using new internal vertices $x_i^{(1)}, \ldots, x_i^{(\ell-1)}$ for $i \in \{1, 2, 3\}$.
- Melt together clause vertices with their corresponding literal vertices.

In Figure 4.4, the part of G for $k = 6$ and clause $c = x \vee y \vee z$ is shown. Clearly, G can be constructed in linear time. It remains to show that there is a not-all-equal truth assignment θ for (U, C) if and only if G admits a pair of independent tree k–spanners w.r.t. r.

From θ to T_1 and T_2. Given a not-all-equal truth assignment θ for (U, C), we can systematically construct two independent tree k–spanners T_1 and T_2. See Figure 4.4 for an example, and let $\theta(x) = \texttt{false}$ and $\theta(y) = \theta(z) = \texttt{true}$. For the lower part of the clause components for odd k confer Figure 4.7.

The construction starts by choosing true literal edges (i.e., literal edges that correspond to true literals) for T_1 and false literal edges for T_2. Within the variable components, it is possible to select edges for T_1 and T_2 according to Figure 4.4, independently of the choice of edges in the clause components.

For every clause component consisting of clause vertices x_1, x_2, and x_3 proceed as follows:

- Choose one true literal[1] x_1 (y in Figure 4.4), and put the path from x_1 to c (or c_1, respectively) into T_1.
- Choose one false literal[1] x_2 (x in Figure 4.4), and put the path from x_2 to c (or c_2, respectively) into T_2.
- Add the path from $x_1^{(1)}$ to c (or to c_2, respectively) to T_2 and the path from $x_2^{(1)}$ to c (or to c_1, respectively) to T_1.
- If $k = 4$, the construction is finished. Otherwise, for the remaining clause vertex, put the path from x_3 to $x_3^{(\ell-1)}$ into the tree to which its literal edge $\{r, x_3\}$ belongs.
  If k is even, put the path from $x_3^{(1)}$ to c into the other tree.
  If k is odd, put the path from $x_3^{(1)}$ to c_3 into the other tree. Put $\{c_1, c_3\}$ into the tree to which $\{r, x_1\}$ belongs and $\{c_2, c_3\}$ into the other tree.

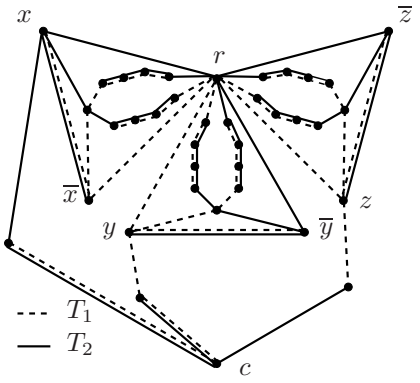

Figure 4.4: Part of an instance of Independent Tree k–Spanners for $k = 6$ and clause $c = x \vee y \vee z$ together with a pair T_1 and T_2 of independent tree k–spanners.

It is easy to see that T_1 and T_2 are independent tree k–spanners of G.

From T_1 and T_2 to θ. The proof of the other direction basically consists of three steps. First, we show in Lemma 4.8 that, given a pair of independent tree k–spanners, these uniquely (modulo choice of T_1 or T_2) induce a truth assignment θ. In Lemma 4.9 we then further examine the structure of two independent tree k–spanners. This result is used in Lemma 4.10 to establish that θ is a not-all-equal truth assignment. The arguments are based on the following fundamental (and straightforward) properties:

Lemma 4.7 *Let T_1 and T_2 be two independent tree k–spanners. Then the following general properties hold:*

[1] This must exist because θ is a not-all-equal truth assignment.

1. *T_1 and T_2 do not contain a cycle.*

2. *For every edge e that does not belong to T_1 (or T_2, respectively), there is exactly one path in T_1 (or T_2, respectively) that connects the end-vertices of e, and this path has length at most k.*

3. *As T_1 and T_2 are independent spanning trees, for every vertex v there are at least two different edges that are incident to v such that one is in T_1 and the other is in T_2.*[2] *In particular, for a vertex v of degree 2, if one incident edge belongs to $rp(v, T_1)$ then the other incident edge has to belong to $rp(v, T_2)$.*

Considering two independent tree k–spanners T_1 and T_2 for a graph G as constructed above, we have the following properties (partly by the previous lemma):

(P1) The literal edges and auxiliary root edges of a variable component cannot be in both T_1 and T_2.

(P2) The auxiliary and internal vertices of a variable component are only connected in G within this variable component.

(P3) The central and internal vertices of a clause component are only connected in G within this clause component.

(P4) Consider a variable component. A path from the root r to a literal vertex via the auxiliary vertex w has length k; the path from r back to r via w has length $2k-2$. Hence, the auxiliary root edges $\{r, u^{(1)}\}$ and $\{r, v^{(1)}\}$ each belong to exactly one of T_1 or T_2, since otherwise the edges of the auxiliary paths cannot be covered.

Suppose that $\{r, u^{(1)}\}$ belongs to T_1. As a consequence of Part 3 of Lemma 4.7, all edges on the path from $u^{(1)}$ to $u^{(k-2)}$ belong to both T_1 and T_2 and the edge $\{u^{(k-2)}, w\}$ belongs to T_2. The same arguments also hold for the other auxiliary path within the variable component, or if T_1 and T_2 are exchanged.

(P5) Consider a clause component. The length of a path from a literal vertex down to a corresponding central vertex and up again to another literal vertex of the same clause is exactly $k-2$.

[2] There can be more than one incident edge in T_1 or T_2 and there may also be edges that are in both trees.

We are now ready to prove the key properties of our construction. First, we establish that, given a pair of independent tree k–spanners, these uniquely (modulo choice of T_1 or T_2) induce a truth assignment θ.

Lemma 4.8 *If T_1 and T_2 are a pair of independent tree k–spanners of G then, for every variable, one literal edge belongs to T_1 and the other to T_2.*

Proof The proof is by contradiction. First, w.l.o.g., suppose that both literal edges of variable x belong to T_2, i.e., $\{r, x\} \in T_2$ and $\{r, \overline{x}\} \in T_2$. By property (P1), $\{r, x\} \notin T_1$ and $\{r, \overline{x}\} \notin T_1$. Now consider the auxiliary root edges $\left\{r, u^{(1)}\right\}$ and $\left\{r, v^{(1)}\right\}$. Because of the length of the paths via w (property (P4)), $\left\{r, u^{(1)}\right\} \in T_1$ and $\left\{r, v^{(1)}\right\} \in T_1$. As neither $\{r, x\}$ nor $\{r, \overline{x}\}$ are in T_1, not only all edges of the path from $u^{(1)}$ to $u^{(t-1)}$ (and from $v^{(1)}$ to $v^{(t-1)}$, respectively) belong to T_1, but also the edges $\left\{w, u^{(t-1)}\right\}$ and $\left\{w, v^{(t-1)}\right\}$. This results in a cycle in T_1, a contradiction.

The same arguments also hold when only one literal edge belongs to one of T_1 or T_2, or that both literal edges do not belong to any of T_1 or T_2. □

Starting from the previous lemma, we can now establish an important property of T_1 and T_2:

Lemma 4.9 *Let T_1 and T_2 be a pair of independent tree k–spanners of G. Then, in both, T_1 and T_2, there is a path of length at most 2 connecting x and $\overline{x}$, and this path does not include r or any vertex outside of the variable component.*

Proof Consider a variable component. By Lemma 4.8, w.l.o.g., assume $\{r, x\} \in T_1$ and $\{r, \overline{x}\} \in T_2$. Then the auxiliary root edges may belong to T_1 or to T_2 independently of each other. To avoid cycles and to guarantee that all internal edges are covered in both trees, we use properties (P2) and (P4) to see that $\{x, w\} \in T_1$ and $\{\overline{x}, w\} \in T_2$.

Observe that if $\{x, \overline{x}\}$ is in T_1 and T_2, or $\{w, \overline{x}\} \in T_1$ and $\{w, x\} \in T_2$, we are done. Suppose that neither $\{x, \overline{x}\}$ nor $\{w, \overline{x}\}$ are in T_1. Thus, both edges have to be covered in T_1. There are three possibilities to cover $\{w, \overline{x}\}$:

- By using edges $\{w, x\}$ and $\{x, r\}$, and returning to $\overline{x}$ via a path through another literal and clause component. By property (P5), we see that such a path has length at least $k - 2 + 3 = k + 1 > k$ (contradiction).

- By following an auxiliary path up to the root and returning to $\overline{x}$. By properties (P4) and (P5), this also results in a path of length strictly greater than k (contradiction).

- By using edge $\{w, x\}$, running down to a central vertex of the own clause component, up to another literal vertex and returning to $\overline{x}$ via a path through another clause component. Again by property (P5), we see that this path has length $2 \cdot (k-2) + 1 = 2k - 3 > k$ for $k \geq 4$ (contradiction).

Hence, either $\{x, \overline{x}\}$ or $\{w, \overline{x}\}$ are in T_1. The case for T_2 is analogous. □

Using the previous lemma, we finish the proof of the second part of Theorem 4.3 by establishing that there is a not-all-equal truth assignment induced by T_1 and T_2.

Lemma 4.10 *Let T_1 and T_2 be a pair of independent tree k–spanners of G. The truth assignment θ that is induced by setting $\theta(x) =$ `true` if and only if $\{r, x\} \in T_1$ is a not-all-equal truth assignment for (U, C).*

Proof By Lemma 4.8, θ is well-defined. It suffices to show that there is no clause $c = x_1 \vee x_2 \vee x_3 \in C$ such that $\theta(x_1) = \theta(x_2) = \theta(x_3)$. This is done by contradiction:

W.l.o.g suppose that there is a subgraph of G corresponding to clause c such that $\{r, x_i\} \in T_1$ and $\{r, \overline{x_i}\} \in T_2$ for all $i \in \{1, 2, 3\}$ (thus inducing an all-equal truth assignment for c). By Lemma 4.9, $|rp(x_i, T_1)| = 1$ and $|rp(x_i, T_2)| \geq 2$.

Now consider edges $\{x_i, x_i^{(1)}\}$ of the corresponding clause component. We show that $\{x_i, x_i^{(1)}\} \in T_2$ for all $i \in \{1, 2, 3\}$ by contradiction: Suppose that $\{x_1, x_1^{(1)}\} \notin T_2$. (The same arguments also hold for x_2 and x_3.) Then there are two possibilities how $\{x_1, x_1^{(1)}\}$ can be covered in T_2:

- By starting from $x_1^{(1)}$, going down to a central vertex of the clause component, following up to another literal vertex (e.g., x_2), further up to r within the variable component and running down again to x_1. This results in a path of length at least $k + 1$ (contradiction).

- By starting from x_1, using another clause component that is incident to x_1 to reach literal vertex x_2 (or x_3, respectively) and returning back via a central vertex of the own clause component to $x_1^{(1)}$. This results in a cycle in T_1 since by Lemma 4.9 both x_1 and x_2 (or x_3, respectively) are connected to r in T_2 (contradiction).

Hence, $\{x_1, x_1^{(1)}\} \in T_2$. The same arguments also hold for x_2 and x_3 and for all other edges of the clause component of c (by property (P3)). This results in a cycle for T_2 and yields a contradiction. □

4.3 Independent Tree k–Root-Spanners

In the previous section, we have added a further condition to the tree and distance condition as imposed by the tree spanner. As we have seen in Part 1 of Theorem 4.3, this concept is very rigid. We now relax on the distance constraint.

Definitions

In many settings, it is sufficient for a subgraph that the distance guarantee just holds for the distances between a vertex and a specified root. In this case, we do not have to care for distances between the other pairs of vertices. This leads to the new concept of *k–root-spanners*:

Definition 4.11 (k–root-spanner)
For any rational $k \geq 1$, a spanning subgraph $S = (V, E')$ is a k–root-spanner *of a graph $G = (V, E)$ w.r.t. r, if*

$$d_S(u, r) \leq k \cdot d_G(u, r) \quad \text{for all } u \in V.$$

A tree k–root-spanner *is a k–root-spanner that is a tree.*

Observe that the shortest path tree is a tree 1–root-spanner and therefore every graph admits a tree k–root-spanner for arbitrary k and r, in contrast to tree k–spanners. Within the context of fault-tolerance, we are now interested in finding a pair of tree k–root-spanners that are independent:

Problem 4.12 Independent Tree k–Root-Spanners

Given: A graph G and a root r.

Problem: Does G admit a pair of independent tree k–root-spanners w.r.t. r?

Figure 4.5 shows examples of graphs (not) admitting two independent tree 2–root-spanners. The vertices of the graphs are drawn according to their levels (as defined in Section 4.1) such that their distances to the root are apparent. In the sequel, we often make use of the these levels.

Related Concepts

Annexstein et al. (1998) have considered related concepts. They consider *radius bounded independent spanning trees*, where they impose an *absolute*, constant upper bound on all vertex-to-root distances. In a second paper, Annexstein et al. (1996) give linear-time algorithms for the construction of

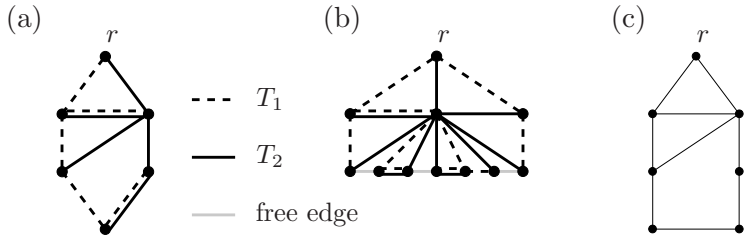

Figure 4.5: (a) A graph admitting a pair of vertex (and edge) independent tree 2–root-spanners, (b) a graph admitting a pair of *edge* (but not *vertex*) independent tree 2–root-spanners, (c) a graph not admitting a pair of independent tree 2–root-spanners.

independent spanning trees with provable upper bounds on the stretch factors. But these stretch factors may well be non-constant. Here, we show that the problem of deciding the existence of independent spanning trees with fixed constant stretch factors is hard by showing its $\mathcal{NP}$-completeness for all non-trivial values of k.

Another concept related to tree k–root-spanners is the *delay-bounded minimum Steiner tree*, where one is interested in finding a Steiner tree [3] T such that T fulfills some given distance constraints especially for the distances from a specified root (see for example Zhu et al. (1995); Rouskas and Baldine (1997); Haberman and Rouskas (1997)). But there an absolute, fixed delay bound is imposed on all vertices.

Results

We now discuss the concept of independent tree k–root-spanners in detail. It is easily seen that, for every fixed k, the class of graphs admitting a pair of *vertex* independent tree k–root-spanners is a proper subclass of the class of graphs admitting a pair of *edge* independent tree k–root-spanners. On the other hand, the class of graphs admitting a pair of independent tree k*–root-spanners* is a superclass of the class of graphs admitting a pair of independent tree k–spanners.

In contrast to k–spanners, it is not possible to just check edges if we want to see whether a given graph is a k–root-spanner: Every vertex has to be considered separately.

Remark 4.13 *As opposed to k–spanners (Remark 2.3), rational stretch factors are relevant for k–root-spanners even in unweighted graphs. Thus in the following, we assume k to be* rational.

[3] see also Minimum Steiner Tree, Problem D.4 in Appendix D

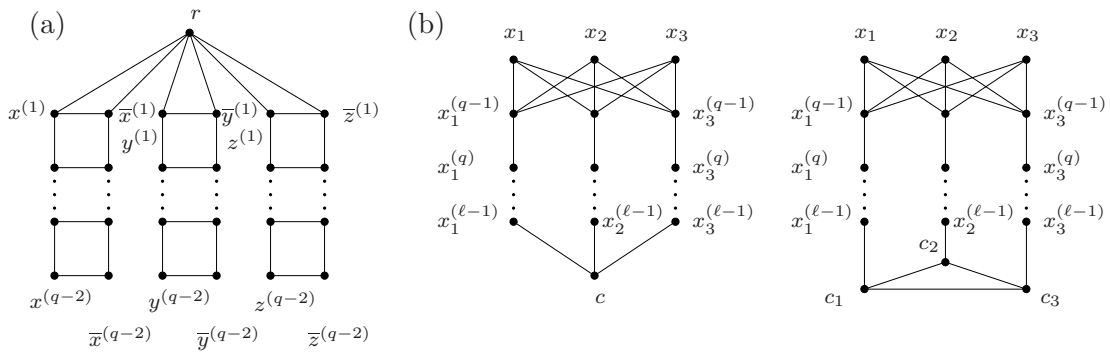

Figure 4.6: (a) The variable components for variables x, y and z, and (b) a clause component for even $p+q$ and odd $p+q$ for the construction of an instance of Independent Tree k–Root-Spanners.

As before, the case for $1 \leq k < 2$ is trivial: A tree k–root-spanner for $1 \leq k < 2$ contains all edges that are incident with the root r, since otherwise the corresponding end-vertex at level 1 would have a distance at least 2 from r. Thus a pair of independent tree k–root-spanners cannot exist for $1 \leq k < 2$. Together with the following theorem, the complexity status of Independent Tree k–Root-Spanners is fully characterized.

Theorem 4.14 Independent Tree k–Root-Spanners *is* $\mathcal{NP}$*-complete for all rational* $k \geq 2$.

Proof As in the previous section, the membership of Independent Tree k–Root-Spanners in $\mathcal{NP}$ is immediate, and we use a reduction from Not–All–Equal 3–Satisfiability. Let $k \geq 2$ be a rational number such that $k = \frac{p}{q}$, where p and q are positive integers with $p \geq 2q$ and $q > \frac{k+3}{2}$ (i.e., for the representation of k, we choose p and q large enough such that the condition is fulfilled; in particular, $q > 2$).

Given an instance (U, C) of Not–All–Equal 3–Satisfiability, we build the graph for Independent Tree k–Root-Spanners as follows:

- Construct a *variable component* (see Figure 4.6(a)):
  - Create a *root vertex* r.
  - For every variable $x \in U$, create $q-2$ layers, each consisting of two vertices $x^{(k)}$ and $\overline{x}^{(k)}$ for $1 \leq k \leq q-2$, where $x^{(k)}$ and $\overline{x}^{(k)}$ are connected by a *horizontal edge*. Connect r with $x^{(1)}$ and with $\overline{x}^{(1)}$ by a *literal edge* each and $x^{(k-1)}$ with $x^{(k)}$ (and $\overline{x}^{(k-1)}$ with $\overline{x}^{(k)}$, respectively) for $2 \leq k \leq q-2$ by a *vertical edge* each. The vertices $x := x^{(q-2)}$ and $\overline{x} := \overline{x}^{(q-2)}$ are called *literal vertices*.

- For every clause $c = x_1 \vee x_2 \vee x_3 \in C$, construct a *clause component* (see Figure 4.6(b)):
  - Create *clause vertices* x_1, x_2, and x_3, one for every literal in c.
  - Create three vertices $x_1^{(q-1)}$, $x_2^{(q-1)}$, and $x_3^{(q-1)}$, and connect each of these to each clause vertex, but to no other vertex (as a $K_{3,3}$).
  - The last part depends on the parity of $p+q$ and is similar to the last part of the clause components of Section 4.2.3. Let $\ell = \lfloor \frac{p+q}{2} \rfloor$.

    If $p+q$ is even, then create a *central vertex* c and connect it to $x_i^{(q-1)}$, by a simple path of length $\ell - q + 1$, using new internal vertices $x_i^{(q)}, \ldots, x_i^{(\ell-1)}$ for $i \in \{1,2,3\}$.

    If $p+q$ is odd, create three *central vertices* c_1, c_2, and c_3, each connected with each other by an edge. Connect c_i to $x_i^{(q-1)}$, by a simple path of length $\ell - q + 1$, using new internal vertices $x_i^{(q)}, \ldots, x_i^{(\ell-1)}$ for $i \in \{1,2,3\}$.
- Identify the clause vertices with their corresponding literal vertices.

Observe that the superscripts of the vertices reflect the distances to r. In Figure 4.7, the part of G for $k = \frac{13}{4}$ and clause $c = \overline{x} \vee y \vee z$ is shown. Again, G can be constructed in polynomial time. It remains to prove that there is a not-all-equal truth assignment θ for (U, C) if and only if G admits a pair of independent tree k–root-spanners w.r.t. r.

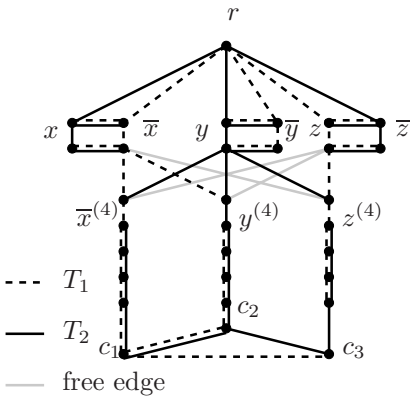

Figure 4.7: The construction of an instance of INDEPENDENT TREE k–ROOT-SPANNERS for $k = \frac{13}{4}$ and clause $c = \overline{x} \vee y \vee z$ together with a pair T_1 and T_2 of independent tree k–root-spanners.

From θ to T_1 and T_2. Given a not-all-equal truth assignment θ for (U, C), we construct two independent tree k–root-spanners T_1 and T_2. See Figure 4.7 for an illustration, and let $\theta(x) = \theta(y) =$ `false` and $\theta(z) =$ `true`. For the lower part of the clause components for even $p+q$ consider again Figure 4.4.

The construction starts by choosing true literal edges (i.e., literal edges that correspond to true literals) for T_1, and false literal edges for T_2. Within the variable components the edges for T_1 and T_2 are chosen independently of the choice of edges in the clause component: Take the vertical edges into the

same tree as the corresponding literal edge and take the horizontal edges into both. For every clause component consisting of clause vertices x_1, x_2, and x_3 proceed as follows:

- Choose one true literal[2] x_1 ($\overline{x}$ in Figure 4.7), and add $\{x_1, x_1^{(q-1)}\}$ to T_1.
- Choose one false literal[2] x_2 (y in Figure 4.7), and add $\{x_2, x_2^{(q-1)}\}$ to T_2.
- Put edge $\{x_3, x_3^{(q-1)}\}$ into the tree to which the corresponding literal edge $\{r, x_3^{(1)}\}$ belongs.
- The remaining part of the clause component is analogous to the construction of Section 4.2.3, starting from vertices $x_i^{(q-1)}$ for $i \in \{1, 2, 3\}$.
- Add three further edges from the $K_{3,3}$ of each clause component to establish the connectivity of the vertices $x_i^{(q-1)}$ for $i \in \{1, 2, 3\}$.

It is easy to see that T_1 and T_2 are independent trees w.r.t. r. Furthermore, T_1 and T_2 obey the distance constraint: All vertices at distance $q-1$ or less are directly connected towards the root in both T_1 and T_2. Consider vertices $x_i^{(q)}$ for $i \in \{1, 2, 3\}$: by construction, for $j \in \{1, 2\}$,

$$|rp(x_i^{(q)}, T_j)| \leq p = k \cdot q = k \cdot d_G(x_i^{(q)}, r).$$

All other vertices below have a longer distance in G to r and hence the resulting paths in the trees may be even longer. Hence the distance constraint is fulfilled for all vertices.

From T_1 and T_2 to θ. Similar to the situation in Section 4.2.3, we first show that a pair of independent tree k–root-spanners uniquely (modulo the choice of T_1 or T_2) induces a truth assignment. As a second step it is shown that this truth assignment is not-all-equal. Observe that, as in Section 4.2.3, Parts 1 and 3 of Lemma 4.7 also hold. If T_1 and T_2 are independent tree k–root-spanners, we now have the following additional properties:

(P1) The literal edges of a variable component cannot be both in T_1 and T_2.

(P2) The internal vertices of a variable component are connected in G only within this variable component.

(P3) The central and internal vertices of a clause component are connected in G only within this clause component.

(P4) Consider a variable component. Any root path from a vertex $x^{(1)}$ (or $\overline{x}^{(1)}$, respectively) that includes a vertex of a clause component (or a part of another variable component) has length at least $2q-3 > k$. Therefore, root paths from $\{r, x^{(1)}\}$ and $\{r, \overline{x}^{(1)}\}$ in T_1 and T_2 can only stay within the variable component. Therefore, either $\{r, x^{(1)}\} \in T_1$ and $\{r, \overline{x}^{(1)}\} \in T_2$, and consequently $|rp(x, T_1)| = |rp(\overline{x}, T_2)| = q-2$ and $|rp(x, T_2)| = |rp(\overline{x}, T_1)| = q-1$, or T_1 and T_2 exchanged.

(P5) Consider a clause component. A path from a vertex $x_i^{(q)}$ down to a corresponding central vertex and up again to a literal vertex has length $p-q+2$.

Suppose we are given two independent tree k–root-spanners T_1 and T_2. By properties (P1), (P2), and (P4), it follows that T_1 and T_2 induce a truth assignment by setting $\theta(x) := \texttt{true}$ if and only if $\{r, x^{(1)}\} \in T_1$. We show that θ is a not-all-equal truth assignment by contradiction:

W.l.o.g., suppose there is a clause $c = x_1 \vee x_2 \vee x_3$ such that $\{r, x_i^{(1)}\} \in T_1$ and thus $\theta(x_i) = \texttt{true}$ for all $i \in \{1,2,3\}$. Consider vertex $x_1^{(q)}$ in the clause component. Since it has distance q from r, its root paths in both T_1 and T_2 has length at most p. By property (P3), $x_1^{(q)}$ has degree 2 and hence there are only two possible edges for a root path starting from $x_1^{(q)}$. In particular, either $\{x_1^{(q-1)}, x_1^{(q)}\} \in rp(x_1^{(q)}, T_1)$ and $\{x_1^{(q)}, x_1^{(q+1)}\} \in rp(x_1^{(q)}, T_2)$, or T_1 and T_2 exchanged. By properties (P5) and (P4), the case $\{x_1^{(q)}, x_1^{(q+1)}\} \in rp(x_1^{(q)}, T_2)$ results in a root path that has length $p+1 > p$, and thus in a contradiction. Hence $\{x_1^{(q-1)}, x_1^{(q)}\} \in rp(x_1^{(q)}, T_2)$. The same argument also holds for x_2 and x_3. This results in a contradiction for $x_2^{(q)}$, which does not have a disjoint root path for T_1.

This completes the proof of Theorem 4.14. □

4.4 Independent Direct Tree *k*–Root-Spanners

As proved in the previous section, it is hard to decide the existence of a pair of general independent tree k–root-spanners in a given graph for all stretch factors at least 2. The situation may change if we impose further restrictions on the root paths. In this section, we consider a case where the results for the edge and vertex disjoint case differ significantly: We can decide the existence of a pair of such edge independent tree k–root-spanners (and its construction) in linear time for all k, whereas the problem of finding such vertex independent tree k–root-spanners remains $\mathcal{NP}$-complete. We first

give the formal definition and then prove the main results in the subsequent subsections.

4.4.1 Definitions and Results

We consider the case where the paths from a vertex to the root within the tree k–root-spanner have to be direct: That means that no deviations to vertices that have a longer distance to the root are allowed. This leads to the following definition:

Definition 4.15 (direct tree k–root-spanner)
For any rational $k \geq 1$, a tree k–root-spanner T of G w.r.t. r is called direct *if for all $v \in V$:*

$$d_G(w, r) \leq d_G(v, r) \quad \text{for every } w \in rp(v, T).$$

Observe that this definition states that a root path either stays within a level or leads to the next level that is closer to r. We are interested in finding a pair of independent direct tree k–root-spanners:

Problem 4.16 Independent Direct Tree k–Root-Spanners

Given: A graph G and a root r.

Problem: Does G admit a pair of independent tree k–root-spanners w.r.t. r, that are both direct?

Paths from a vertex to the root in a *direct* tree k–root-spanner always stay in the same level or lead to a level with smaller index.

For an example consider again Figure 4.5 of the previous section: The graph in (a) admits a pair of independent tree 2–root-spanners, but not two *direct* ones, whereas the 10–wheel in (b) admits a pair of *edge* (but not vertex) independent direct tree 2–root-spanners.

It can be easily seen that, for every fixed k, the class of graphs admitting a pair of *vertex* independent *direct* tree k–root-spanners is a proper subclass of the class of graphs admitting a pair of *edge* independent *direct* tree k–root-spanners, which itself is a proper subclass of the class of graphs admitting a pair of edge independent tree k–root-spanners.

Observe that, as before, independent direct tree k–root-spanners cannot exist for $1 \leq k < 2$ for both the vertex and edge independent case. For other stretch factors, however, the complexity of the decision problem for the direct version differs significantly for vertex or edge independent tree k–root-spanners. We fully characterize the Independent Direct Tree k–Root-Spanners by the following theorem.

Theorem 4.17

1. *A pair of edge independent direct tree k–root-spanners can be found in linear time, if it exists. Thus,* Edge Independent Direct Tree k–Root-Spanners *is in $\mathcal{P}$, for all rational $k \geq 2$.*

2. Vertex Independent Direct Tree k–Root-Spanners *is $\mathcal{NP}$-complete for all rational $k \geq 2$.*

The proofs are given in the next two subsections. As the main properties of edge independent direct tree k–root-spanners (see Lemma 4.18 and Corollary 4.20) can be extended to more than two edge independent direct tree k–root-spanners. Hence, the first part of Theorem 4.17 also holds for ℓ edge independent direct tree k–root-spanners for $\ell \geq 2$.

4.4.2 Edge Independent Direct Tree k–Root-Spanners

To prove the first part of Theorem 4.17, we first examine the structure of edge independent direct tree k–root-spanners (if they exist). Combining the results, we get an algorithm for deciding the existence, and, if so, for constructing a pair of edge independent direct tree k–root-spanners.

In the following, assume that G admits a pair of edge independent direct tree k–root-spanners. The first lemma establishes that there also exists a pair of edge independent direct tree k–root-spanners such that every vertex is connected to a parent vertex in at least one of the two trees. Moreover, if a vertex has more than one parent vertex, it is always possible to choose a direct connection to the parent level in both trees. Recall that $l(v)$ denotes the level index, i.e., the distance from r, for vertex v.

Lemma 4.18 *If $G = (V, E)$ admits a pair of edge independent direct tree k–root-spanners then there exists a pair T_1 and T_2 of edge independent direct tree k–root-spanners w.r.t. r such that*

1. *for all $v \in V \backslash \{r\}$ there is a $w \in L_{l(v)-1}$ such that either $\{v, w\} \in T_1$ or $\{v, w\} \in T_2$.*

2. *for all $v \in V \backslash \{r\}$ with $|N(v) \cap L_{l(v)-1}| \geq 2$:*
   there are $w_1, w_2 \in N(v) \cap L_{l(v)-1}$ with $w_1 \neq w_2$ such that $\{v, w_1\} \in T_1$ and $\{v, w_2\} \in T_2$.

Proof

1. The proof of Part 1 is by induction. Consider an arbitrary pair T_1' and T_2' of edge independent direct tree k–root-spanners. We modify T_1' and T_2' step by step such that the constraint to be proved is fulfilled:

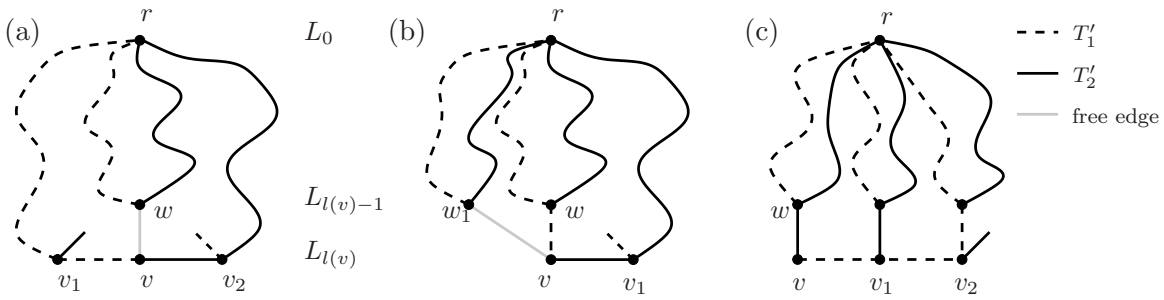

Figure 4.8: Illustrations for (a) Part 1 of Lemma 4.18, (b) Part 2 of Lemma 4.18, and (c) Lemma 4.19.

Let v be a vertex with minimal level index $l(v)$ such that v is connected to a parent vertex neither in T_1 nor in T_2. Let w be a parent vertex of v, and v_1 (and v_2, respectively) be the sibling vertex of v on $rp(v, T_1')$ (and $rp(v, T_2')$, respectively). See Figure 4.8(a) for an illustration. Obtain T_1 from T_1' by deleting edge $\{v, v_1\}$ and adding edge $\{v, w\}$, and let $T_2 := T_2'$. Continue with the new trees.

We show that, in each step, T_1 and T_2 remain edge independent direct tree k–root-spanners: If $l(v) = 1$, then $w = r$ and all conditions for T_1 and T_2 are fulfilled. Otherwise, by choice of v, we know that $d_{T_i'}(w, r) \leq k \cdot (l(v) - 1)$, and that the root paths $rp(w, T_i')$ are direct and edge disjoint. By construction, T_1 and T_2 are trees, all root paths are direct and the distance constraints are fulfilled. It remains to show that $rp(v, T_1)$ and $rp(v, T_2)$ are edge disjoint:

If $rp(v, T_2')$ and $rp(w, T_1')$ are vertex disjoint we are done. Otherwise, take the first common vertex x in $rp(v, T_2')$ and $rp(w, T_1')$. By induction, since $l(x) < l(v)$, the root paths $rp(x, T_1')$ and $rp(x, T_2')$ are edge disjoint and hence are $rp(v, T_1)$ and $rp(v, T_2)$.

2. Using Part 1 of the lemma, suppose that T_1' and T_2' fulfill the constraint given there. The proof for Part 2 then follows along the same lines. Figure 4.8(b) illustrates the situation. Here, T_2 is obtained from T_2' by deleting edge $\{v, v_1\}$ and adding edge $\{v, w_1\}$, and let $T_1 := T_1'$. □

The characterization can be further enhanced by examining the stretch factor of two edge independent direct tree k–root-spanners. As shown in the following lemma, the directness constraint dominates the stretch factor, such that the class of graphs admitting a pair of edge independent direct tree k–root-spanners for arbitrary k and the class of graphs admitting a pair of edge independent direct tree 2–root-spanners collapse.

Lemma 4.19 *If G admits a pair of edge independent direct tree k–root-spanners w.r.t. r then there exists a pair of edge independent direct tree 2–root-spanners w.r.t. r.*

Proof The key to proving the lemma is the following fact: If G admits a pair T_1' and T_2' of edge independent direct tree k–root-spanners then there exist also two edge independent direct tree k–root-spanners T_1 and T_2 such that for every v both root paths stay at most once within a level. Thus the distance to the parent level in T_1 and T_2 is at most 2. Stated formally:

$$(*) \qquad |\{w \mid w \in rp(v,T_i) \cap L_{l(v)},\ w \neq v\}| = 1 \quad \text{for } i \in \{1,2\}.$$

This can be seen by similar arguments as in Lemma 4.18 starting with T_1' and T_2' as indicated in Part 2 of that lemma. See Figure 4.8(c) for an illustration of the situation. Here, T_1 is obtained from T_1' by deleting edge $\{v, v_1\}$ and adding edge $\{v, w\}$, and T_2 is obtained from T_2' by deleting edge $\{v, w\}$ and adding edge $\{v, v_1\}$. Accumulating this over all levels results in a pair of edge independent direct tree 2–root-spanners. □

As a corollary from the previous lemmas, we get the following characterization of graphs that admit a pair of edge independent direct tree k–root-spanners: All vertices have at least two parent or sibling vertices in total.

Corollary 4.20 *A graph G does* not *admit a pair of edge independent direct tree k–root-spanners if and only if $|N(v) \cap L_{l(v)-1}| = 1$ and $N(v) \cap L_{l(v)} = \emptyset$ for some vertex v.*

As a consequence of property $(*)$ of Lemma 4.19, the connection of a vertex to its parent level in both tree 2–root-spanners is independent of the connections in lower levels. Accordingly, using the given characterization, we obtain a top–down algorithm as described in Figure 4.9. It starts from the top level vertices and constructs the edge independent direct tree 2–root-spanners level by level, if possible.

We begin with level L_1, the direct neighbors of the root. Once we have fixed the tree edges for all vertices of a level $L_{\ell-1}$, we continue with the vertices of level L_ℓ. For this aim, we choose an arbitrary vertex v of this level, and first check whether it is possible to connect it directly in both trees to a parent vertex (Step 2). If v has only one parent vertex, we have to check whether it is possible to find a second root path via a sibling vertex. If there is no sibling the algorithm stops indicating that G does not admit a pair of edge independent direct tree 2–root-spanners (Step 3(a)). Otherwise, by Lemma 4.18, the existence of two edge independent direct root paths

Procedure PROCESS-LEVEL(ℓ)

Input: A graph G (all vertices of level L_ℓ unmarked) with root r, and two trees T_1 and T_2 that are edge independent direct tree 2–root-spanners for vertices of level $L_{\ell-1}$ or lower.

Output: Edges of two edge independent direct tree 2–root-spanners T_1 and T_2 connecting $L_{\ell-1}$ and L_ℓ, if they exist.

Algorithm:

1. Arbitrarily choose an unmarked $v \in L_\ell$, and mark v;
   let $N := N(v) \cap L_{\ell-1}$. (* set of parent vertices of v. *)
2. If $|N| \geq 2$: (* v can be connected to $L_{\ell-1}$ in both trees. *)
   arbitrarily choose two vertices $v'_1, v'_2 \in N$;
   add $\{v, v'_1\}$ to T_1 and $\{v, v'_2\}$ to T_2.
3. If $N = \{v'\}$: (* v has one distinguished parent vertex. *)
   let $M = N(v) \cap L_\ell$. (* M is the set of sibling vertices of v. *)
   (a) If $M = \emptyset$, then stop: G does not admit a pair of edge independent direct tree 2–root-spanners.
   (b) Else, if all $w_i \in M$ unmarked, arbitrarily choose $w \in M$ and let w' be an arbitrary parent vertex of w;
   add $\{v, v'\}$, $\{v, w\}$ to T_1, and $\{w, w'\}$, $\{v, w\}$ to T_2;
   mark w.
   (c) Else, arbitrarily choose a marked w;
   add $\{v, w\}$ to the tree in which w is connected to its parent level, and add $\{v, v'\}$ to the other tree.

Figure 4.9: The algorithm for EDGE INDEPENDENT DIRECT TREE 2–ROOT-SPANNERS for vertices of level L_ℓ

fulfilling the distance constraint is guaranteed. Steps 3(b) and 3(c) construct these paths. Step 3(c) is necessary to fulfill also condition $(*)$ of Lemma 4.19.

The correctness of the algorithm follows directly from Corollary 4.20. As every vertex is treated exactly once, it can be implemented in linear time. Thus, the first part of Theorem 4.17 is proved.

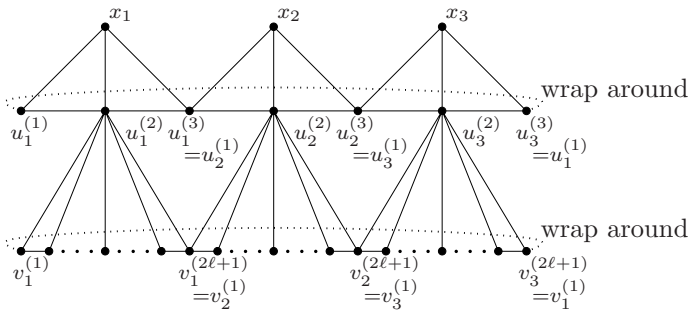

Figure 4.10: A clause component for the construction of an instance of Vertex Independent Direct Tree k–Root-Spanners.

4.4.3 Vertex Independent Direct Tree k–Root-Spanners

For the vertex independent case the situation is different, because the induction in Lemma 4.18 does not work for vertex disjoint root paths. In contrast, in this subsection, we prove Part 2 of Theorem 4.17, the $\mathcal{NP}$-completeness of Vertex Independent Direct Tree k–Root-Spanners. We again use a reduction from Not–All–Equal 3–Satisfiability.

Let $k \geq 2$ be a rational number, $k = \frac{p}{q}$, where p and q are chosen to be positive integers with $p \geq 2q$ and $q > 2$. Given an instance (U, C) of Not–All–Equal 3–Satisfiability, we construct the graph G for Vertex Independent Direct Tree k–Root-Spanners as follows.

- Construct the *variable component* as in the proof of Section 4.3.
- For every clause $c = x_1 \vee x_2 \vee x_3 \in C$, create a *clause component* (see Figure 4.10):
  - Create *clause vertices* x_1, x_2, and x_3, one for each literal in c.
  - For each $x_i, i \in \{1, 2, 3\}$, create a path of length 2 consisting of three new vertices $u_i^{(1)}$, $u_i^{(2)}$, and $u_i^{(3)}$, and connect each of the vertices with x_i. Melt together $u_1^{(3)}$ with $u_2^{(1)}$, $u_2^{(3)}$ with $u_3^{(1)}$, and $u_3^{(3)}$ with $u_1^{(1)}$ ('wrap around').
  - The third part is similar, but now creating paths of length 2ℓ for every x_i, $i \in \{1, 2, 3\}$. All these new vertices are connected to $u_i^{(2)}$, and the end-vertices of the new paths are melted together as above.
- Identify the clause vertices with their corresponding literal vertices.

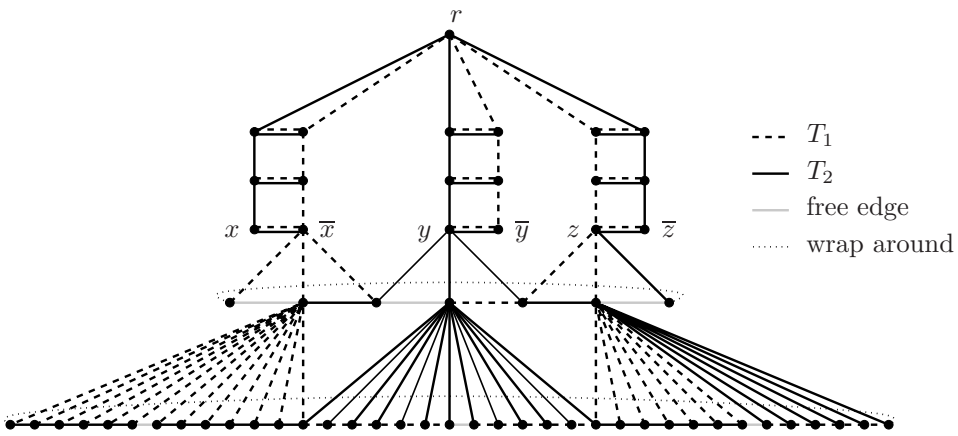

Figure 4.11: The construction of an instance of VERTEX INDEPENDENT DIRECT TREE k–ROOT-SPANNERS for $k = \frac{11}{5}$ and clause $c = \overline{x} \vee y \vee z$ and a pair T_1 and T_2 of vertex independent direct tree k–root-spanners.

In Figure 4.11, the part of G for $k = \frac{11}{5}$ and clause $c = \overline{x} \vee y \vee z$ is shown. Again, G can be constructed in polynomial time and it remains to prove that there is a not-all-equal truth assignment θ for (U, C) if and only if G admits a pair of vertex independent direct tree k–root-spanners w.r.t. r.

From θ to T_1 and T_2. Given a not-all-equal truth assignment θ, we construct T_1 and T_2 as indicated in Figure 4.11. Let $\theta(x) = \theta(y) =$ `false` and $\theta(z) =$ `true`. The construction starts again by choosing true literal edges for T_1 and false literal edges for T_2. The choice of the remaining edges is straightforward from the example in Figure 4.11. It is easy to see that T_1 and T_2 are vertex independent direct tree k–root-spanners of G.

From T_1 and T_2 to θ. Again, we first show that a pair of vertex independent direct tree k–root-spanners induces a truth assignment and then conclude that this truth assignment is not-all-equal. As in Section 4.2.3, Parts 1 and 3 of Lemma 4.7 and also properties (P1)–(P3) hold. If T_1 and T_2 are vertex independent direct tree k–root-spanners, we now have the following additional properties:

(P4)' Because of the directness constraint, root paths from vertices within the variable component can only stay within the variable component.

(P5)' Consider a clause component. Due to the directness constraint of the root paths, the clause components are independent of each other with

respect to the choice of edges in two vertex independent direct tree k–root-spanners. They only depend on the choice of the corresponding literal edges $\{r, x^{(0)}\}$ and $\{r, \overline{x}^{(0)}\}$.

(P6)' For $i \in \{1,2,3\}$, $l(u_i^{(j)}) = q-1$ and $l(v_i^{(j)}) = q$.

Suppose we are given two vertex independent direct tree k–root-spanners T_1 and T_2. By the above properties, T_1 and T_2 induce a truth assignment by setting $\theta(x) := \texttt{true}$ if and only if $\{r, x^{(1)}\} \in T_1$. We show that θ is a not-all-equal truth assignment by contradiction:

W.l.o.g., suppose that there is a clause $c = x_1 \vee x_2 \vee x_3$ where $\{r, x_i^{(1)}\} \in T_1$ and hence $\theta(x_i) = \texttt{true}$ for $i \in \{1,2,3\}$. Moreover,

$$|rp(x_i^{(q-2)}, T_1)| = q-2 \quad \text{and} \quad |rp(x_i^{(q-2)}, T_2)| = q-1.$$

Consequently, for $i \in \{1,2,3\}$

$$\begin{array}{ll} \text{either} & |rp(u_i^{(2)}, T_1)| = |rp(u_i^{(2)}, T_2)| = q \\ \text{or} & |rp(u_i^{(2)}, T_1)| = q-1 \quad \text{and} \quad |rp(u_i^{(2)}, T_2)| = q+1. \end{array}$$

Now all possible choices of root paths result in a contradiction at a vertex $v_i^{(j)}$ for some $1 \leq j \leq 2\ell+1$.

Observe that on the other hand, if the variable setting of a clause is not-all-equal, then it is possible to find T_1 and T_2 such that

$$|rp(u_i^{(2)}, T_1)| = q-1 \quad \text{and} \quad |rp(u_i^{(2)}, T_2)| = q.$$

4.5 Further Remarks

In this chapter, we have considered *independent tree k–spanners*, i.e., a pair of independent spanning trees that are tree k–spanners, thus combining the aspects of fault-tolerance (with respect to both vertices and edges), distance guarantee, sparseness, and structural simplicity. *k–root-spanners* are a less restrictive variant of k–spanners, where the distance constraint is relaxed to distances between the vertices and the root. As a special case, we have considered tree k–root-spanners where the paths from any vertex to the root are detour-free. Most of the considered problems have been shown to be $\mathcal{NP}$-hard, but we have also given efficient algorithms for some special cases.

Observe that the complexity of Independent Tree 3–Spanners remains open. Until now we have neither found a characterization of graphs

admitting a pair of independent tree 3–spanners nor a proof of its $\mathcal{NP}$-completeness. On the other hand, it would be interesting to examine other variants of tree k–root-spanners such as *balanced* tree k–root-spanners: here, the length of the paths in both trees may at most differ by a fixed number (1, for example), hence guaranteeing similar delay behavior in both tree k–root-spanners.

Chapter 5

Steiner Tree Spanners

In the previous chapter we have considered tree k–root-spanners, which are a relaxed version of tree k–spanners. Following this line, we now discuss a further variant; but instead of slackening the distance constraint, we now relax on the condition that the subgraph is spanning.

5.1 Tree Spanners for Subgraphs

Reconsider again tree k–spanners as discussed, for example, in Sections 2.2 and 4.2. The concept is very restrictive in the sense that the class of graphs that admit a tree k–spanner for a fixed k is quite small. In some settings in the design of subnetworks, however, it is not strictly necessary to span all vertices or edges. Stated from another point of view, we are given a graph with specified terminals and the goal is to cover all edges induced by them; in order to do this there are additional vertices and edges that may or may not be used. In this chapter, we introduce a generalized version of tree k–spanners, called *Steiner tree k–spanners*, which models this situation.

Steiner Tree Spanners

Let us first recall the definition of the well-known Minimum Steiner Tree Problem in unweighted graphs (Problem D.4 in Appendix D). We are given a graph $G = (R \dot{\cup} S, E)$ with a set of *terminal vertices* R and a set of *Steiner vertices* S such that R and S are disjoint. Minimum Steiner Tree asks for a minimum length tree within G that includes all terminals R. In contrast to the problem of finding a minimum spanning tree, the Steiner tree may include some of the Steiner vertices, but not necessarily all, as long as all terminals are included in the tree. See for example Hwang et al. (1992) for a

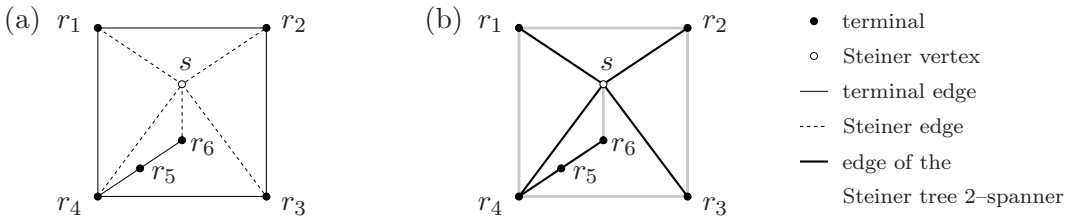

Figure 5.1: (a) The graph induced by the terminals does not have a tree 2–spanner, but (b) a Steiner tree 2–spanner.

survey and Appendix D for complexity results on Minimum Steiner Tree.

Translated to our case, we are again given an input graph consisting of *terminals* and *Steiner vertices*. We are now looking for a *tree k–spanner* that includes all terminals and covers the edges induced by them. Steiner vertices may be used as intermediate, auxiliary vertices. By this, the graph induced by the terminals itself may not admit a tree k–spanner, but using some Steiner vertices might yield a Steiner tree k–spanner. See Figure 5.1 for an example. When dealing with Steiner tree k–spanners, the decision problem as well as the optimization problem are of interest.

Steiner spanners have appeared in the literature for example in Althöfer et al. (1993), where they deal with absolute lower bounds on the number of edges that an arbitrary Steiner k–spanner (not necessarily being a tree) can have. Chen et al. (1996) examine lower bounds for the running time of algorithms that find general Steiner k–spanners with $o(n \log n)$ edges for the restricted case of geometric graphs. Both papers do not consider the trees.

As mentioned in the previous chapter, some authors have considered *delay-bounded minimum Steiner trees*, where one is interested in finding a Steiner tree T such that T fulfills some given distance constraints especially for the distances from a specified root (see for example Zhu et al. (1995); Rouskas and Baldine (1997); Haberman and Rouskas (1997)). But there an absolute, fixed delay bound is imposed on the distance of the vertices to a root, and the concept is more related to root-spanners. To our knowledge, this is the first time where the generalization of tree k–spanners in terms of Steiner k–spanners is examined.

Formal Definitions and Results

We now define the generalized version of tree k–spanners formally. In the following, the vertex set of the input graph G is partitioned into two disjoint subsets, the set of *terminals* R and the set of *Steiner vertices* S. The edges of $G[R]$, i.e., the edges that are induced by R, are called *terminal edges*, all

other edges are called *Steiner edges*. Denote by $S(T)$, where T is a subgraph, the set of Steiner vertices in G used in T.

Definition 5.1 (Steiner tree k–spanner)
Given a graph $G = (R \dot\cup S, E)$ where $G[R]$ is connected, and an arbitrary $k \geq 1$, a subgraph T of G is a Steiner tree k–spanner *of G if T is a tree, and*

$$d_T(u, v) \leq k \quad \text{for all edges } e = \{u, v\} \text{ of } G[R].$$

Observe that we implicitly use Lemma 2.2 here. In particular, a Steiner tree k–spanner T must contain all vertices of R, and may include some of the vertices of S too. The edges of T may be a combination of both terminal and Steiner edges. Note that Steiner edges of $G[V(T)]$ do not necessarily have to be covered in T (cf. edge $\{s, r_6\}$ in Figure 5.1), while terminal edges of $G[V(T)]$ do have to be covered.

Observe that, by this definition, it is possible that the distance between two vertices in T may be shorter than their distance in $G[R]$. Note also that it suffices to consider only graphs where $G[R]$ is connected because otherwise we can cover all connected components separately.

In contrast to tree k–spanners, it is interesting to consider a decision problem as well as an optimization problem defined as follows:

Problem 5.2 Steiner Tree k–Spanner

Given: A graph $G = (R \dot\cup S, E)$ where $G[R]$ is connected.

Problem: Does G admit a Steiner tree k–spanner?

Problem 5.3 Minimum Steiner Tree k–Spanner

Given: A graph $G = (R \dot\cup S, E)$ where $G[R]$ is connected, and a positive integer K.

Problem: Does G admit a Steiner tree k–spanner T such that $|S(T)| \leq K$?

In the latter, we are looking for a Steiner tree k–spanner of the input graph that has the least number of Steiner vertices, if one exists at all. Observe that the number $|E(T)|$ of edges of a Steiner tree k–spanner T is related to the number $|S(T)|$ of Steiner vertices: $|E(T)| = |R| + |S(T)| - 1$. If we now measure the quality of T in terms of $|E(T)|$ instead of $|S(T)|$, then a solution to Minimum Steiner Tree k–Spanner is also a minimum Steiner tree k–spanner in this sense.

Since Steiner Tree k–Spanner is a generalized version of Tree k–Spanner, the $\mathcal{NP}$-completeness for $k \geq 4$ is immediate. It remains to consider the cases where k is 1, 2, or 3. The case $k = 1$ coincides with Tree

1–Spanner because no terminal edge can be covered using Steiner edges. Thus, it suffices to check whether $G[R]$ is a tree because this is the only possibility for G having a Steiner tree 1–spanner.

In Sections 5.2 and 5.3, we are dealing with the interesting cases of $k = 2$ and $k = 3$. For $k = 2$, we first develop a model in terms of a tree covering problem. Using this, we then show that Steiner Tree 2–Spanner is $\mathcal{NP}$-complete, in contrast to Tree 2–Spanner. Observe that our model only copes with the case $k = 2$; it is inappropriate for other stretch factors. But we show in Subsection 5.3.3 how we can use the $\mathcal{NP}$-completeness for $k = 2$ to show the $\mathcal{NP}$-completeness for $k = 3$.

If we now assume that we know in advance that a graph G admits a Steiner tree k–spanner for some fixed k, it makes sense to also study the optimization problem. Clearly, the $\mathcal{NP}$-hardness for this follows directly from the results above. In Section 5.4, we examine the approximability status and show the following: Minimum Steiner Tree k–Spanner is hard to approximate within anything better than logarithmic ratio.

Finally, in Section 5.5, we consider further tree covering problems that are similar to the one that arises in the light of Steiner Tree 2–Spanner. These problems do not carry over to the context of Steiner tree k–spanners, but we consider them of independent interest.

5.2 A Model for Steiner Tree 2–Spanners

Instead of trying to solve Steiner Tree 2–Spanner directly, we first examine the underlying structure of Steiner tree 2–spanners in order to find a suitable model. When dealing with Steiner tree 2–spanners, we have the following situation: a terminal edge can either be covered by itself or by a path of length 2 that consists either of two terminal edges or two Steiner edges. That means that exactly one Steiner vertex may be used for covering a terminal edge unless it is covered by two terminal edges or itself.

Observe that we do not consider the degenerated case where a singleton Steiner edge is added to a Steiner tree 2–spanner, i.e., an edge that connects an unused Steiner vertex to a terminal. This Steiner edge does not contribute to covering any edge in $G[R]$.

5.2.1 The Role of the Biconnected Components

As shown in Cai and Corneil (1995), there is an efficient algorithm to decide whether or not a graph admits a tree 2–spanner. Among others, this algorithm uses the fact that the biconnected components may be treated

separately and that any tree 2–spanner must contain all edges that connect the vertices of a 2–separator, see Lemma 2.7.

Steiner Tree 2–Spanners in a Block

Also in the case of Steiner tree 2–spanners, the biconnected components of $G[R]$ play an important role: Consider a biconnected graph $G[R]$ that does not admit a tree 2–spanner. Then, the only possibility for $G[R]$ for having a Steiner tree 2–spanner is the existence of a universal Steiner vertex:

Lemma 5.4 *Let $G = (R \dot\cup S, E)$, and let $G[R]$ be a biconnected graph that does not admit a tree 2–spanner. Then a subgraph T of G is a Steiner tree 2–spanner of G if and only if there is a Steiner vertex $s \in S$ such that $\{s, r\} \in E(T)$ for all terminals $r \in R$.*

Proof The 'if-part' of the lemma is trivial. We show the opposite direction by contradiction. Suppose that G does not admit a tree 2–spanner, that T is a Steiner tree 2–spanner, and that T is not a star centered at a Steiner vertex.

First, assume that T contains a terminal edge $\{u_1, v_1\}$. Since T is not a tree 2–spanner, there exists a terminal edge $\{u_2, v_2\}$, which is covered by a path via a Steiner vertex. W.l.o.g., assume that u_1, v_1, u_2, and v_2 are pairwise distinct. As $G[R]$ is biconnected, there are paths consisting of terminal edges connecting u_1 with u_2, and v_1 with v_2 (or u_1 with v_2, and v_1 with u_2, respectively). All the edges of these path have to be covered by a path of length at most 2. By the preselection of $\{u_1, v_1\}$ and the Steiner edges connecting u_2 and v_2, this results in a cycle, a contradiction to T being a tree.

The case where T contains only Steiner edges, but where more than one Steiner vertex is involved can be treated similarly: If the covering paths have length at most 2 then this situation induces again a cycle in T. Consequently, T consists of one central Steiner vertex together with all Steiner edges connecting the Steiner vertex to each vertex of R. □

Note that this lemma heavily uses the stretch factor of 2. It does not hold for larger values of k.

By the previous lemma, it is shown that Steiner Tree 2–Spanner can be solved polynomially when $G[R]$ is biconnected: There either exists a tree 2–spanner or the Steiner tree 2–spanner is a star centered at a Steiner vertex. No other Steiner tree 2–spanner can exist. Both possibilities can be checked in polynomial time (cf. Theorem 2.8), and hence, Problems 5.2 and 5.3 can be solved efficiently in this case.

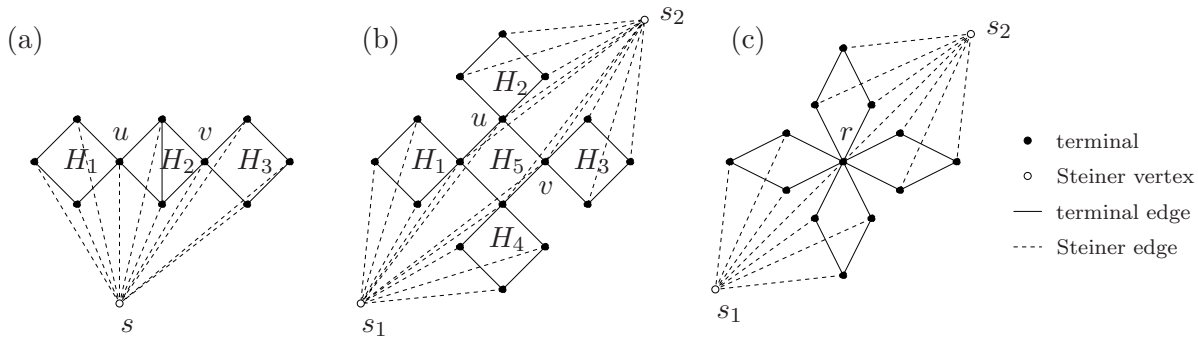

Figure 5.2: (a) Block H_2 admits a tree 2–spanner but this is not part of the Steiner tree 2–spanner of the whole graph. (b) Every block has a universal Steiner vertex but there is no Steiner tree 2–spanner. (c) r is an articulation vertex shared by four blocks. This graph admits a Steiner tree 2–spanner.

Now consider a graph G where $G[R]$ is connected and consists of several blocks. Let T be a Steiner tree 2–spanner of G. Then, for every block there is *at most one* universal Steiner vertex in a T:

Lemma 5.5 *Let G be a graph where $G[R]$ is connected. If T is a Steiner tree 2–spanner of G then for every block H of $G[R]$, $T[V(H)]$ is either a tree 2–spanner of H or there is* exactly one *Steiner vertex $s \in S$ such that $\{s, r\} \in E(T)$ for all $r \in V(H)$.*

Proof As a consequence of Lemma 5.4, there must either exist a tree 2–spanner or a universal Steiner vertex for every block H of $G[R]$. By contradiction, suppose that T contains two (or more) universal Steiner vertices s_1 and s_2 for a block H of $G[R]$, and let u and v be two vertices of H. Then, $\{u, s_1, v, s_2, u\}$ forms a cycle in T, a contradiction. □

In the following, we say that a Steiner vertex s *spans* a block H of $G[R]$ whenever s is universal for H and the star centered at s is a subgraph of the Steiner tree 2–spanner. In other words, the previous lemma indicates that every block of a graph is spanned by exactly one Steiner vertex (unless it is covered by a tree 2–spanner).

Unfortunately, the existence of a tree 2–spanner or universal Steiner vertex for every block is not sufficient. It does *not* suffice to examine every block separately and then simply combine the Steiner tree 2–spanners of every block (as it is the case for tree 2–spanners). For example, even if a block admits a tree 2–spanner, it may be inevitable to span this block by a Steiner vertex. See for example Figure 5.2(a): Block H_2 admits a tree 2–spanner T,

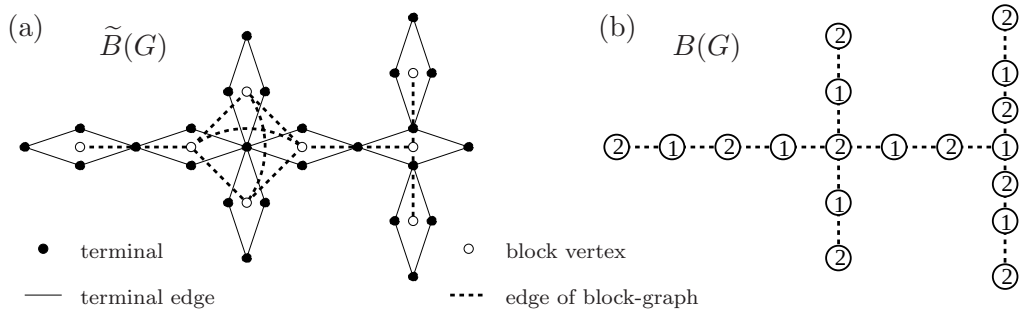

Figure 5.3: (a) Simple block-graph $\widetilde{B}(G)$ with clique, and (b) corresponding block-graph $B(G)$.

whereas Blocks H_1 and H_3 do not. We cannot use T together with the star centered at s that spans H_1 and H_3, since this results in a cycle containing vertices u and v. The only Steiner tree 2–spanner of the whole graph consists of the star centered at s spanning all blocks.

Figure 5.2(b) shows an example of a graph where every block has some universal Steiner vertex, but there is no Steiner tree 2–spanner for the whole graph: By Lemma 5.5, in a Steiner tree 2–spanner T, Block H_5 can be spanned either by s_1 or by s_2. W.l.o.g., assume the first such that $\{u, s_1\}$ and $\{v, s_1\}$ are edges in T. Since H_2 and H_3 have to be spanned by s_2 also edges $\{u, s_2\}$ and $\{v, s_2\}$ belong to T, thus inducing a cycle.

The situation in Figure 5.2(c) is different: r is an articulation vertex that is shared by four blocks. Here, the two stars centered at s_1 and s_2, respectively, form a Steiner tree 2–spanner. Consequently, articulation vertices may be touched (spanned) by more than one star in T. Thus, we have to take into account the whole block structure of $G[R]$.

The Block-Graph

Observe that the blocks of a connected graph have a tree-like structure: Define the *simple block-graph* $\widetilde{B}(G)$ to be the graph that arises from G by taking the blocks of G as nodes (to avoid ambiguities we refer to the vertices of $\widetilde{B}(G)$ as *nodes*). Two of these nodes are connected if the corresponding blocks share an articulation vertex. See Figure 5.3(a) for an example.

By the definition of a block, a simple block-graph can be viewed as a tree where some of the tree vertices of degree $\delta \geq 3$ are replaced by a clique of size δ. These cliques arise from articulation vertices that are shared by more than two blocks. Accordingly, the simple block-graph is not a tree. Our goal now is to modify the notion of the block-graph such that it is a tree and remains appropriate for our purpose of finding Steiner tree 2–spanners.

We therefore extend the notion of the simple block-graph to a ①–②–*tree* (see also Harary (1969)). This consists of two different types of nodes: a ①–node for every block, and a ②–node for every articulation vertex. We connect the nodes in a way such that a ②–node is connected to a ①–node by an edge, whenever the corresponding articulation vertex is contained in the respective block. For reasons of simplicity, we also create an additional ②–node for every block that contains only one articulation vertex. We refer to this graph as *block-graph*, denoted by $B(G)$; see Figure 5.3(b) for an example.

Altogether, we have the following straightforward properties: $B(G)$ is a tree, and the leaves of $B(G)$ are ②–nodes. Moreover, every ①–node (or ②–node, respectively) is adjacent only to ②–nodes (or ①–nodes, respectively).

Certainly, there are also other methods to adapt the simple block-graph in order to make it a tree, for example by replacing each clique of size at least 3 by a new node that is adjacent to all nodes of the clique. A second simple modification is collapsing each clique of size at least 3 into one node. But observe that these straightforward methods do not distinguish between the new nodes and the ones that represent real blocks of G. As we will see later, this is necessary for a proper model of Steiner Tree 2–Spanner.

In the sequel, we abbreviate the notation $B(G[R])$ as $B(G)$. Concerning Steiner tree 2–spanners, we get the following key observation:

Lemma 5.6 *If two blocks H_1 and H_2 of $G[R]$ are spanned in a Steiner tree 2–spanner T by the same universal Steiner vertex s then also all other blocks that lie on the path from H_1 to H_2 in $B(G)$ are spanned by s in T.*

Proof Let T be a Steiner tree 2–spanner of G. By contradiction, suppose that there are three consecutive ①–nodes in $B(G)$ that correspond to blocks H_1, H_2 and H_3, such that H_1 and H_3 are spanned by a Steiner vertex s_1 and H_2 is not spanned by s_1. Let u and v be the articulation vertices that separate H_2 (see Figure 5.4). Then, $\{u, s_1\}$ and $\{v, s_1\}$ in T. We have to consider two cases:

1. H_2 is spanned by s_2, a second Steiner vertex, and hence path $\{u, s_2, v\}$ is in T.

2. H_2 is spanned by a tree 2–spanner.

In both cases, the edges $\{u, s_1\}$ and $\{v, s_1\}$ induce a cycle, a contradiction to T being a tree.

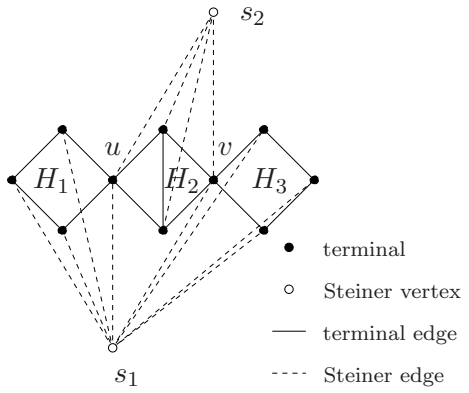

Figure 5.4: Three consecutive blocks.

It is easy to see that the property also holds for blocks with more than one intermediate block H_2. □

5.2.2 An Equivalent Tree Covering Problem

Our goal is to model Steiner Tree 2–Spanner in terms of an equivalent problem that is easier to handle. For this, we need some more notation for the block-graph: A subtree of a ①–②–tree B is called ①–②–*subtree*. A ①–②–subtree T of a ①–②–tree B is called *proper* if the following holds: whenever a ①–node b belongs to T then also all ②–nodes that are adjacent to b in B belong to T. The intuition here is as follows: whenever a block (corresponding to a ①–node) is selected, we also select all incident articulation vertices (corresponding to a ②–node). Therefore, in a proper ①–②–subtree, the neighborhood of every ①–node is preserved. A collection of proper ①–②–subtrees that are node disjoint is called a *proper* ①–②–*forest*.

We use the block-graph as a basis for our model. For this, we first enlarge the given graph as follows: For every block that admits a tree 2–spanner, we create a new Steiner vertex (called a *fake* Steiner vertex, in contrast to *real* Steiner vertices). Note that every block of $G[R]$ that consists of only a single edge admits a tree 2–spanner and is thus treated here. In the sequel, we always use this extended set of Steiner vertices.

Now, for every Steiner vertex s of G, we denote by $F^{(s)}$ the subgraph of $B(G)$ that is induced by the ①–nodes that correspond to blocks that may be spanned by s, together with their adjacent ②–nodes. Then, $F^{(s)}$ is a proper ①–②–forest of $B(G)$, consisting of several maximal proper ①–②–subtrees of $B(G)$. Denote these proper ①–②–subtrees by $\{T_1^{(s)}, \ldots, T_\ell^{(s)}\}$, and let $\mathcal{F}$ be the collection of the $F^{(s)}$ for all (real or fake) Steiner vertices s.

Our goal is to define a tree covering problem in the following sense: Given a tree and a collection of subtrees thereof, find a cover of the tree, i.e., a collection of the subtrees such that each vertex of the tree is included in a subtree, subject to some further constraints. Here, we take $B(G)$ as the underlying tree and $\mathcal{F}$ as the collection of subgraphs. In the sequel, we develop the constraints such that a covering selection of subtrees induces a Steiner tree 2–spanner and vice versa.

For this, we translate the facts about Steiner tree 2–spanners that we have compiled above into the context of the $(B(G), \mathcal{F})$ tree cover problem:

1. Lemma 5.6 indicates that all 'intermediate' blocks must be spanned whenever two blocks are to be spanned by one distinguished Steiner vertex. Stated the other way round: whenever a block H_1 is spanned

by a Steiner vertex s_1, and another block H_2 is spanned by a different Steiner vertex s_2, then no other block H_3 such that H_2 lies on the path from H_1 to H_3 in the block-graph is spanned by s_1. In the context of the tree cover problem, this means that we may choose at most one $T_i^{(s)}$ of $F^{(s)}$ for every s.

2. If we select one particular Steiner vertex s, it is not necessary that we use s for all blocks that are potentially spanned by s. In other words, we may pick only some of these blocks, as long as these blocks induce a connected component within $B(G)$. In the context of the tree cover problem, this corresponds to choosing subtrees of the trees contained in each $F^{(s)}$.

3. By Lemma 5.5, each block has to be spanned by exactly one Steiner vertex. That means that each ①–node of $B(G)$ has to be covered exclusively. Since articulation vertices may be touched repeatedly, ②–nodes may be covered more than once. We can achieve this by considering edge disjoint proper ①-②–subtrees: Two proper ①-②–subtrees of the same ①-②–tree that are edge disjoint never share a ①–node. They may share at most one ②–node.

Using this, we can model Steiner Tree 2–Spanner as a problem of an edge disjoint cover of the ①-②–tree by proper ①-②–subtrees:

Problem 5.7 Edge Disjoint ①-②–Subtree Cover

Given: A ①-②–tree B and a set of ①-②–forests $\mathcal{F} = \{F^{(1)}, \ldots, F^{(n)}\}$ such that each $F^{(j)}$ consists of a set of node disjoint proper ①-②–subtrees $T_i^{(j)}$ of B.

Problem: Find a cover of the nodes of B consisting of edge disjoint proper ①-②–subtrees $\widehat{T}^{(j)}$ of a $T_i^{(j)}$ such that every index j is chosen at most once.

Observe how the three topics above are converted to Edge Disjoint ①-②–Subtree Cover: Topic 1 is reflected by claiming that every index j is chosen at most once. Topic 2 is implemented by considering proper ①-②–subtrees, and finally Topic 3 is achieved by considering edge disjoint ①-②–subtrees. It remains to show that both problems are equivalent:

Lemma 5.8 *G admits a Steiner tree 2–spanner if and only if there is a solution to* Edge Disjoint ①-②–Subtree Cover *with input $(B(G), \mathcal{F})$ as described above.*

Proof By construction, $(B(G), \mathcal{F})$ is a feasible instance of Edge Disjoint ①–②–Subtree Cover. Given a Steiner tree 2–spanner of G it is straightforward from the remarks above that we can select a tree cover that fulfills the conditions given in the problem description. It remains to prove the other direction.

Given a solution $\{\widehat{T}_{i_1}^{(s_1)}, \ldots, \widehat{T}_{i_\ell}^{(s_\ell)}\}$ of Edge Disjoint ①–②–Subtree Cover, we can construct a Steiner tree 2–spanner as follows: For everys_j that corresponds to a real Steiner vertex, take the star centered at s_j including all vertices that belong to the blocks corresponding to the ①–nodes of $\widehat{T}_{i_j}^{(s_j)}$. All these edges must exist by definition of $T_{i_j}^{(s_j)}$. Combine all these stars to form T. For every selected fake Steiner vertex, simply add the corresponding tree 2–spanner. Then, T is a tree, and all terminal edges are covered by a path of length at most 2. □

Furthermore, also any instance of Edge Disjoint ①–②–Subtree Cover can be re-interpreted and re-constructed in terms of Steiner Tree 2–Spanner. We have thus found an equivalent alternative formulation of Steiner Tree 2–Spanner, and use this in the following section to discuss its complexity status.

5.3 Complexity of Finding Steiner Tree *k*–Spanners

As mentioned in the introduction, the complexity status remains to be settled for the cases $k = 2$ and $k = 3$. In this section, we show that Steiner Tree k–Spanner is $\mathcal{NP}$-complete in general for both stretch factors, and discuss some polynomially solvable special cases for $k = 2$.

5.3.1 Hardness for $k = 2$

In this subsection, we deal with the case $k = 2$. Instead of showing the hardness of hardness of Steiner Tree 2–Spanner directly, we prove the $\mathcal{NP}$-completeness of the equivalent problem Edge Disjoint ①–②–Subtree Cover:

Theorem 5.9 Edge Disjoint ①–②–Subtree Cover *is $\mathcal{NP}$-complete.*

Proof It is clear that Edge Disjoint ①–②–Subtree Cover is in $\mathcal{NP}$. We show the $\mathcal{NP}$-completeness of the problem by a reduction from the

following $\mathcal{NP}$-complete Domatic Number Problem (Problem D.1 in Appendix D): We are given a graph $G = (V = \{v_1, \dots, v_{|V|}\}, E)$ and a positive integer $K \leq |V|$. The problem is to decide whether V can be partitioned into $\ell \geq K$ disjoint sets $V_1, V_2, \dots, V_\ell$ such that every V_i is a dominating set for G (i.e., such that for every $1 \leq i \leq \ell$, every vertex in V is adjacent to at least one vertex in V_i)? W.l.o.g., we can restrict ourselves to the case $\ell = K$.

Let us first review the problem: In a feasible solution $V_1, V_2, \dots, V_K$ for the instance G of Domatic Number, every vertex v_j appears in exactly one of the V_i. The V_i can be viewed as bins, and every v_j is put into exactly one of them. Moreover, the cumulated, closed neighborhood of all of the vertices in each of the bins is V; i.e.,

$$V_i \cup \bigcup_{v_j \in V_i} N(v_j) = V \quad \text{for all } i \in \{1, \dots, K\}.$$

Consequently, each of the bins 'induces' an isomorphic copy of V.

In the reduction from Domatic Number to Edge Disjoint ①–②–Subtree Cover, the idea is as follows: Each isomorphic copy of V that is induced by one of the bins, say V_i, is modeled by an individual component B_i of the block-graph, which each contains one ①–node for every vertex of V. The adjacencies in G are reflected in $\mathcal{F}$. Formally, given an instance of Domatic Number, we construct the instance of Edge Disjoint ①–②–Subtree Cover as follows:

- The ①–②–tree B consists of one central ①–node c and K isomorphic block-graph components B_i for $1 \leq i \leq K$: Create a ②–node a_i and connect it to $|V|$ new ①–nodes b_i^j for $1 \leq j \leq |V|$; add a new ②–node to every ①–node as a leaf. Finally, connect each a_i to the central ①–node c. See Figure 5.5(a).

- $\mathcal{F}$ consists of $1 + |V|$ ①–②–forests $F^{(0)}, \dots, F^{(|V|)}$ as follows (see Figure 5.5(b)):

  - $F^{(0)}$ consists of just one proper ①–②–subtree $T^{(0)}$ induced by the nodes a_i and by c.
  - For $1 \leq j \leq |V|$, $F^{(j)}$ consists of K disjoint (but isomorphic) proper ①–②–subtrees $T_i^{(j)}$ for $1 \leq i \leq K$, one in each block-graph component. $T_i^{(j)}$ contains the ②–node a_i and the ①–node b_i^ℓ if v_j is adjacent to v_ℓ in G, or if $\ell = j$. Add the corresponding ②–nodes together with all induced edges.

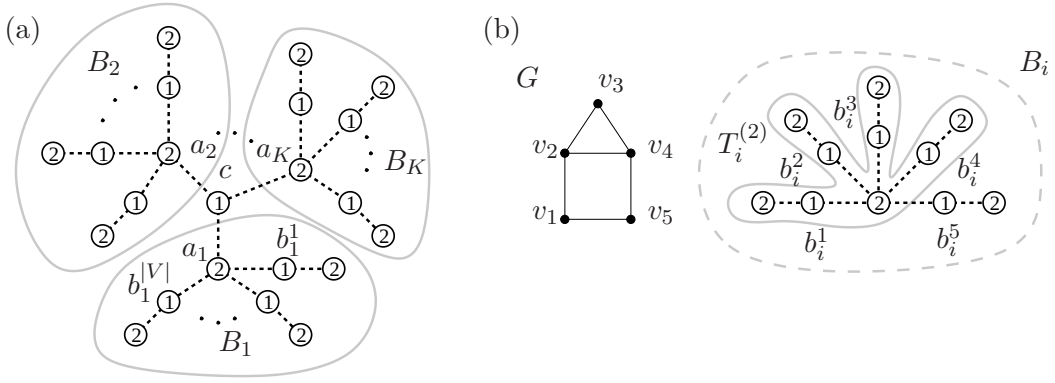

Figure 5.5: (a) The construction of B, and (b) of the ①–②–subtree $T_i^{(2)}$ from the given graph G.

In B, every block-graph component B_i represents a copy of V. Each B_i is a ①–②–star centered at a ②–node a_i, i.e., a proper ①–②–tree in which a central ②–node is adjacent to some ①–nodes, and each of these ①–nodes is adjacent to exactly one further ②–node. Every ①–node b_i^j corresponds to the vertex v_j in V. The central ①–node c works as a separator.

The adjacencies of a vertex v_j in G are modeled by the $F^{(j)}$: Each $T_i^{(j)}$ contains all ①–nodes b_i^j that correspond to the closed neighborhood of v_j in G (i.e., $\{v_j\} \cup N(v_j)$). Observe that all $T_i^{(j)}$ are isomorphic and node disjoint for a fixed j, and that none contains the central ①–node c. Moreover, every $T_i^{(j)}$ contains only nodes from one block-graph component B_i.

Thus, if $T_i^{(j)}$ or a proper ①–②–subtree thereof is selected in a solution of $(B, \mathcal{F})$, then this exactly mirrors the case where v_j is put into bin V_i in a solution of Domatic Number. In particular, ②–node a_i mirrors the bin V_i: V_i contains a vertex v_j if a_i is covered by $T_i^{(j)}$ (or a subtree thereof). Since we are only allowed to choose at most one ①–②–tree from every ①–②–forest $F^{(j)}$, this corresponds to the exclusive selection of vertices to bins.

This construction gives a feasible instance of Edge Disjoint ①–②–Subtree Cover, and it can be constructed in polynomial time. Furthermore, all ①–nodes b_i^j are covered by at least one of the proper ①–②–subtrees. It remains to prove the equivalence of the instances.

Lemma 5.10 *Let $(B, \mathcal{F})$ be as constructed above from the instance G of* Domatic Number. *G has a solution of* Domatic Number *if and only if $(B, \mathcal{F})$ has a feasible edge cover.*

Proof It is easy to see from the remarks above that an edge disjoint ①–②–subtree cover of $(B, \mathcal{F})$ induces a feasible partition of V by selecting v_j

for V_i if a subtree of $T_i^{(j)}$ is in the cover.

For the other direction, given a partition $V_1, V_2, \ldots, V_K$ of V that is a solution of Domatic Number, we get an edge disjoint ①–②–subtree cover of $(B, \mathcal{F})$ as follows:

- Choose $T^{(0)}$ to cover c.
- If v_j is in V_i then select $T_i^{(j)}$.

As every v_j is contained in exactly one V_i, by this we select exactly one proper ①–②–subtree from each of the ①–②–forests $F^{(j)}$. Moreover, since every V_i is a dominating set, every ①–node (and also every ②–node) is covered. But, observe that, by construction, the selected proper ①–②–subtrees $T_i^{(j)}$ may not be edge disjoint. Thus, it remains to choose proper ①–②–subtrees $\widehat{T}_i^{(j)}$ from the selected $T_i^{(j)}$ such that every ①–node of B is covered exclusively:

- for every ①–node b_i^j that is covered more than once, simply pick an arbitrary covering ①–②–subtree and remove b_i^j from all other covering ①–②–subtrees (together with the corresponding ②–node).

As the central nodes of the B_i are ②–nodes, it is guaranteed that each such subtree of the $T_i^{(j)}$ remains connected and proper. Observe that it may happen that a selected $T_i^{(j)}$ is erased up to the node a_i. This does not result in conflicts because this just indicates that the vertex v_j corresponding to $T_i^{(j)}$ is not really necessary for V_i to cover V. Recall that in Edge Disjoint ①–②–Subtree Cover, it is not necessary to choose a ①–②–tree from every forest; some forests may remain untouched. □

A solution for the instance of Domatic Number can be computed efficiently from the solution of an instance of Edge Disjoint ①–②–Subtree Cover and vice versa, and thus the proof of Theorem 5.9 is complete. □

Corollary 5.11 Steiner Tree 2–Spanner *is $\mathcal{NP}$-complete.*

5.3.2 Discussion of Special Cases

We now discuss how some special cases of Steiner Tree 2–Spanner can be solved polynomially. Observe that we have already shown that the problem is polynomially solvable if $G[R]$ is biconnected. Furthermore, if the number of Steiner vertices $|S|$ or the number of blocks of $G[R]$ is bounded by a constant, we can use dynamic programming or even exhaustive search to get polynomial algorithms.

Now consider the case that the block-graph $B(G)$ is a ①–②–star centered at a ②–node. In this case, it is sufficient to check if every block can be spanned by at least one Steiner vertex. Given an arbitrary selection of these Steiner vertices such that every block of G is spanned, we can simply combine the respective stars without getting conflicts (since the center of $B(G)$ corresponds to an articulation vertex). But, as we will see in the next subsection, even in this simple case, we cannot hope for finding a solution to the optimization problem (Minimum Steiner Tree 2–Spanner) that is anything better than a logarithmic factor of the optimal solution.

5.3.3 Hardness for $k = 3$

In this subsection, we consider the complexity of finding Steiner tree 3–spanners. In particular, we use the hardness of Steiner Tree 2–Spanner to show the $\mathcal{NP}$-hardness of Steiner Tree 3–Spanner.

Theorem 5.12 Steiner Tree 3–Spanner *is $\mathcal{NP}$-complete.*

Proof The membership of Steiner Tree 3–Spanner in $\mathcal{NP}$ is clear as before. Given an instance of Steiner Tree 2–Spanner (i.e., a graph G consisting of a set of terminals R and a set of Steiner vertices S), we construct the graph G_3 as follows:

- For every block of $G[R]$ that contains a tree 2–spanner, create a new fake Steiner vertex and connect it as in Subsection 5.2.2. This results in a graph G'.
- Let ℓ be the maximum of 5 and the maximal number of articulation vertices that are contained in a block of $G'[R]$. Replace in G' every block by an induced cycle of length 2ℓ, $C_{2\ell}$. By the choice of ℓ, it is guaranteed that the overall block structure of $G[R]$ can be retained, and the terminals can be arranged such that not two articulation vertices of one block are adjacent.
- For every Steiner vertex s of G' create a Steiner vertex in G_3.
- For every block B of $G'[R]$ and for every Steiner vertex s of G' do the following: If s is universal for B (i.e., B may be spanned in a Steiner tree 2–spanner by s), then connect the Steiner vertex corresponding to s in G_3 to all 2ℓ vertices of the cycle corresponding to B. No other Steiner edge is inserted to G_3.

As an example of the construction reconsider Figure 5.2(b) and (c), but now using $C_{2\ell}$'s instead of the C_4's for the terminal edges.

Remark 5.13

1. *Observe that by the last step of the construction, Steiner edges in G_3 always connect a Steiner vertex with a terminal.*

2. *Consider a terminal x in G_3 which is not an articulation vertex. If there is a Steiner edge $\{s, x\}$, then G_3 also contains Steiner edges connecting s to all other terminals of the unique block to which x belongs.*

By construction, it is clear that G_3 admits a Steiner tree 2–spanner if and only if G does. Furthermore, if G_3 admits a Steiner tree 2–spanner, then G_3 also admits a Steiner tree 3–spanner. It remains to show the following lemma:

Lemma 5.14 *If G_3 admits a Steiner tree 3–spanner, then G_3 also admits a Steiner tree 2–spanner.*

Proof Let T be a Steiner tree 3–spanner of G_3, and B be a block of $G_3[R]$. We construct a tree T' from T, which is a Steiner tree 2–spanner.

First observe that B does not contain a tree 3–spanner. The only possibility of covering a terminal edge $\{u, v\}$ of B is by including it into T or by using some Steiner edges. Thus, there are three basic cases to be considered:

1. $\{u, v\}$ is in T. Since by the previous observation not all edges of B are in T, there is at least one Steiner edge that connects another terminal of B, which is not an articulation vertex, to a Steiner vertex s. By Part 2 of the previous remark, also edges $\{u, s\}$ and $\{v, s\}$ are in G_3 and can thus be chosen for T'.

2. $\{u, v\}$ is covered by a path via one Steiner vertex. This path has length 2 and can be selected for T'.

3. $\{u, v\}$ is covered by a path $P = \{u, s, x, v\}$ via one Steiner vertex s and one terminal x distinct from u and v. Observe that x may belong to B, or, if v is an articulation vertex, to a distinct block B'. In the second case, by choice of cycle length 2ℓ, x is not an articulation vertex and thus belongs exclusively to block B'. In both cases, at least one of u or v is not an articulation vertex. Consequently, by Part 2 of Remark 5.13, all vertices of B (and B', respectively) are connected to s in $G[R]$, and we can hence choose all these edges for T'.

Observe that $\{u, v\}$ cannot be covered by a path via two Steiner vertices (by Part 1 of Remark 5.13) or by a path via two terminals (by construction of B). Altogether, since T is tree it follows that we can construct a tree T', which for each block of $G[R]$ is a star centered at a Steiner vertex, and which is thus a Steiner tree 2–spanner. □

By this, given a Steiner tree 3–spanner for G_3, we can construct a Steiner tree 2–spanner for G_3 in polynomial time, and hence G_3 admits a Steiner tree 3–spanner if and only if G admits a Steiner tree 3–spanner. □

Note that the construction generalizes to the case where $k \geq 4$ by choosing longer induced cycles as blocks for G_3. But as mentioned above, for $k \geq 4$, Steiner Tree k–Spanner is $\mathcal{NP}$-complete also by the hardness of Tree k–Spanner. This concludes the discussion of the decision problem for Steiner tree k–spanners.

5.4 The Optimization Problem

We now turn to the optimization problem Minimum Steiner Tree k–Spanner. As a consequence of Theorem 5.9 and Corollary 5.11, Minimum Steiner Tree 2–Spanner is also $\mathcal{NP}$-complete. Hence, even if we know in advance that the given instance admits some Steiner tree 2–spanner, we cannot hope for finding efficiently one that uses the minimum number of Steiner vertices.

We now strengthen this result by proving an inapproximability result. We do this for the general case of arbitrary $k \geq 2$. In particular, we show that it is even $\mathcal{NP}$-hard to find a Steiner tree k–spanner in which the number of Steiner vertices is guaranteed to be within anything better than a logarithmic factor of the number of Steiner vertices in an optimal solution.

Since we now consider arbitrary k, in this section, we cannot use the equivalent formulation as an edge disjoint subtree covering problem. Instead, we show the result directly by proving that finding an (approximate) minimum Steiner tree k–spanner is as hard as solving Minimum Hitting Set (Problem D.9 in Appendix D): Given a collection C of subsets of a finite set F, and a positive integer $K \leq |F|$, the problem is to decide whether there is a subset $F' \subseteq F$ with $|F'| \leq K$ such that F' contains at least one element from each subset in C.

As shown by Ausiello et al. (1980), this problem is equivalent to Minimum Set Cover (Problem D.8 in Appendix D). Using the inapproximability results of Feige (1996) for Minimum Set Cover, there is no $((1-\epsilon)\log|F|)$–approximation algorithm for Minimum Hitting Set for any $\epsilon > 0$, unless

$\mathcal{NP} \subset \mathcal{DTIME}(n^{\log\log n})$. Here, $\mathcal{DTIME}(f(n))$ denotes the class of decision problems that can be solved within $\mathcal{O}(f(n))$ time by a deterministic Turing machine, for any non-decreasing function $f : \mathbb{N} \to \mathbb{N}$. Note that $\mathcal{NP} \not\subset \mathcal{DTIME}(n^{\log\log n})$ means that $\mathcal{NP}$ does not have quasi-polynomial deterministic algorithms, and that this is a slightly weaker assumption than $\mathcal{P} \neq \mathcal{NP}$.

Theorem 5.15 *For any fixed integer $k \geq 2$,* Minimum Steiner Tree k–Spanner *cannot be approximated within ratio $(1-\epsilon)\log|S|$ for any $\epsilon > 0$, unless $\mathcal{NP} \subset \mathcal{DTIME}(n^{\log\log n})$.*

Proof We prove the hardness of Minimum Steiner Tree k–Spanner by showing that finding an (approximate) minimum Steiner tree k–spanner is as hard as solving Minimum Hitting Set. Starting from an instance (C, F) of Minimum Hitting Set where $C = \{C_1, \ldots, C_m\}$ and $F = \{f_1, \ldots, f_n\}$, the idea is as follows: A subset C_i corresponds to a block in G that does not contain a tree k–spanner (but a Steiner tree k–spanner), whereas the elements of F are modeled by Steiner vertices. We construct the graph G for Minimum Steiner Tree k–Spanner in the following way:

- Create terminals and terminal edges:
  For every $C_i \in C$ for $1 \leq i \leq m$, create a simple cycle R_i of length $k+2$ consisting of vertices $V(R_i) = \{r_i^c, r_i^1, \ldots, r_i^{k+1}\}$. Melt together all vertices r_i^c to form one central vertex r.

- Create Steiner vertices and edges:

  - For every $f_\ell \in F$ for $1 \leq \ell \leq n$, create a new Steiner vertex s_ℓ.
  - For every $f_\ell \in C_i$, connect s_ℓ to r and to r_i^j for $1 \leq j \leq k+1$.

We get the graph $G = (R \dot\cup S, E)$ with Steiner vertices $S = \{s_1, \ldots, s_n\}$ and terminals $R = \{r\} \cup \{r_i^j \mid 1 \leq i \leq m,\ 1 \leq j \leq k+1\}$. Observe that $|S| = |F|$. Figure 5.6 shows an example of the construction for $k = 2$.

For every $C_i \in C$, there is a block R_i in G, and the members of C_i represent Steiner vertices that may span R_i. Observe that $G[R]$ is connected, that the blocks of $G[R]$ are formed by the R_i for $1 \leq i \leq m$, and that the blocks R_i do not admit a tree k–spanner. Moreover, every block R_i admits a Steiner tree k–spanner.

It remains to show the equivalence of Minimum Hitting Set and Minimum Steiner Tree k–Spanner on the given instances.

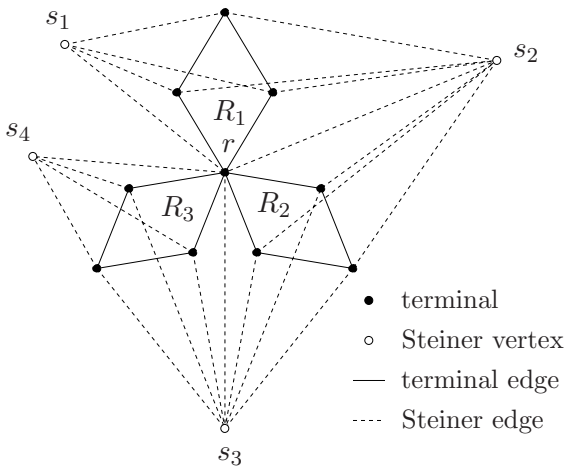

Figure 5.6: The graph for MINIMUM STEINER TREE 2–SPANNER from the instance (C, F) of MINIMUM HITTING SET where $C = \{\{1, 2\}, \{2, 3\}, \{3, 4\}\}$ and $F = \{1, 2, 3, 4\}$.

Lemma 5.16 *There is a hitting set of cardinality K for the instance (C, F) of* MINIMUM HITTING SET*, if and only if G as constructed above admits a Steiner tree k–spanner containing K Steiner vertices.*

Proof Let $F' = \{f'_1, \ldots, f'_K\}$ be a hitting set of (C, F). Then a Steiner tree k–spanner T of G can be constructed as follows: If a $C_i \in C$ is hit by more than one member of F', arbitrarily choose one of the hitting elements f', otherwise f' is unique. Take the corresponding Steiner edges $\{s', r_i^j\}$ for $1 \leq j \leq k + 1$ into T. To complete T, connect to r each Steiner vertex s' that has been chosen in the first step. It is easily seen that T is a Steiner tree k–spanner using K Steiner vertices.

For the other direction, let T be a Steiner tree k–spanner of G, and let T contain K Steiner vertices. By construction of G, T can consist only of Steiner edges. Taking all elements f' of F that correspond to Steiner vertices included in T, then yields a hitting set for (C, F) of cardinality K. □

Thus, given a solution for MINIMUM HITTING SET, we can construct the corresponding Steiner tree k–spanner in polynomial time, and vice versa. Now, an approximation algorithm within ratio $(1 - \epsilon) \log |S|$ for any $\epsilon > 0$ for MINIMUM STEINER TREE k–SPANNER would yield an approximation algorithm within ratio $(1 - \epsilon) \log |F|$ for MINIMUM HITTING SET. This contradicts the inapproximability results for MINIMUM HITTING SET, and the proof of Theorem 5.15 is complete. □

As a consequence of the previous theorem, we cannot hope for finding an approximation algorithm that has a constant approximation ratio.

Observe that the instance G of MINIMUM STEINER TREE k–SPANNER as constructed above has a block-graph $B(G)$ that is a ①–②–star centered at a ②–node. As stated in Section 5.3.2 above, in this special case at least STEINER TREE 2–SPANNER is solvable efficiently.

In the definition of MINIMUM STEINER TREE k–SPANNER, we have measured the quality of a Steiner tree k–spanner T in terms of the *number of Steiner vertices*. Let us now reconsider the optimization problem with respect to the *total number of edges*. As mentioned above, for any Steiner tree k–spanner T, $|E(T)| = |R| + |S(T)| - 1$, and hence a minimum Steiner tree k–spanner with respect to the number of Steiner vertices is also a minimum Steiner tree k–spanner with respect to the total number of edges. By this, the modified optimization problem clearly remains $\mathcal{NP}$-complete. But note that we cannot deduce similar results concerning the (in)approximability of this problem from our reduction from MINIMUM HITTING SET, because the size of the instance ($|R|$) is an additive term within the quality measure of T. Thus, for big instances, the approximation ratio may tend to 1.

5.5 Related Tree Covering Problems

Motivated by the subtree covering problem that has appeared in the context of STEINER TREE 2–SPANNER, we now examine different variations of this kind of covering problems. Observe however, that these new problems cannot be translated to the context of Steiner tree k–spanners. We deal with them here since we consider them interesting in themselves, in particular the tree covering problems at the end of this section.

Edge Disjoint ①–②–Tree Cover by ①–②–Trees

Instead of considering the possibility of choosing a proper subtree of a selected ①–②–subtree, we now examine the case where we are forced to take the whole selected ①–②–subtree. This results in the following problem:

Problem 5.17 EDGE DISJOINT ①–②–TREE COVER

Given: A ①–②–tree B and a set of ①–②–forests $\mathcal{F} = \{F^{(1)}, \ldots, F^{(n)}\}$ such that each $F^{(j)}$ consists of a set of node disjoint proper ①–②–subtrees $T_i^{(j)}$ of B.

Problem: Find a cover of the nodes of B consisting of an edge disjoint collection of $T_i^{(j)}$ such that every index j is chosen at most once.

Observe that Problem 5.7, Edge Disjoint ①–②–Subtree Cover, is not a direct generalization of this problem. The choice of whole ①–②–subtrees of the $F^{(j)}$ instead of picking parts thereof significantly changes the objective. Hence, the approaches of Theorem 5.9 do not apply here.

However, this variant also turns out to be hard. In particular, it is hard even for the simple case of ①–②–trees B that are ①–②–stars centered at a ②–node. Note again, that for this case the corresponding subtree cover problem is solvable.

Theorem 5.18 Edge Disjoint ①–②–Tree Cover *is $\mathcal{NP}$-complete.*

Proof The membership of Edge Disjoint ①–②–Tree Cover in $\mathcal{NP}$ is clear. We show the $\mathcal{NP}$-completeness of the problem by a reduction from the following $\mathcal{NP}$-complete One–in–Three Satisfiability Problem (Problem D.13 in Appendix D): As in 3–Satisfiability, we are given a set $U = \{x_1, \ldots, x_n\}$ of Boolean variables and a collection $C = \{c_1, \ldots, c_m\}$ of clauses over U such that every clause $c_i \in C$ has exactly three literals. The problem is to find a truth assignment for U such that every clause in C has *exactly one* true literal. Observe that One–in–Three Satisfiability remains $\mathcal{NP}$-complete for instances where no clause contains a negated literal.

The idea of the construction is as follows: The ①–②–tree B contains a ①–node for every variable and for every clause. Furthermore, the ①–②–subtrees are built in a way such that a variable 'covers' all the clauses in which it appears. Formally, given an instance (U, C) of One–in–Three Satisfiability, we construct the instance $(B, \mathcal{F})$ of Edge Disjoint ①–②–Tree Cover as follows:

- For the ①–②–tree B, create a central ②–node a. For every $x_j \in U$, create a ①–node b_j and connect it to a. For every $c_i \in C$, create a ①–node c_i and connect it to a. To every ①–node, add a new ②–node as a leaf.
- $\mathcal{F}$ consists of $2 \cdot |U|$ proper ①–②–forests $F^{(j)}$ and $\overline{F}^{(j)}$ for $1 \leq j \leq |U|$, corresponding to the literals x_j and $\overline{x}_j$, respectively:
  - For $1 \leq j \leq |U|$, $F^{(j)}$ contains the ②–node a, the ①–node b_j, and all the ①–nodes c_i that correspond to clauses c_i that contain x_j, together with all adjacent ②–nodes and all induced edges.
  - For $1 \leq j \leq |U|$, $\overline{F}^{(j)}$ only contains the ②–node a and the ①–node b_j together with the adjacent ②–node and the induced edges. (This is due to the fact that we consider only clauses that do no contain any negative literal.)

This gives a feasible instance $(B, \mathcal{F})$, which can be constructed in polynomial time. Moreover, all ①–nodes are covered by at least one of the proper ①–②–trees. Observe that each $F^{(j)}$ or $\overline{F}^{(j)}$ consists of exactly one tree.

Now, given a truth assignment such that every clause is fulfilled exactly once, we can construct an edge disjoint ①–②–tree cover of B by taking the tree $F^{(j)}$ if x_j is set to `true`, and the tree $\overline{F}^{(j)}$ otherwise.

To see the other direction, observe that $F^{(j)}$ and $\overline{F}^{(j)}$ cannot both belong (or both not belong) to an edge disjoint ①–②–tree cover of B. Accordingly, an edge disjoint ①–②–tree cover of B induces a truth assignment of U by setting x_j to `true` if $F^{(j)}$ is in the cover, and to `false` otherwise. By construction of $\mathcal{F}$, the ①–nodes c_i are only covered by true literals. Let Cov be an edge disjoint cover of B. Then all ①–nodes c_i are covered. Additionally, they cannot be covered twice because this would violate the edge disjointness of the ①–②–trees in Cov (consider edge $\{c_i, a\}$). Thus, this results in a truth assignment such that every clause is fulfilled exactly once. □

Note that the construction used in the previous proof does not work for Edge Disjoint ①–②–Subtree Cover, Problem 5.7: It would be possible to select parts of two ①–②–subtrees $F^{(i)}$ and $F^{(j)}$ such that these selected ①–②–subtrees are edge disjoint, but $F^{(i)}$ and $F^{(j)}$ are not. In the corresponding truth assignment, there is a clause that contains more then one true literal, which contradicts the constraint that every clause is fulfilled exactly once.

Trees instead of ①–②–Trees

In contrast to the problems considered so far, we now generalize to arbitrary trees, not ①–②–trees. Thus, there is no need to deal with *proper* subtrees.

Problem 5.19 Edge Disjoint Subtree Cover

Given: A tree B and a set of forests $\mathcal{F} = \{F^{(1)}, \ldots, F^{(n)}\}$, where each $F^{(j)}$ consists of a set of node disjoint trees $T_i^{(j)}$ of B.

Problem: Find a cover of the nodes of B consisting of edge disjoint *subtrees* $\widehat{T}^{(j)}$ of a $T_i^{(j)}$ such that every index j is chosen at most once.

Problem 5.20 Edge Disjoint Tree Cover

Given: A tree B and a set of forests $\mathcal{F} = \{F^{(1)}, \ldots, F^{(n)}\}$, where each $F^{(j)}$ consists of a set of node disjoint trees $T_i^{(j)}$ of B.

Problem: Find a cover of the nodes of B consisting of an edge disjoint collection of $T_i^{(j)}$ such that every index j is chosen at most once.

We get similar complexity results as for the case of ①–②–trees:

Theorem 5.21 EDGE DISJOINT SUBTREE COVER *and* EDGE DISJOINT TREE COVER *are* $\mathcal{NP}$*-complete.*

Proof We can use the same constructions and arguments as in Theorems 5.9 and 5.18. In the construction given there, the distinction of the different types of nodes and the restriction to proper subtrees does not affect the argumentation. □

5.6 Further Remarks

In this chapter, we have considered Steiner tree k–spanners, a new variant of tree k–spanners where we have relaxed on the vertices (and edges) that have to be covered in the k–spanner. In particular, we have shown that the decision problem for $k = 2$ is equivalent to a tree covering problem. Apart from showing the hardness of the decision as well as the optimization problem for all $k \geq 2$, we have also discussed some variants of the tree covering problem, which are interesting in their own.

Some open problems remain: In the discussion in Section 5.3.2, it would be interesting to extend the list of cases where STEINER TREE 2–SPANNER is solvable efficiently. Furthermore, as mentioned at the end of Section 5.4, we cannot deduce any inapproximability results for MINIMUM STEINER TREE k–SPANNER from our construction, when the quality of a Steiner tree k–spanner is measured with respect to the total number of edges.

Apart from that, another natural question arises within the context of the tree covering problems: What is the complexity status of the considered tree covering problems if we ask for node disjointness instead of edge disjointness? Here we cannot use the methods used above.

Chapter 6

Survivable Networks with Bounded Delay

In the previous chapters, we have considered different aspects of the problem of finding sparse, almost distance preserving *subgraphs* of a given graph. So we have *analyzed* a given graph. Now, we follow the opposite course: We investigate how graphs that exhibit certain properties can be *characterized* and *constructed* from scratch.

6.1 Reliable and Fault-Tolerant Networks

Survivable Networks

The main function of a network is to provide connectivity between the sites. In many cases it is crucial, that this is preserved even in the case of faults in either sites or links. Accordingly, a major concern in network design is fault-tolerance and reliability. That means in particular that the network to be constructed shall remain reliable even in the case of node or link faults.

According to the actual applications and requirements, the term 'reliability' may stand for different features. Following the lines of the previous chapters, our objective are bounded distances again: Our goal is to design networks such that distances between nodes remain small even in the case of faulty links or nodes. We call such networks *survivable*.

As the underlying model, we use again unweighted graphs, and measure the distance in a network in which faults have occurred by a shortest path in the subnetwork that is induced by the non-faulty components.

Self-Spanners

In this chapter, we introduce and characterize new classes of graphs that guarantee constant delay factors even when a multiple number of *edges* have failed. In a first step, we do not limit the number of edge faults at all, that is we allow for *unlimited* edge faults. The graph class that models this case is called *k–self-spanners*. Secondly, we examine the more realistic case where the number of edge faults is *bounded*. For this, we introduce the class of *(k, ℓ)–self-spanner* graphs.

In both cases, the name is motivated by strong relationships to the concept of *k–spanners*. By fixing the values k (and ℓ), we obtain specific new graph classes. Within this chapter, we are interested in characterizational as well as structural aspects of the new graph classes.

Related Concepts

In Chapter 4, we have already considered a notion of fault-tolerance and reliability. But there our goal was to find subgraphs with a certain structure in a given graph such that a constant distance guarantee can be given. There, we used independent subtrees as a model. In contrast to that, we now do not consider substructures; we concentrate on the construction of new networks, such that fault-tolerance and distance constraints can be guaranteed simultaneously.

Cicerone and Di Stefano (1998) have considered graphs that guarantee constant delay factors even when an *unlimited* number of *vertices* fail. The class of such graphs is called *k–bounded induced distance graphs*. Unfortunately, their results do not carry over to the dual case of *edge faults*.

A graph class that has strong constraints with respect to k–bounded induced distance graphs is for example the class of *distance-hereditary* graphs, i.e., graphs such that the distance between any two vertices in every connected induced subgraph is preserved (see also Appendix B or Brandstädt (1993)). Thus, in distance-hereditary graphs, the distance between any two vertices does not increase, even if an arbitrary number of vertices fail: distance-hereditary graphs are 1–bounded induced distance graphs.

Farley and Proskurowski (1993) study a similar concept: they give characterizations for graphs in which *no delay* occurs in the case that a *single* vertex fails. These graphs are called *self-repairing*.

A large amount of research has been dedicated to fault-tolerant network design, mainly starting from a given desired topology and introducing fault-tolerance to it. Basically, there are two main approaches for that: The first approach consists of techniques that add redundancy to the desired

architecture by introducing new network components. The goal here is to maintain the full performance of the desired topology while minimizing the cost of the redundant components (e.g., see Hayes (1976); Bruck et al. (1997); Sung et al. (1997)).

In the second approach, the fault-tolerance is achieved not by adding redundancy to the network, but by using the non-faulty part of the network to simulate the desired architecture. That means that the role of a site in the network may vary depending on the actual faults. For example in Leighton et al. (1992) and Cole et al. (1997), it is shown how arrays and several bounded-degree networks with faults can emulate non-faulty such networks. The general approach used there is that of finding an embedding of the faulty graph into the non-faulty graph subject to some quality measure. In this approach, the structure of the network is fixed, but the role of any vertex may vary depending on the actual fault. This incurs additional work for re-routing and re-structuring the network. See also, e.g., Annexstein (1991).

In our work, we do not start with a fixed target graph, nor do we allow a re-structuring of the graph; we keep the identification of each vertex fixed. Thus, we cannot use the framework of graph embedding here.

In this chapter, we first investigate networks with bounded delay and unlimited fault-tolerance, the class of k–self-spanners. The subsequent section then contains the main part of this subject and discusses the case of a limited number of possibly faulty edges, i.e., the class of (k, ℓ)–*self-spanners*.

6.2 Networks with Bounded Delay and Unlimited Fault-Tolerance

Cicerone and Di Stefano (1998) introduce a class of graphs that guarantees constant delays even in the case of an *unlimited* number of *vertex faults*. In this section, we follow the dual approach and examine graphs that guarantees constant delays even in the case of an *unlimited number of edge faults*.

After a formal definition, we first give equivalent, strict characterizations for this graph class and list major properties. As all these turn out to be non-algorithmic in general, we describe the structure of k–self-spanners for $k \leq 3$ in detail, and show that the problem of finding a minimum k such that a graph is a k–self-spanner is hard. As a further result, we show a strong relationship between networks with bounded delay and unlimited fault-tolerance with respect to edge and vertex faults.

6.2.1 Definitions and Basic Observations

Our goal is to model the following graph property: Given a graph in which an arbitrary number of edges have failed, the distance between two vertices in this faulty graph is at most k times the distance in the non-faulty graph, unless these vertices become disconnected by the edge faults. Let us first repeat the definition of k–bounded induced distance graphs from Cicerone and Di Stefano (1998), which models the dual case for vertex faults:

Definition 6.1 (k–bounded induced distance graph)
For any fixed real number $k \geq 1$, a graph $G = (V, E)$ is a k–bounded induced distance graph *if for every* connected induced *subgraph $G' = (V', E')$ of G:*

$$d_{G'}(u, v) \leq k \cdot d_G(u, v) \quad \text{for every } u, v \in V'.$$

Denote the class of all k–bounded induced distance graphs by $\mathrm{BID}(k)$.

Now, dealing with edge faults, we have to consider edge induced subgraphs. This results in the following definition:

Definition 6.2 (k–self-spanner)

1. *For any fixed real number $k \geq 1$, a graph $G = (V, E)$ is a* k–self-spanner *if for every subgraph $G' = (V, E')$ of G:*

   $(*)$ $d_{G'}(u, v) \leq k \cdot d_G(u, v)$ *for all $u, v \in V$ that are connected in G'.*

   Denote the class of all k–self-spanners by $\mathrm{SS}(k)$.
   The parameter k is called stretch factor.

2. *For a graph G, $minS(G)$ denotes the smallest k such that $G \in \mathrm{SS}(k)$.*

The name of the class is motivated by its strong relationship with the concept of k–spanners: all connected components C of G' are k–spanners of $G[V(C)]$. Thus a graph of this class 'spans itself'. Figure 6.1 gives examples for the self-spanner properties of two graphs. The graph G in (a) belongs to SS(3), but as $minS(G) = 3$, it does not belong to SS(2). If G' is achieved from G by adding the edge $\{u, v\}$ then $minS(G') = 6$, and thus G' does not belong to SS(3) anymore. The graph in (b) belongs to SS(4), but not to SS(3).

Observe that the definition works equally well for connected and disconnected graphs; but it is obvious as before that we can restrict our analysis to connected graphs in the following. Furthermore, in the faulty graph we only have to consider connected vertices since otherwise the definition of distance does not apply.

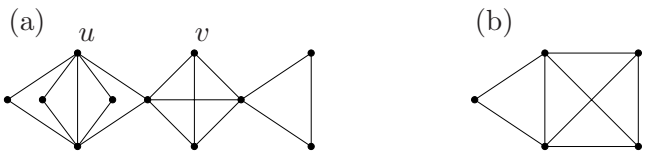

Figure 6.1: Examples for k–self-spanners.

Our first goal is to examine what is required to check whether a graph belongs to a class $\mathrm{SS}(k)$. As the following lemma indicates, we do not have to consider all possible subgraphs. In Part 2 of that lemma, we show that we can restrict ourselves to *spanning* (and also connected) subgraphs. A further important observation is stated in Part 3: The distance constraint has to be fulfilled within every connected (but not necessarily spanning) subgraph.

Lemma 6.3 *Let $G = (V, E)$. The following statements are equivalent:*

1. *$G \in \mathrm{SS}(k)$;*
2. *every* spanning *subgraph $G' = (V, E')$ of G is a k–spanner of G;*
3. *every* connected *subgraph $G' = (V', E')$ of G is a k–spanner of $G[V']$.*

Proof [1. $\Rightarrow$ 2.] is trivial. It remains to show [2. $\Rightarrow$ 3.] and [3. $\Rightarrow$ 1.]

[2. $\Rightarrow$ 3.] Assume that every *spanning* subgraph of G (which is surely connected since G is) is a k–spanner of G and there is a connected (not necessarily spanning) subgraph $G' = (V', E')$ of G such that $d_{G'}(u, v) > k \cdot d_G(u, v)$ for two vertices $u, v \in V'$. Expand G' to a connected spanning subgraph $G'' = (V, E'')$ by linking missing vertices of G to V' such that these vertices do not lie on a cycle (this is always possible because G is connected). Then, G'' is a spanning subgraph of G and $d_{G''}(u, v) > k \cdot d_G(u, v)$, a contradiction.

[3. $\Rightarrow$ 1.] Assume that every *connected* subgraph $G' = (V', E')$ of G is a k–spanner of $G[V']$. We can hence combine these subgraphs to form any arbitrary subgraph of G that fulfills $(*)$ in Definition 6.2. □

Observe that, as a consequence of Part 3 of the previous lemma, k–self-spanners do not only model networks in which a constant delay is guaranteed with respect to an unlimited number of *edge faults*, but also to *vertex faults*. Accordingly, k–self-spanners model the general case of vertex and edge faults.

The definition of k–self-spanners as given in Definition 6.2 stems from the point of view of network design. Observe, however, that we could have equally

well chosen Part 2 of Lemma 6.3 for the definition. The characterization of $\mathrm{SS}(k)$ is then motivated from a graph theoretical point of view.

Remark 6.4 *Since we are only dealing with unweighted graphs and we directly use the notion of k–spanners for the characterization of the class $\mathrm{SS}(k)$ in Lemma 6.3, it is clear that $G \in \mathrm{SS}(k)$ if and only if $G \in \mathrm{SS}(\lfloor k \rfloor)$ for all real $k \geq 1$. Thus in the following we only consider integer values for k.*

Note that the previous remark does *not* hold for k–bounded induced distance graphs. Here real values for k really do matter. It is easy to see that the following straightforward properties hold:

Lemma 6.5

1. *Let $G = (V, E)$. Then $G \in \mathrm{SS}(k)$ for all $k \geq |V| - 1$.*
2. *If $1 \leq k \leq k'$ then $\mathrm{SS}(k) \subseteq \mathrm{SS}(k')$.*

This induces the following natural recognition problem for $\mathrm{SS}(k)$:

Problem 6.6 Minimum Self-Spanner

Given: A graph G and an integer $k \geq 1$.

Problem: Does G belong to $\mathrm{SS}(k)$, i.e., $minS(G) \leq k$?

6.2.2 Characterization of k–Self-Spanners

In order to find a solution for Minimum Self-Spanner, in this subsection, we identify some characterizing properties for the class of k–self-spanners in terms of longest simple paths and cycles. On the one hand, this leads to strict (and efficient) characterizations for $\mathrm{SS}(k)$ for small k. On the other hand, these results imply that Minimum Self-Spanner is hard in general. We also show a nice relationship between the classes of k–self-spanners and k–bounded induced distance graphs.

We start by giving straightforward characterizations of k–self-spanners.

Lemma 6.7 *Let $G = (V, E)$. The following propositions are equivalent:*

1. *$G \in \mathrm{SS}(k)$;*
2. *every simple cycle of G has at most $k + 1$ edges;*
3. *for every edge $e = \{u, v\} \in E$, a longest simple path ($\neq \{e\}$) between u and v in G has length at most k.*

Proof [2. $\Rightarrow$ 3.] is trivial. It remains to show [1. $\Rightarrow$ 2.] and [3. $\Rightarrow$ 1.]

[1. $\Rightarrow$ 2.] By contradiction, let us assume that there exists a simple cycle C in G with at least $k+2$ edges. Let $\{u, v\}$ be an edge of C, and let G' be the subgraph of G induced by the edges of C except $\{u, v\}$. Hence, $d_{G'}(u, v) \geq k+1$. This inequality implies that G' is not a k–spanner of $G[V(G')]$, a contradiction to Part 3 of Lemma 6.3.

[3. $\Rightarrow$ 1.] By contradiction, let us assume that $G \notin \mathrm{SS}(k)$. By Part 3 of Lemma 6.3, there exists a connected subgraph $G' = (V', E')$ of G such that G' is not a k–spanner of $G[V']$. By Lemma 2.2, there exists an edge $e = \{u, v\}$ in $G[V']$ that does not belong to E' such that $d_{G'}(u, v) > k$. This results in a simple path of length at least $k+1$, a contradiction. □

As a consequence of the previous theorem, the class of k–self-spanners is closed under subgraphs:

Corollary 6.8 *If G is in $\mathrm{SS}(k)$ for some fixed k then also every subgraph of G is in $\mathrm{SS}(k)$.*

Proof This is an immediate consequence of Part 3 of Lemma 6.7: The length of a longest simple path in G cannot increase in subgraphs of G. □

Together with Part 2 of Lemma 6.3, the previous corollary shows that in particular the *spanning trees* play an important role for the determination of the stretch factor of a graph, and thus for the characterization of a graph in terms of $\mathrm{SS}(k)$.

As a further consequence of Lemma 6.7, it is possible to describe the graphs that belong to $\mathrm{SS}(k)$ for small stretch factors k as in the following lemma. Recall that a *diamond* is a biconnected graph formed by two possibly adjacent vertices u and v, which are connected by $K \geq 2$ disjoint paths of length 2 (see for example the leftmost block in Figure 6.1(a)).

Lemma 6.9 *Let G be a graph. Then following characterizations hold:*

1. *$G \in \mathrm{SS}(1)$ if and only if every block is a K_2;*
2. *$G \in \mathrm{SS}(2)$ if and only if every block is a K_3 or K_2;*
3. *$G \in \mathrm{SS}(3)$ if and only if every block is a diamond, K_4, K_3, or K_2.*

Proof

1. The characterization of SS(1) follows from the definition.

2. Observe that $minS(K_3) = 2$. On the other hand, any biconnected graph G with at least four vertices has a stretch factor $minS(G) \geq 3$.

3. It is easy to see that $minS(K_4) = 3$, and $minS(D) = 3$ for any diamond D. For the other direction, note that every biconnected subgraph of a diamond is either a K_2, K_3, or another diamond. Moreover, the extension of a diamond to another biconnected graph by adding vertices or edges yields either a diamond or a graph G' with a stretch factor $minS(G') \geq 4$. The situation for a K_4 is similar. □

By Part 1 of the previous lemma, SS(1) is the set of all trees, or if we consider also disconnected graphs, the set of all forests.

Although the recognition problem for the class SS(k) is polynomially solvable for small fixed values of k (as in case $k \leq 3$), we show that the problem is hard for the general case. We do this by showing that the complementary problem of MINIMUM SELF-SPANNER is $\mathcal{NP}$-complete. The key observation comes from Part 2 of Lemma 6.7: In order to find the minimal stretch factor $minS(G)$ of a graph G, we have to examine the longest simple cycles of G.

Theorem 6.10 MINIMUM SELF-SPANNER *is co-$\mathcal{NP}$-complete.*

Proof Consider the following $\mathcal{NP}$-complete LONGEST CIRCUIT Problem (Problem D.6 in Appendix D): Given a graph $G = (V, E)$ and a positive integer $K \leq |V|$, is there a simple cycle in G of length K or more? By Part 2 of Lemma 6.7 this is exactly the complementary problem of MINIMUM SELF-SPANNER, and hence MINIMUM SELF-SPANNER is co-$\mathcal{NP}$-complete. □

Observe that Lemmas 6.7 and 6.9 show that we have to pay for a large stretch factor k, if we ask for a class SS(k) that contains non-trivial networks. This fact is due to the strong constraint for the fault-tolerance that we have used in the definition of k–self-spanners: A k–self-spanner has to guarantee for fixed bounded stretch factor even in case of an *unlimited* number of edge faults. In the light of applicability, this assumption is overly pessimistic; usually a *limited* number of edge faults is sufficient. Thus, the model of (k, ℓ)–self-spanners as treated in the following Section 6.3 is much more realistic.

But before we turn on to these, we establish a strong relationship between the classes of k–self-spanners and k–bounded induced distance graphs:

Lemma 6.11 *For all integers $k \geq 3$, $\mathrm{SS}(k) \subseteq \mathrm{BID}(\frac{k-1}{2})$.*

Proof If a graph G belongs to $\mathrm{SS}(k)$ then, by Part 2 of Lemma 6.7, the longest simple cycle of G has at most $k+1$ edges. In Cicerone and Di Stefano (1998), it has been shown that if a longest simple cycle of G has at most $k+1$ edges, then the smallest k' such that G is in $\mathrm{BID}(k')$ is at most $(k-1)/2$. □

Note that $\mathrm{BID}(1)$ is the well-known class of *distance-hereditary graphs*: A graph G is distance-hereditary if for every connected induced subgraph G' of G the following holds: $d_{G'}(u,v) = d_G(u,v)$, for all $u, v \in G'$. See for example Bandelt and Mulder (1986); Howorka (1977) for surveys. For the case of k–self-spanners, Lemma 6.9 shows that the classes of k–self-spanners for small stretch factors contain only distance-hereditary graphs. On the other hand, for all $k \geq 3$, we can find a distance-hereditary graph that contains a longest cycle with k edges. Consequently, the class of k–self-spanner contains distance-hereditary graphs for any $k \geq 1$, and we cannot characterize the distance-hereditary graphs in terms of $\mathrm{SS}(k)$. But, we will see in Section 6.3.3 that these graphs have strong self-spanner properties when only a limited number of edge faults is allowed.

6.3 Networks with Bounded Delay and Limited Fault-Tolerance

In this section, we follow the line of the previous section, but we consider limited fault-tolerance. That means that we introduce a class of graphs in which the stretch factor is bounded by the constant k when at most ℓ edge faults occur. Our first main results in Subsection 6.3.2 concern the problems of deciding whether a given graph is a (k,ℓ)–self-spanner: The problem is $\mathcal{NP}$-complete for the general case where both k and ℓ are part of the input, and remains $\mathcal{NP}$-complete if $k \geq 5$ is fixed. However, if $k \leq 3$ is fixed, or if $\ell \geq 0$ is fixed, then there are polynomial time algorithms.

Although the general characterization problem for (k,ℓ)–self-spanners is hard, in Subsection 6.3.3, we show that some well-known graph classes such as *distance-hereditary graphs* and *chordal graphs* exhibit strong self-spanner properties, by providing upper bounds on the stretch factor / fault-tolerance value trade off. Furthermore, we investigate how popular graph operations like *Cartesian product* or *split decomposition* can be used to construct new graphs with certain self-spanner properties.

Finally, the last part of this section shows how the new graph classes of (k,ℓ)–self-spanners fit into the context of some popular network topolo-

gies. We show for example that mesh-like topologies such as *grids*, *tori*, and *hypercubes* exhibit strong self-spanner properties in particular for small fault-tolerance values. Bounded-degree approximations of the hypercube such as *butterflies* and *cube-connected cycles*, however, result in big stretch factors even in the case of small fault-tolerance values.

6.3.1 Definitions and Basic Observations

Let us start by giving a formal definition of the new graph class. We are interested in graphs that exhibit the following property: If at most ℓ edges in a graph G fail, then for all pairs of vertices that *remain connected*, the distance constraint $d_{G'}(u,v) \leq k \cdot d_G(u,v)$ is fulfilled, where G' is the graph arising from G by deleting the faulty edges. Hence, as opposed to the previous section, the number of faulty edges is limited to a fixed number ℓ. This leads to the following definition (cf. Definition 6.2):

Definition 6.12 ((k,ℓ)–self-spanner)

1. *For any fixed real $k \geq 1$ and fixed integer $\ell \geq 0$, a graph $G = (V,E)$ is a* (k,ℓ)–self-spanner *if for every subgraph $G' = (V,E')$ of G with $|E'| \geq |E| - \ell$ and, $E' \subseteq E$:*

   $$d_{G'}(u,v) \leq k \cdot d_G(u,v) \quad \text{for all } u,v \in V \text{ that are connected in } G'.$$

   Denote the class of all (k,ℓ)–self-spanners by $\mathrm{SS}(k,\ell)$.
   The parameter k is called stretch factor, *and the parameter ℓ is called* fault-tolerance value *of the class* $\mathrm{SS}(k,\ell)$.

2. *For a graph G, $minS_\ell(G)$ denotes the smallest k such that $G \in \mathrm{SS}(k,\ell)$ (i.e., ℓ is fixed), whereas $maxT_k(G)$ denotes the largest ℓ such that $G \in \mathrm{SS}(k,\ell)$ (i.e., k is fixed).*

Observe that we again do not care for cases where two vertices are separated by the fault of edges because then the definition of distance does not apply; see also Remark 6.13 below. For examples consider again Figure 6.1, and let G be the graph in (a). Then, $minS_1(G) = 2$, $minS_2(G) = 3$, $maxT_2(G) = 1$, and $maxT_3(G) = 2$. Thus, G is in $\mathrm{SS}(2,1)$ and in $\mathrm{SS}(3,2)$, but not in $\mathrm{SS}(2,2)$. The 'opaque cube' C as shown in Figure 6.2 has $minS_1(C) = 3$ and $maxT_3(C) = 1$. Thus, G belongs to $\mathrm{SS}(3,1)$ but not to $\mathrm{SS}(3,2)$.

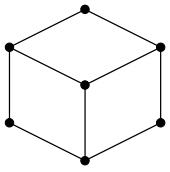

Figure 6.2: Opaque cube.

Remark 6.13 *Note that the definition of (k,ℓ)–self-spanners does* not *imply that G is $(\ell+1)$–edge-connected. As stated above, we do not care for pairs of vertices (or edges) that are separated by the edge faults. If we want to take this into account (e.g., to achieve 'true' fault-tolerance, such that we can always guarantee for a bounded connection between any pair of vertices even in the case of ℓ edge faults), we can restrict our attention to graphs belonging to the intersection of the classes of $(\ell+1)$–edge-connected graphs and (k,ℓ)–self-spanners.*

Clearly, if $G \in \mathrm{SS}(k)$ for some k, then $G \in \mathrm{SS}(k,\ell)$ for any ℓ. The remarks concerning the choice of name, non-integer stretch factors, and disconnected graphs apply as in the case of k–self-spanners. Hence, in the following, we restrict ourselves to *positive integer* values for k and to *connected* graphs.

Remark 6.14 *By similar arguments as in Lemma 2.2, it suffices to consider only* faulty edges *of every subgraph if we want to check whether a given graph belongs to a class* $\mathrm{SS}(k,\ell)$*: i.e., if $G'=(V,E')$ is a subgraph of $G=(V,E)$ with $|E'| \geq |E|-\ell$ and $E' \subseteq E$, we have to check if*

$$(**) \qquad d_{G'}(u,v) \leq k \text{ for every } e=\{u,v\} \in E \backslash E'.$$

We can furthermore simplify the procedure to check whether a graph belongs to a class $\mathrm{SS}(k,\ell)$: we do not have to consider all (possibly disconnected) subgraphs but only connected subgraphs. We get the following lemma:

Lemma 6.15 *For fixed integers $k \geq 1$ and $\ell \geq 0$, $G \in \mathrm{SS}(k,\ell)$ if and only if every* connected *and* spanning *subgraph $G'=(V,E')$ with $|E'| \geq |E|-\ell$ and $E' \subseteq E$ is a k–spanner of G.*

Proof It suffices to show the 'if'-part: Suppose every connected spanning subgraph $G'=(V,E')$ with $|E'| \geq |E|-\ell$ and $E' \subseteq E$ is a k–spanner of G, and, by contradiction, assume that G is not a (k,ℓ)–self-spanner. By definition, there is a subgraph $G''=(V,E'')$ with $|E''| \geq |E|-\ell$ and $E'' \subseteq E$ (not necessarily connected) such that there is a pair of vertices u and v (within one connected component of G'') and $d_{G''}(u,v) > k \cdot d_G(u,v)$.

Since G is connected, there is also a connected subgraph $\widetilde{G}=(V,\widetilde{E})$ with $E'' \subset \widetilde{E} \subseteq E$ (and thus $|\widetilde{E}| \geq |E|-\ell$) constructed as follows: Let $\mathcal{C}$ be the set of connected components of G''. Obtain $\widetilde{G}$ from G'' by adding $|\mathcal{C}|-1$ bridge edges such that $\widetilde{G}$ is minimally connected. Then $d_{\widetilde{G}}(u,v) > k \cdot d_G(u,v)$ and hence $\widetilde{G}$ is not a k–spanner of G, a contradiction. □

Note that, as opposed to k–self-spanners, here we cannot directly incorporate vertex faults. Consider for example again the 'opaque cube' as shown in Figure 6.2. As stated above, this graph is in $\mathrm{SS}(3,1)$, but the graph G' obtained from removing the internal vertex is not (in fact, it has a stretch factor $minS_1(G) = 5$, and thus is in $\mathrm{SS}(5,1)$). Hence, in the case of $\mathrm{SS}(k,\ell)$, we purely model edge faults. In the sequel, we use Lemma 6.15 as a characterization for the class of (k,ℓ)–self-spanners.

6.3.2 Characterization of (k,ℓ)–Self-Spanners

As mentioned before, we are interested in characterizational as well as structural aspects of the class of (k,ℓ)–self-spanners. We now consider the problem of recognizing graphs that belong to a given class and investigate characterization problems where we are interested in finding the optimal stretch factor or fault-tolerance value of a given graph. For this aim, we start by stating some (more or less) straightforward results on (k,ℓ)–self-spanners and define the problems to be considered formally. As our main results, we establish an almost complete set of complexity results for these problems.

Trivial Cases and Straightforward Results

Let us first consider some lower and upper bounds for k and ℓ:

Lemma 6.16

1. *No delay, i.e., $k = 1$:*
   For all $\ell > 0$, $\mathrm{SS}(1,\ell)$ is the set of all trees.
   Thus, $\mathrm{SS}(1)$ and $\mathrm{SS}(1,\ell)$ describe the same class of graphs.

2. *No edge fault, i.e., $\ell = 0$:*
   For all $k \geq 1$, $\mathrm{SS}(k,0) = \mathrm{SS}(1,0)$ is the set of all graphs.

3. *Weak delay constraints, i.e., large stretch factors:*
   G is in $\mathrm{SS}(k,\ell)$ for all $k \geq |V|-1$ for any $\ell \geq 0$.

4. *Strong fault-tolerance constraints, i.e., large fault-tolerance values:*
   If G belongs to $\mathrm{SS}(k,|E|-|V|+1)$ for some $k \geq 1$, then G also belongs to $\mathrm{SS}(k,\ell)$ for all $\ell > |E|-|V|+1$. Consequently, if G belongs to $\mathrm{SS}(k,|E|-|V|+1)$ then G also belongs to $\mathrm{SS}(k)$.

Proof Parts 1 and 2 are straightforward. To see Part 3, observe that if we allow for a stretch factor of $k \geq |V|-1$ then we do not really impose a distance constraint: any vertex of G may be used for a covering path.

It remains to prove Part 4: Suppose that G is a (k, ℓ)–self-spanner for $\ell = |E| - |V| + 1$. That means that, in particular, every spanning tree of G is a k–spanner of G. If more than $|E| - |V| + 1$ edges fail then the resulting subgraph is necessarily disconnected and hence (by Lemma 6.15) no further constraints are imposed. □

Hence in the following, given a graph G, we consider only stretch factors of $2 \leq k \leq |V| - 2$ and fault-tolerance values of $1 \leq \ell \leq |E| - |V| + 1$. The cases for small and large stretch factors and small fault-tolerance values can be considered trivial, whereas the case of large fault-tolerance values coincides with the case of k–self-spanners as discussed in Section 6.2.

As a consequence of the last lemma, it is clear that for every connected graph G there are some parameters k and ℓ such that G belongs to $\mathrm{SS}(k, \ell)$. Analogously, if we fix one of the parameters we can always find a feasible value for the other parameter. Furthermore, it is easy to see that (k, ℓ)–self-spanners have inductive properties with respect to the parameters as stated below.

Lemma 6.17

1. *If* $k \leq k'$ *then* $\mathrm{SS}(k, \ell) \subseteq \mathrm{SS}(k', \ell)$.

2. *If* $\ell \leq \ell'$ *then* $\mathrm{SS}(k, \ell') \subseteq \mathrm{SS}(k, \ell)$.

Remark 6.18 *In contrast to k–self-spanners (Corollary 6.8), the class of (k, ℓ)–self-spanners is* not *closed under subgraphs (cf. Figure 6.2 and the remark behind Lemma 6.15). Also it is not closed under supergraphs in the following sense: If a graph G is in $\mathrm{SS}(k, \ell)$ for some fixed parameters k and ℓ then there may be a supergraph of G on the same vertex set (i.e., a graph with additional edges) that does* not *belong to $\mathrm{SS}(k, \ell)$. The same remains true if we consider only $(\ell + 1)$–edge-connected graphs.*

As a consequence of the previous remark, the self-spanner properties of a graph cannot be inferred directly from the self-spanner properties of sub- or supergraphs.

For examples of standard graphs that exhibit some particular self-spanner properties, it is easy to see that $P_n \in \mathrm{SS}(1, \ell)$ for any $\ell \geq 1$ because P_n is a tree. Furthermore $C_n \in \mathrm{SS}(n-1, \ell)$ but $C_n \notin \mathrm{SS}(n-2, \ell)$ for any $\ell \geq 1$, since $minS_\ell(C_n) = n - 1$ for any $\ell \geq 1$ (i.e., the fault of one edge results in a path of length $n - 1$).

Characterizational Problems

Starting from the above observations, we are interested in finding non-trivial parameters such that a graph is a (k, ℓ)–self-spanner. This includes the problem of deciding for given parameters k and ℓ whether a given graph belongs to $\mathrm{SS}(k, \ell)$ as well as the more general recognition problems where we fix one of the parameters and try to optimize the other. This brings up the following optimization and characterization problems.

Problem 6.19 Minimum ℓ–Stretch-Factor

Given: A graph G and an integer $k \geq 1$.

Problem: Does G belong to $\mathrm{SS}(k, \ell)$, i.e., $minS_\ell(G) \leq k$?

Problem 6.20 Maximum k–Fault-Tolerance

Given: A graph G and an integer $\ell \geq 0$.

Problem: Does G belong to $\mathrm{SS}(k, \ell)$, i.e., $maxT_k(G) \geq \ell$?

Problem 6.21 General Self-Spanner

Given: A graph G and two integers $k \geq 2, \ell \geq 1$.

Problem: Does G belong to $\mathrm{SS}(k, \ell)$?

Thus, in Minimum ℓ–Stretch-Factor we consider ℓ as a fixed parameter, whereas in Maximum k–Fault-Tolerance k is a fixed parameter.

Complexity Results

We now turn to analyzing the complexity of the problems mentioned above. Let us first consider the special case where we allow for single edge faults only, i.e., $\ell = 1$. Recall that a bridge is an edge whose deletion disconnects the graph. An ℓ–edge-connected graph does not contain a bridge if $\ell \geq 2$.

Lemma 6.22 *$G \in \mathrm{SS}(k, 1)$ if and only if every edge of G is either a bridge or belongs to an induced cycle of length at most $k + 1$.*

Proof For the 'if'-part, let e be an arbitrary edge of G and consider $G' = G - e$. We have to show property $(**)$ of Remark 6.14. If e is a bridge in G then G' is disconnected and there is nothing to show. If e belongs to an induced cycle of length at most $k + 1$ then G' remains connected and by assumption G' is a k–spanner of G.

We show the opposite direction by contradiction. Assume $G \in \mathrm{SS}(k,1)$, and there is an edge $e = \{u,v\}$ that is not a bridge and that does not belong to an induced cycle of length at most $k+1$. Consider $G' = G - e$. Then G' is connected and $d_{G'}(u,v) > k$, a contradiction. □

Considering multiple edge faults, it is clear that bridges again do not contribute to the stretch factor. But unfortunately we cannot extend the characterization in a straightforward way. If we restrict ourselves to $(\ell+1)$–edge-connected graphs we get the following lemma:

Lemma 6.23 *Let $G = (V,E)$ be $(\ell+1)$–edge-connected. Then $G \in \mathrm{SS}(k,\ell)$ if and only if for every edge $e = \{u,v\}$ of G there are at least ℓ edge disjoint paths (not involving e) of length at most k connecting u and v.*

Proof For the 'if'-part, let $G' = (V,E')$ be a subgraph with $E' \subseteq E$ and $|E'| \geq |E| - \ell$, and let $e = \{u,v\}$ be an edge that does not belong to E'. Assume that there are ℓ edge disjoint paths (not involving e) of length at most k connecting u and v. Thus, even if the remaining $\ell - 1$ edge faults happen to appear in one of these paths each, at least one covering path for e in G' remains.

We show the opposite direction by contradiction: Assume $G \in \mathrm{SS}(k,\ell)$, and there is an edge $e = \{u,v\}$ such that there are at most $j < \ell$ edge disjoint paths (not involving e) of length at most k connecting u and v. Since j is maximal, there are j edges within the edge disjoint paths such that the following holds: the subgraph G' constructed by deleting e and these selected edges remains connected (since G is $(\ell+1)$–edge-connected) but $d_{G'}(u,v) > k$, a contradiction to $G \in \mathrm{SS}(k,\ell)$. □

Observe that we cannot relax on the edge–connectivity constraint in this lemma. Consider for example the diamond consisting of a C_4 and one chord: this graph is 2–edge-connected and belongs to $\mathrm{SS}(3,2)$, but it does not fulfill the constraints of Lemma 6.23.

Now, if we fix the fault-tolerance value ℓ, we can determine the smallest stretch factor of a given graph in polynomial time:

Theorem 6.24 Minimum ℓ–Stretch-Factor *is in $\mathcal{P}$ for all $\ell \geq 0$.*

Proof Let $\mathcal{G} = \{G' = (V,E') \mid |E'| \geq |E| - \ell\}$ be the set of all subgraphs of the given graph $G = (V,E)$ in which we have removed at most ℓ edges. Then a straightforward (brute-force, rather naive) algorithm to solve Minimum ℓ–Stretch-Factor is as follows:

For any $G' \in \mathcal{G}'$, and for any edge $e = \{u, v\} \in E$, run a shortest path algorithm on G' for $\{u, v\}$ and determine $stretch(G', e) := d_{G'}(u, v)$ if u and v remain connected in G'. Otherwise let $stretch(G', e) := 0$. Then $minS_\ell(G) = \max_{G'} \max_e \{stretch(G', e)\}$.

This algorithm can be implemented in polynomial time since

$$|\mathcal{G}| = \sum_{i=1}^{\ell} \binom{|E|}{i} \leq \sum_{i=1}^{\ell} |E|^i \leq \ell \cdot |E|^\ell \leq |V|^{2(\ell+1)}.$$

The last inequality holds since by Part 4 of Lemma 6.16, we may assume that $\ell \leq |V|^2$. Additionally, for every element of $\mathcal{G}$, we have to repeat a polynomial time shortest-path algorithm at most $|V|^2$ times. □

Observe that the running time is bounded by a polynomial in $|V|$ that contains ℓ as an exponent; but here we consider ℓ fixed. Surely, this algorithm and its analysis are rather naive and the running time may be lowered significantly. But here, we do not elaborate on that. As a consequence of the previous theorem, we also have:

Corollary 6.25 *The problem of deciding whether a graph is a (k, ℓ)–self-spanner for fixed $k \geq 1$ and $\ell \geq 0$ is in $\mathcal{P}$.*

If we consider the dual problem where we fix the stretch factor and we want to find the largest fault-tolerance value of a given graph, the situation is different:

Theorem 6.26

1. Maximum k–Fault-Tolerance *is $\mathcal{NP}$-complete for all $k \geq 5$.*
2. Maximum k–Fault-Tolerance *is in $\mathcal{P}$ for $k = 1, 2, 3$.*
3. General Self-Spanner *is $\mathcal{NP}$-complete.*

Proof As stated in Lemma 6.23, to solve Maximum k–Fault-Tolerance, it is crucial to find edge disjoint paths between any two vertices such that each path has bounded length. More formally, we have to solve the following problem: Given a graph G, two vertices s and t, and positive integers $K, L \leq n$, we have to decide whether G contains L or more mutually edge disjoint paths from s to t, which all have length at most K.

The problem described above is known as Maximum Length-Bounded Disjoint Paths (Problem D.7 in Appendix D). As shown in Itai et al. (1982), it is $\mathcal{NP}$-complete for all fixed $K \geq 5$; it is polynomially solvable for

$K \leq 3$, and it is open for $K = 4$. The proofs given there work for $(\ell+1)$–edge-connected graphs as well. Together with the observation that we can decide in polynomial time whether a given graph is $(\ell+1)$–edge-connected, Parts 1 and 2 of the theorem are proved. Part 3 is a direct corollary. □

Only MAXIMUM 4–FAULT-TOLERANCE remains to be settled. Observe that it does not suffice to look for a maximum number of edge disjoint paths from s to t under *no length constraint*. This problem is solvable in polynomial time. But in our case, the distance guarantee for every path is crucial.

Some Sufficient Conditions

We now consider some sufficient conditions that guarantee that a given graph is a (k,ℓ)–self-spanner for some k and ℓ. The main idea here is the following: If a graph contains a long cycle that has only few chords, then this graph is likely to have bad self-spanner properties. In other words, if we can guarantee that a graph does not contain such a long cycle with only few chords, then the self-spanner properties are good. This fact is expressed in the following lemma. In the sequel, we denote by $\mathscr{C}_n$ a cycle on n vertices that may contain an arbitrary number of chords (in contrast to C_n denoting an *induced* cycle).

Lemma 6.27 *Given a graph $G = (V,E)$ and two fixed positive integers k and ℓ, let $\mathscr{C}_n$ be a subgraph of G that is a cycle with at most $\ell-1$ chords and that has maximum length among all such cycles. If $n \leq k+1$ then G belongs to* $\mathrm{SS}(k,\ell)$.

Proof By contradiction, suppose that $n \leq k+1$ and $G \notin \mathrm{SS}(k,\ell)$. By Lemma 6.15, there exists a subgraph $G' = (V,E')$ of G with $|E'| \geq |E| - \ell$ such that G' is not a k–spanner of G. By Lemma 2.2, this implies that there exists an edge $e = \{u,v\} \in E \setminus E'$ such that $d_{G'}(u,v) > k$. The path P giving the distance $d_{G'}(u,v)$ together with edge e forms a cycle $\mathscr{C}_{n'}$ of G. Since P is obtained from G by removing e and at most $\ell-1$ other edges of E, then $n' > k+1$ and $\mathscr{C}_{n'}$ contains at most $\ell-1$ chords. This is a contradiction to $\mathscr{C}_n$ being a maximum cycle of G that has most $\ell-1$ chords. □

We call a condition as given in the previous lemma a *cycle-chord condition*. Observe that this lemma does not provide a strict characterization for the class $\mathrm{SS}(k,\ell)$: There are (k,ℓ)–self-spanners that do not fulfill the cycle-chord condition. As an example consider the graph G shown in Figure 6.3. Even if two arbitrary edges fail, any two vertices that are adjacent in G either become disconnected, or there remains a path of length at most 6 connecting them. Thus, $G \in \mathrm{SS}(6,2)$. On the other hand, the external cycle has eight

vertices and only one chord, and hence the cycle-chord condition of the previous lemma is *not* fulfilled.

We can extract some further cycle-chord condition from Lemma 6.27 resulting in an upper bound on the stretch factor / fault-tolerance value trade-off:

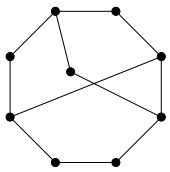

Figure 6.3: A graph of $\mathrm{SS}(6,2)$ for which the cycle-chord condition of Lemma 6.27 does not hold.

Corollary 6.28 *Let $G = (V, E)$ be a graph, $n \geq 4$ an integer, and $f : \mathbb{N} \to \mathbb{N}$ a monotone increasing function. If every cycle of G on n vertices has at least $f(n)$ chords, then G belongs to* $\mathrm{SS}(n, f(n+2))$.

Proof If every cycle of G on n vertices has at least $f(n)$ chords, then, by monotonicity of f, also every cycle on n or more vertices has at least $f(n)$ chords. Let $\mathscr{C}_{n'}$ be a cycle of maximum length that has at most $f(n) - 1$ chords. Then $n' \leq n - 1$. By Lemma 6.27, G belongs to $\mathrm{SS}(n-2, f(n))$, and hence also to $\mathrm{SS}(n, f(n+2))$. □

The cycle-chord conditions also support the intuition that graphs in which every vertex has a large degree are likely to have good self-spanner properties. In the next subsection, we use the previous corollary to investigate the self-spanner properties of well known classes of graphs for which cycle-chord conditions are known.

6.3.3 Self-Spanner Properties of Certain Graphs

Although the general characterization problem is hard, there are several ways of examining the self-spanner properties of certain graphs. In this subsection, we investigate some popular graph classes, graph operations and network topologies with respect to their self-spanner properties.

Self-Spanner Properties of Graph Classes

We start this subsection by studying distance-hereditary graphs and chordal graphs. Both graph classes have been studied widely, see, e.g., Brandstädt (1993); Brandstädt et al. (1999b).

Distance-hereditary graphs. An important characterization of distance-hereditary graphs is based on *one-vertex extension* operations, which can be used to enlarge a distance-hereditary graph to another distance-hereditary graph containing more vertices. Let G be a graph, u be any vertex of G,

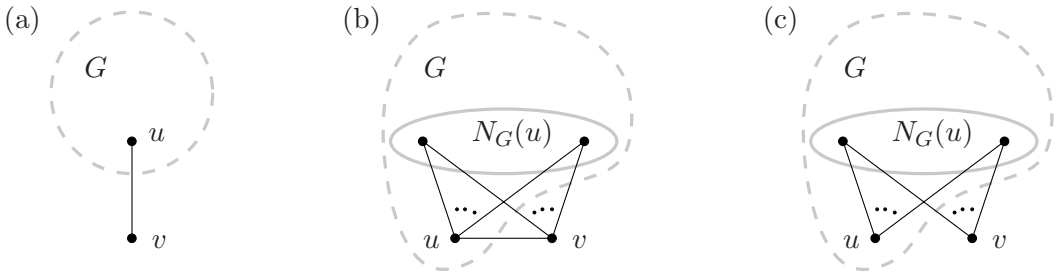

Figure 6.4: One-vertex extension operations for distance-hereditary graphs: introducing (a) a pendant vertex, (b) a true twin, (c) a false twin.

and v be a new vertex. The operations that can be applied to extend G by adding v are the following (see Figure 6.4):

$\boldsymbol{\alpha(u,v)}$: v is adjacent only to u (v is a *pendant vertex*);

$\boldsymbol{\beta(u,v)}$: v is adjacent to u and to every neighbor of u (v is a *true twin* of u);

$\boldsymbol{\gamma(u,v)}$: v is adjacent to every neighbor of u (v is a *false twin* of u).

Distance-hereditary graphs can be generated by using only these operations:

Lemma 6.29 (Bandelt and Mulder (1986)) *Every distance-hereditary graph is obtained starting from a single vertex by applying a sequence of α-, β-, and γ-operations.*

Remark 6.30 *Let G be a distance-hereditary graph and let $m \geq 4$ be the length of a maximum cycle in G. If G' is obtained from G by applying an α-operation then G' does not contain a cycle that is longer than m. The same also holds for a γ-operation if $m \geq 4$. In other words, α- and γ-operations do not increase cycle lengths in non-trivial distance-hereditary graphs.*

Observe that the remark on γ-operations does not hold for example for trees or for K_3. The previous lemma yields the following cycle-chord conditions.

Lemma 6.31 *In a distance-hereditary graph, every cycle $\mathscr{C}_n$, for $n \geq 3$, has*

1. *at least $n-3$ chords if n is odd, and*
2. *at least $n-4$ chords if n is even.*

Proof By Lemma 6.29 it is clear that distance-hereditary graphs form a hereditary class, i.e., every induced subgraph of a distance-hereditary graph is distance-hereditary. Let G be a distance-hereditary graph. Then, every subgraph of G that induces a cycle $\mathscr{C}_n$ is also a distance-hereditary graph.

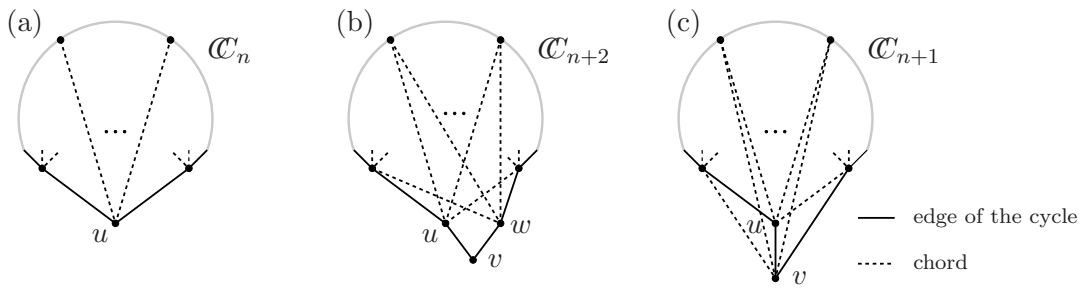

Figure 6.5: (a) A cycle $\mathcal{C}_n$, (b) the construction of a $\mathcal{C}_{n+2}$ from $\mathcal{C}_n$ by applying the operations $\alpha(u, v)$ and $\gamma(u, w)$, and (c) the construction of a $\mathcal{C}_{n+1}$ from $\mathcal{C}_n$ by applying the operation $\beta(u, v)$.

1. Let $n \geq 4$ be an even number, and consider a cycle $\mathcal{C}_n$ of G. We use an induction on n, and thus on the number of operations that generate $\mathcal{C}_n$.

   First observe that the induced cycle C_4 is distance-hereditary and that the base case of the induction is true. Now assume that the lemma holds for some even $n \geq 4$. We have to show that every cycle $\mathcal{C}_{n+2}$ has at least $n-2$ chords. According to the remarks above, every $\mathcal{C}_{n+2}$ is built from a cycle $\mathcal{C}_n$ by applying two operations of type α, β, or γ. Let $m \geq n-4$ denote the number of chords of this $\mathcal{C}_n$. Analyzing all nine possible pairs of operations shows that the minimum number of chords is obtained by first applying an α-operation and then a γ-operation on the same vertex. See Figure 6.5 for an illustration. By this, the number of chords in $\mathcal{C}_{n+2}$ is $m + deg(u) - 2 + 2 = m + deg(u)$, where u is the vertex to which the α- and γ-operations are applied, and $deg(u)$ denotes the degree of u.

   Note that by this construction, the vertex that is introduced by the α-operation (vertex v in Figure 6.5(b)) does not have an incident chord, and thus has degree 2. Using this as a lower bound in the induction, it follows that $\mathcal{C}_{n+2}$ contains at least $m + 2 \geq n - 2$ chords.

2. Now consider odd cycles, i.e., consider a cycle $\mathcal{C}_{n+1}$ where n is even. We have to show that $\mathcal{C}_{n+1}$ contains at least $n-2$ chords. As above, $\mathcal{C}_{n+1}$ is built from an even cycle $\mathcal{C}_n$ by applying one extension operation. But, by Remark 6.30 this operation is neither an α- nor a γ-operation. Hence, $\mathcal{C}_{n+1}$ is obtained from $\mathcal{C}_n$ by applying an operation $\beta(u, v)$, which results in the creation of $deg(u)$ new chords (see Figure 6.5(c)). Together with the results from Part 1, the minimal number of new chords is 2, and thus $\mathcal{C}_{n+1}$ contains at least $n - 2$ chords. □

Chordal graphs. A graph is chordal if every cycle of length at least 4 possesses a chord. Equivalently, a chordal graph does not contain an induced subgraph isomorphic to C_n for any $n \geq 4$.

Lemma 6.32 *Let G be a chordal graph. Every cycle $\mathscr{C}_n$ of G, for $n \geq 4$, has at least $n-3$ chords.*

Proof By definition, every subgraph of G that induces a cycle $\mathscr{C}_n$ is chordal. We show the lemma by induction.

The statement is true for $n = 4$, by definition. Now suppose that every cycle $\mathscr{C}_n$ for some $n \geq 4$ contains at least $n-3$ chords. Every cycle $\mathscr{C}_{n+1}$ of G (on $n+1$ vertices) can be generated from cycles $\mathscr{C}_n$ by inserting a new vertex. In order to maintain chordality, it is also necessary to produce a new chord. Thus, $\mathscr{C}_{n+1}$ contains at least $n-2$ chords, and the proof is complete. □

Self-spanner properties. By use of Corollary 6.28 together with Lemmas 6.31 and 6.32, we get the following self-spanner properties for the two graph classes:

Theorem 6.33

1. *Every distance-hereditary graph is in* $\mathrm{SS}(n, n-2)$ *for every even* $n \geq 4$*; for odd* $n \geq 3$*, distance-hereditary graph even belong to* $\mathrm{SS}(n, n-1)$.

2. *Every chordal graph is in* $\mathrm{SS}(n, n-1)$ *for every* $n \geq 4$.

To summarize this subsection, distance-hereditary and chordal graphs exhibit strong self-spanner properties: The stretch factor does not grow faster than the number of edge faults. In particular, if the number of edge faults is bounded by a constant then also the stretch factor is bounded by more or less the same constant.

Self-Spanner Properties of Graph Operations

In this subsection, we discuss two well-known graph operations that allow for efficient construction of self-spanner networks or easy recognition of special cases. In particular, we consider the *Cartesian product* and the *split composition* of graphs, and show how these operations affect the construction of (k, ℓ)–self-spanners.

Cartesian Product. The Cartesian product is defined as follows:

Definition 6.34 (Cartesian product, cf. Harary (1969))
Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two nontrivial graphs. The Cartesian product $G := G_1 \times G_2$ *is the graph with vertex set V and edge set E as follows:*

- $V = \{(x_1, x_2) \mid x_1 \in V_1, x_2 \in V_2\}$
- $E = \{\{(x_1, x_2), (y_1, y_2)\} \mid (x_1 = y_1 \text{ and } \{x_2, y_2\} \in E_2) \text{ or } (x_2 = y_2 \text{ and } \{x_1, y_1\} \in E_1)\}$.

Consequently, two vertices of $G_1 \times G_2$ are adjacent if and only if the first components are equal and the second components form an edge in G_2 or vice versa. W.l.o.g., we do not consider the case where G_1 or G_2 is a trivial graph having no edge; see Figure 6.6 for an example.

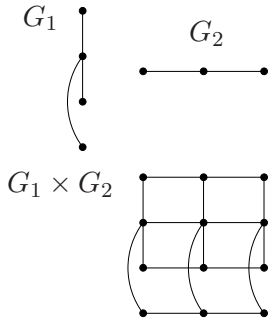

Figure 6.6: Cartesian product.

Many graphs that are used for modeling common network topologies can be defined in terms of Cartesian product of simpler graphs. We use this fact in the following subsection. The following properties of Cartesian product graphs are straightforward. Let $G := G_1 \times G_2$ and $i \in \{1, 2\}$. Then

1. $|V| = |V_1| \cdot |V_2|$, $|E| = |V_2| \cdot |E_1| + |V_1| \cdot |E_2|$.
2. G is connected if and only if both G_1 and G_2 are connected.
3. Let $cc(G)$ denote the number of connected components of G. Then $cc(G) = cc(G_1) \cdot cc(G_2)$.
4. If G_i is c_i–edge-connected then G is $(c_1 + c_2)$–edge-connected.
5. For any $x_1 \in V_1$, $G[\{(x_1, x_2) \mid x_2 \in V_2\}]$ is isomorphic to G_2, and for any $x_2 \in V_2$, $G[\{(x_1, x_2) \mid x_1 \in V_1\}]$ is isomorphic to G_1.

Since we are again only interested in connected components of G, it suffices to consider connected graphs G_1 and G_2. The next lemma shows that graphs that arise from the Cartesian product of two graphs have strong self-spanner properties. In particular, it indicates that a stretch factor of 3 plays an important role.

Theorem 6.35 *Let G_1 and G_2 be two connected graphs, $G = G_1 \times G_2$, and $i \in \{1, 2\}$. Then the following holds:*

1. *If $G_i \in \mathrm{SS}(k_i, \ell_i)$ and G_i is $(\ell_i + 1)$–edge-connected then $G \in \mathrm{SS}(\max\{k_1, k_2\}, \min\{\ell_1, \ell_2\})$.*

2. *Let δ be the minimum vertex degree of both G_1 and G_2. Then $G \in \mathrm{SS}(3, \delta)$ (in particular, $G \in \mathrm{SS}(3, 1)$).*

3. *$G \in \mathrm{SS}(2, \ell)$ if and only if every edge in G_i belongs to at least ℓ disjoint triangles within G_i.*

4. *If G_1 or G_2 contains a bridge then $maxT_2(G) = 0$, i.e., there is no $\ell > 0$ such that $G \in \mathrm{SS}(2, \ell)$. In particular, if G_1 or G_2 contains a bridge and $G \in \mathrm{SS}(k, \ell)$ for some $\ell > 0$, then $k \geq 3$.*

Proof Let $G = (V, E) = G_1 \times G_2$, $G_i = (V_i, E_i)$, and $i \in \{1, 2\}$.

1. Consider edge $e = \{(x_1, x_2), (y_1, y_2)\}$ in G. By Remark 6.14, it suffices to show that $d_{G'}((x_1, x_2), (y_1, y_2)) \leq \max\{k_1, k_2\}$, where G' is obtained from G by removing edge e and $\min\{\ell_1, \ell_2\}-1$ other arbitrary edges. By Item 5 above, e belongs to an induced subgraph G'' that is isomorphic either to G_1 or to G_2. By assumption, $G_i \in \mathrm{SS}(k_i, \ell_i)$ and G_i is (ℓ_i+1)–edge-connected. Hence, after the removal of at most $\min\{\ell_1, \ell_2\}$ edges (including e) from G'', G'' remains connected and the distance between (x_1, x_2) and (y_1, y_2) is at most $\max\{k_1, k_2\}$.

2. Consider edge $e = \{(x_1, x_2), (y_1, y_2)\}$ in G. We have to show that $d_{G'}((x_1, x_2), (y_1, y_2)) \leq 3$, where G' is obtained from G by removing e and $\delta - 1$ other arbitrary edges. By definition of the Cartesian product, either $x_1 = y_1$ and $\{x_2, y_2\} \in E_2$, or $x_2 = y_2$ and $\{x_1, y_1\} \in E_1$. W.l.o.g., assume the first case. As G_1 has minimum vertex degree δ, there exist δ vertices x_1^j in $V_1, 1 \leq j \leq \delta$, such that $\{x_1, x_1^j\} \in E_1$. The existence of these edges in G_1 implies that $\{(x_1, x_2), (x_1^j, x_2)\}$, $\{(x_1^j, x_2), (x_1^j, y_2)\}$, and $\{(x_1^j, y_2), (x_1, y_2)\}$ are disjoint edges in G for $1 \leq j \leq \delta$. These three edges each form disjoint paths in G from vertex (x_1, x_2) to vertex (y_1, y_2) (since by assumption $x_1 = y_1$), and hence $d_{G'}(\{(x_1, x_2), (y_1, y_2)\}) \leq 3$.

3. We have to show the 'only if'-part: Consider edge $e = \{(x_1, x_2), (y_1, y_2)\}$ in G and, w.l.o.g., assume that $x_1 = y_1$ and $\{x_2, y_2\} \in E_2$. Since $G \in \mathrm{SS}(2, \ell)$, there are ℓ edge disjoint paths from (x_1, x_2) to (y_1, y_2) of length at most 2 in G not using e. According to the proof of Part 2, any path from (x_1, x_2) to $(y_1,\ y_2) \equiv (x_1, y_2)$ via a vertex (v, w) with $v \neq x_1$ has length at least 3. Thus, there are vertices $z_j \in V_2$ such that $\{(x_1, x_2), (x_1, z_j)\}$, $\{(x_1, z_j), (x_1, y_2)\} \in E$, and $\{x_2, z_j\}, \{z_j, y_2\} \in E_2$

for $1 \leq j \leq \ell$. Hence, e belongs to ℓ disjoint triangles in G_2. The same arguments hold for G_1.

4. Part 4 is a special case of Part 3. □

Observe that, for Part 1 of the previous theorem, it is really necessary to claim the respective edge connectivity. Otherwise we cannot guarantee that the graph considered in the proof remains connected. Also, for Part 3 of that theorem, it does not suffice to claim that $G_1 \in \mathrm{SS}(2, \ell)$ (and $G_2 \in \mathrm{SS}(2, \ell)$, respectively): we again need that both graphs are $(\ell + 1)$–edge-connected. For smaller stretch factors, i.e., $k = 1$, we already know that $G_1 \times G_2$ has a stretch factor smaller than 2 if and only if it is a tree.

Remark 6.36 *Part 2 of Theorem 6.35 is tight in the following sense: If* $G_i \notin \mathrm{SS}(2, 1)$ *and* G_i *has minimum vertex degree* δ *for* $i \in \{1, 2\}$*, then* $minS_\delta(G_1 \times G_2) = 3$ *and* $maxT_3(G_1 \times G_2) = \delta$*. Thus* $G_1 \times G_2 \in \mathrm{SS}(3, \delta)$*, but* $G_1 \times G_2 \notin \mathrm{SS}(2, \delta)$ *and* $G_1 \times G_2 \notin \mathrm{SS}(3, \delta + 1)$*.*

Split Composition. The split composition is defined as follows:

Definition 6.37 (split composition)
Let $G_i = (V_i \cup \{m_i\}, E_i)$*, for* $i \in \{1, 2\}$*, where* $V_1 \cap V_2 = \emptyset$*. The vertices* m_1 *and* m_2 *are called* marked vertices*. The* split composition $G := G_1 * G_2$ *is the graph having vertex set* V *and edge set* E *as follows:*

- $V = V_1 \cup V_2$,
- $E = E(G_1[V_1]) \cup E(G_2[V_2]) \cup \{\{x, y\} \mid x \in N_{G_1}(m_1), y \in N_{G_2}(m_2)\}$*, where* $N_G(v)$ *denotes the neighborhood of vertex* v *in* G*.*

That means that $G := G_1 * G_2$ is obtained from G_1 and G_2 by taking all edges that lie within V_1 and V_2, and by connecting all neighbors of m_2 in G_2 to all neighbors of m_1 in G_1. See Figure 6.7 for an example. Observe that for all $y \in N_{G_2}(m_2)$, $G[V_1 \cup \{y\}]$ is isomorphic to G_1. The same holds analogously for G_2.

The split composition is the inverse of the decomposition operation introduced in Cunningham (1982). In Brandstädt (1993), split composition has been used for example to build distance-hereditary graphs, complete graphs, complete bipartite graphs, trees, and cographs. In Cicerone and Di Stefano (1998), it is

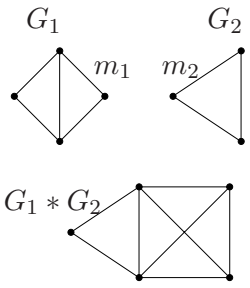

Figure 6.7: Split composition.

shown how split composition can be used to construct k–bounded induced distance graphs. The following lemma shows how self-spanner properties can be achieved when using the split composition.

Theorem 6.38 *Let $G_i \in \mathrm{SS}(k_i, \ell_i)$, for $i \in \{1, 2\}$.*
Then $G_1 * G_2 \in \mathrm{SS}(\max\{k_1, k_2\}, \min\{\ell_1, \ell_2\})$.

Proof Any edge in $G_1 * G_2$ belongs to an induced subgraph isomorphic to G_1 or to G_2. The rest of the proof follows the same lines as the proof of Part 1 of Lemma 6.35. □

It is clear that the stretch factor and fault-tolerance value of the previous lemma are tight: If k_1 and k_2 are minimum stretch factors for fault-tolerance values ℓ_1 and ℓ_2, then $G_1 * G_2$ cannot have a stretch factor less than $\max\{k_1, k_2\}$ for the fault-tolerance value $\min\{\ell_1, \ell_2\}$.

Self-Spanner Properties of Network Topologies

As we have seen in the previous subsection, we can construct graphs that exhibit certain self-spanner properties by using the Cartesian product. We now follow the opposite approach and examine some popular network topologies with respect to their self-spanner properties. In particular, we consider mesh-like networks like *grid*, *torus*, and *hypercube*. As examples for hypercube derived networks, we investigate *cube connected cycles* and *butterfly* networks. Especially the last two topologies, which are bounded-degree approximations of the hypercube, are widely studied (and used) in the interconnection network community (see for example Leighton (1992); Annexstein et al. (1990) and the references therein).

Grids, tori, and hypercubes. We first consider mesh-like networks:

Definition 6.39 (mesh-like networks)

1. *A* grid $G_{n,m}$ *(with $n, m \geq 2$) has $m \cdot n$ vertices labeled with distinct pairs (i, j), where $1 \leq i \leq n$ and $1 \leq j \leq m$. Two vertices are adjacent if their labels differ by 1 in exactly one coordinate.*

2. *A* torus $T_{n,m}$ *is a grid $G_{n,m}$ with wrap-around edges.*

3. *The d-*dimensional binary hypercube H_d *for $d \geq 1$ has 2^d vertices, which are labeled with the binary strings of length d. Two vertices in H_d are adjacent if their labels differ in exactly one bit.*

These topologies are simply structured with respect to the Cartesian product:

- the grid $G_{n,m}$ is the Cartesian product $P_n \times P_m$ for $n, m \geq 2$;
- the torus $T_{n,m}$ is the Cartesian product $C_n \times C_m$ for $n, m \geq 3$;
- The hypercube H_d is recursively defined from P_2 by $H_d = P_2 \times H_{d-1} = \underbrace{P_2 \times \cdots \times P_2}_{d \text{ times}}$.

The following lemma indicates the self-spanner properties of these topologies.

Theorem 6.40

1. *$G_{n,m}$ belongs to $\mathrm{SS}(3,1)$, but not to $\mathrm{SS}(2,1)$.*
   If $n > 2$ or $m > 2$ then $G_{n,m}$ does not belong to $\mathrm{SS}(3,2)$.
   If $n, m > 2$ then $G_{n,m}$ belongs to $\mathrm{SS}(5,2)$,
   but not to $\mathrm{SS}(4,2)$ or $\mathrm{SS}(5,3)$.

2. *$T_{n,m}$ belongs to $\mathrm{SS}(3,2)$, but not to $\mathrm{SS}(2,2)$.*
   If $n > 3$ or $m > 3$ then $T_{n,m}$ does not belong to $\mathrm{SS}(3,3)$.
   $T_{n,m}$ belongs to $\mathrm{SS}(\min\{5, \max\{n,m\} - 1\}, 3)$.
   If $n, m \geq 5$ then $T_{n,m}$ belongs to $\mathrm{SS}(5,4)$, but not to $\mathrm{SS}(4,4)$.
   If $n, m > 5$ then $T_{n,m}$ does not belong to $\mathrm{SS}(5,5)$.

3. *H_d belongs to $\mathrm{SS}(3, d-1)$, but not to $\mathrm{SS}(3,d)$ or to $\mathrm{SS}(2,1)$.*

Proof

1. $G_{n,m} \in \mathrm{SS}(3,1)$ and $G_{n,m} \notin \mathrm{SS}(2,1)$ are immediate consequences of Parts 2 and 4 of Lemma 6.35. To see the other self-spanner properties, observe that, for any edge on the boundary of the grid, there is only one path of length 3 connecting the end-vertices of that edge, all other paths have length 5 or longer. This 3–path (and the edge itself) may be broken by a double edge fault such that the end-vertices still remain connected (if n, m are large enough). Accordingly, $G_{n,m} \in \mathrm{SS}(5,2)$. If $G_{n,m} \neq C_4$ then $G_{n,m} \notin \mathrm{SS}(4,2)$ and if $n, m > 2$, $G_{n,m} \notin \mathrm{SS}(5,3)$.

2. Parts 2 and 3 of Lemma 6.35 directly imply that $T_{n,m} \in \mathrm{SS}(3,2)$ and $T_{n,m} \notin \mathrm{SS}(2,2)$. From Remark 6.36 it follows that $T_{n,m} \notin \mathrm{SS}(3,3)$, if $m > 3$ or $n > 3$. Observe that $T_{3,3} \in \mathrm{SS}(3,3)$.

   For every edge $\{x, y\}$ in $T_{n,m}$ there are two edge disjoint paths of length 3 connecting x and y and one (also disjoint) path of length at most $\max\{n,m\} - 1$. If n and m are at least 5, then there are six different paths of length 5 connecting x and y, but only two of length at most 4. It is easy to see that at least one of these paths

of length 5 remains complete if $\{x, y\}$ and three further edges are removed. If n and m are at least 6, consider the case of fault of five direct parallel edges in $T_{n,m}$: $T_{n,m}$ remains connected and the middle failing edge has a stretch factor that is greater than 5. Consequently, $T_{n,m} \in \mathrm{SS}(\min\{5, \max\{n, m\} - 1\}, 3)$. For m, n large enough, $T_{n,m} \in \mathrm{SS}(5, 4)$, but $T_{n,m} \notin \mathrm{SS}(4, 4)$ and also $T_{n,m} \notin \mathrm{SS}(5, 5)$.

3. To show that H_d belongs to $\mathrm{SS}(3, d-1)$, but not to $\mathrm{SS}(3, d)$, it is sufficient to observe that every edge e of H_d belongs to $d-1$ induced cycles of length 4 that are edge disjoint apart from e. By Part 4 of Lemma 6.35, H_d does not belong to $\mathrm{SS}(2, 1)$. □

Observe that the fault-tolerance value of the torus is higher than that of the grid, due to the additional wrap-around connections, which make the topology symmetric. But note that the addition of edges does not result in higher fault-tolerance values in general, confer Remark 6.18.

Furthermore, note that the hypercube H_d still guarantees a constant stretch factor 3, even if $d-1$ edges fail, i.e., if the number of edge faults is in the order of the dimension of H_d. Consequently, this topology expresses especially strong self-spanner properties.

Hypercube derived networks. We now consider two different types of bounded-degree approximations of the hypercube. We (mostly) follow the notation as in Heydemann et al. (1994).

Definition 6.41 (hypercube derived networks)

1. *The* cube-connected cycles graph *of dimension d, denoted CCC_d, is derived from H_d by replacing each vertex of H_d by a* fundamental cycle *of length d. Each vertex of such a cycle is labeled by a tuple (i, x) for $0 \leq i \leq d-1$, and i is called the* level *of the vertex. Apart from the* cycle edges *of the fundamental cycles, every vertex (i, x) is connected to vertex $(i, x(i))$, where $x(i)$ denotes the vertex of H_d that is labeled by the same string as vertex x but with bit i flipped. These edges are called* hypercube edges.

2. *The* butterfly graph *(with wrap-around) of dimension d, denoted B_d, is derived from H_d similarly as CCC_d: B_d consists of the same vertices (i, x) for $0 \leq i \leq d-1$ as CCC_d, and the same* fundamental cycles *of length d. But now every vertex (i, x) is connected by two* hypercube edges *to vertices $(i+1, x(i))$ and $(i-1, x(i-1))$.*

CCC_d can be obtained from B_d by replacing every pair of hypercube edges $\{(i,x),(i+1,x(i))\}$ and $\{(i,x),(i-1,x(i-1))\}$ by one edge $\{(i,x),(i,x(i))\}$. Thus, CCC_d can be viewed as a spanning subgraph of B_d.

In Annexstein et al. (1990), it is shown that different hypercube-derived topologies can be embedded within other such topologies with small slowdown. Results on the existence of cycles and the construction of k–spanners can be found in Rosenberg (1991) and Heydemann et al. (1994), respectively. But all these results do not imply on the self-spanner properties of the topologies studied here. We get the following results concerning the self-spanner properties of the topologies above:

Theorem 6.42

1. *B_d belongs to* $\mathrm{SS}(3,1)$ *and to* $\mathrm{SS}(d+1,2)$, *but not to* $\mathrm{SS}(2,1)$, $\mathrm{SS}(d,2)$, *or* $\mathrm{SS}(d+1,3)$.

2. *CCC_d belongs to* $\mathrm{SS}(7,1)$ *and to* $\mathrm{SS}(\max\{7,d-1\},2)$, *but not to* $\mathrm{SS}(6,1)$.

Proof

1. Any edge of B_d belongs to exactly one induced cycle of length 4 consisting of two cycle edges and two hypercube edges. Thus, $B_d \in \mathrm{SS}(3,1)$. From Rosenberg (1991), we know that B_d does not contain a cycle of length 3 if $d > 3$. For smaller d, no cycle of length 3 contains a hypercube edge. Hence, $B_d \notin \mathrm{SS}(2,1)$.

   Now consider the case when two edges fail: If two edges of the same fundamental cycle fail, there still remains a path of length 3 connecting the end-vertices of the faulty edges each. If both cycle edges of a 4-cycle as mentioned above fail then there remains a path of length $d-1$ via a fundamental cycle, but no shorter one. If a cycle edge and a hypercube edge within such a 4-cycle fail then a shortest path of length $d+1$ remains but not two such paths.

2. CCC_d consists of the same fundamental cycles as B_d, but contains only half of the hypercube edges. This results in longer cycles: For every *hypercube edge*, there are two (shortest) edge disjoint paths of length 7 that connect the end-vertices. For every *cycle edge*, there is a path of length $d-1$ (via the fundamental cycle) and another (disjoint) path of length 7 using hypercube edges. Consequently, $CCC_d \in \mathrm{SS}(7,1)$ and $CCC_d \in \mathrm{SS}(\max\{7,d-1\},2)$, but $CCC_d \notin \mathrm{SS}(6,1)$. □

The previous lemma shows that bounded-degree approximations of the hypercube like CCC_d and B_d perform poorly with respect to their self-spanner properties: In the case of single edge faults the stretch factor is still a constant (though much larger than for the hypercube), but for double edge faults the stretch factor grows linearly with the dimension d. Thus, the guarantees for delays in case of faults are really weak for these kinds of topologies. The big difference between the self-spanner properties of H_d on the one side, and CCC_d and B_d on the other are due to the bounded degree.

6.4 Further Remarks

In this chapter, we have introduced the classes of k–self-spanners and (k, ℓ)–self-spanners, which model survivable networks with bounded delay. Hence, these networks guarantee constant stretch factors even in the case of multiple edges faults. We have considered both the cases of unlimited and limited number of edge faults. We have given characterizational as well as structural results, and we have shown that some popular network topologies and graph classes exhibit (more or less) strong self-spanner properties.

We consider this work as a first step towards a more general approach to the design of survivable networks with bounded delay, and naturally there remain many open problems. On the one hand, it would be interesting how well Maximum k–Fault-Tolerance can be approximated for the cases where it is $\mathcal{NP}$-complete. On the other hand we are interested in further investigating the self-spanner properties of other popular topologies. Another further goal is to design sparse (k, ℓ)–self-spanner networks for given parameters k and ℓ such that specific connectivity requirements are fulfilled.

Chapter 7

Conclusion

In this thesis, we have studied different graph-theoretic aspects of a problem that arises within network design. Our goal was to find or construct sparse networks that guarantee short distances. Furthermore, our emphasis was on integrating some notion of simple structure and fault-tolerance. As a starting point, we have used the general framework of k–spanners, which are spanning subgraphs that fulfill some distance constraint.

Basically, we have followed two different approaches: On the one hand, in Chapter 6, we have employed k–spanners to construct survivable networks with bounded delay. On the other hand, in Chapters 3, 4, and 5, we have used k–spanners to analyze a given graph and to find subgraphs that fulfill particular requirements according to the actual objective. We have considered three different thematic contexts. In this chapter, we review this work in terms of the methodologies used.

As previous work on k–spanners has shown, spanners are often difficult to find, and most problems in this area turn out to be $\mathcal{NP}$-hard. Moreover, the concept of minimum, and in particular tree k–spanners, exhibit some drawbacks with respect to network design, such as error-proneness. In the first part of the thesis, we have thus studied variations of k–spanners in order to overcome the structural disadvantages and the hardness. Essentially, we have used four techniques:

- Instead of considering general graphs, we have restricted the instances to *planar* and ℓ*–outerplanar* graphs (see Sections 3.2 and 3.3). We have shown that the problem of finding minimum k–spanners remains $\mathcal{NP}$-hard for most stretch factors k in planar graphs, whereas the problem is efficiently solvable for ℓ–outerplanar graphs.

- In order to model additional objectives, we have imposed even stronger constraints on the subgraph. Due to this, we expect to have fewer

graphs that fulfill the new requirements. Consequently, this may affect the complexity of the considered problem significantly.

In particular, we have taken this course by studying minimum *planar* k–spanners and *independent* tree k–spanners (see Sections 3.4 and 4.2). In both cases, however, the complexity status of the respective problems does not change significantly.

- Other modifications of the requirements have followed the opposite direction. Instead of making the requirements stronger, we have slackened them. Also in this case the complexity status may change. Here, we have studied two modifications:

  In the first case, we have relaxed on the distance constraint. Instead of demanding that the distance constraint is fulfilled for every pair of vertices, the distance constraint has to hold for only some specified pairs (see Section 4.3: independent tree k–root-spanners). Here, the complexity status of the corresponding decision problem gets even worse compared with the stricter version.

  In the second modification, we have introduced auxiliary vertices and edges such that these may help in finding a tree k–spanner without imposing further restrictions (see Section 5.1: Steiner tree k–spanners). Also in this case the complexity status remains hopeless.

- We have reconsidered an $\mathcal{NP}$-hard optimization problem with respect to its (in)approximability (see Section 5.4: minimum Steiner tree k–spanner). As it turned out here, we cannot even hope for giving a constant approximation algorithm; at its best a logarithmic approximation algorithm might be found.

To summarize, the results indicate even more that the problem of finding k–spanners in their different shaping is difficult, and only some special cases can be solved efficiently.

Naturally, this work leaves many open problems and directions of further research. We do not go into detail here, but refer to the discussions at the end of each chapter of the body of this thesis.

Appendix A

List of Symbols

Graphs

$\|f\|$	size of face f, i.e., number of incident edges
$\{u, v\}$	undirected edge from u to v
(u, v)	directed edge (arc) from u to v
B_d	d-dimensional butterfly graph
$B(G)$	block-graph of G, a ①–②–tree
$\widetilde{B}(G)$	simple block-graph of G
$\mathrm{BID}(k)$	class of k–bounded induced distance graphs
C_n	induced cycle of length n
$\not{C}_n$	non-induced cycle of length n (containing chords)
$cc(G)$	number of connected components of G
CCC_d	d-dimensional cube-connected cycles graph
$d_G(u, v)$	distance between u and v in G
$E(G)$	set of edges of G
$G_{n,m}$	$n \times m$ grid
$G[R]$	subgraph of G induced by R
G^*	geometric dual of G
$G - E'$	subgraph of G in which the edges of E' are deleted
$G - F$	subgraph of G in which the faces of F are deleted
H_d	d-dimensional binary hypercube
$IR\,(e)$	influence region of edge e
$IR\,(f)$	influence region of face f
K_n	complete graph on n vertices
$K_{n,m}$	complete bipartite graph on n and m vertices

Graphs

L_ℓ	level ℓ: vertices at distance ℓ from the root
$l(v)$	level index of v
$maxT_k(G)$	maximum fault-tolerance value of G for fixed k
$minS_\ell(G)$	minimum stretch factor of G for fixed ℓ w.r.t. (k,ℓ)–self-spanners
$minS(G)$	minimum stretch factor w.r.t. k–self-spanners
$N_G(v)$	neighborhood of v in G
P_n	induced path of length $n-1$ on n vertices
$rp(v,T)$	root path: unique path from v to r in T
$\mathrm{SS}(k)$	class of k–self-spanner graphs
$\mathrm{SS}(k,\ell)$	class of (k,ℓ)–self-spanner graphs
$T_{n,m}$	$n \times m$ torus
$V(G)$	set of vertices of G

Miscellaneous

$\mathbb{N}$	set of natural numbers
$\mathbb{R}^+$	set of non-negative real numbers
$\mathbb{R}^{\geq c}$	set of real numbers at least c
log	logarithm to any base
$\lvert A\rvert$	cardinality of a set A
$A \subseteq B$	A is a subset of B (proper or non-proper)
$f(n) \in \mathcal{O}(g(n))$	f grows at most as fast as g
$f(n) \in o(g(n))$	f grows more slowly than g
$\mathcal{DTIME}(f(n))$	class of decision problems that can be solved within time $\mathcal{O}(f(n))$ by a deterministic Turing machine
$\mathcal{NTIME}(f(n))$	class of decision problems that can be solved within time $\mathcal{O}(f(n))$ by a non-deterministic Turing machine

Appendix B

Glossary of Graph-Theoretic Terms

Apart from parts of Chapter 3, we deal with *unweighted* and *undirected* graphs. Thus, in this glossary, we concentrate on the notation for unweighted, undirected graphs and only mention some of the necessary (but straightforward) extensions for the case of directed graphs or graphs that have associated weights. For further details on standard graph theoretic results see for example Harary (1969); Bondy and Murty (1976); Even (1979); Golumbic (1980), just to mention a few. Brandstädt (1993); Brandstädt et al. (1999b) contain catalogs of different graph classes. For a survey on planar graphs see, e.g., Nishizeki and Chiba (1988).

articulation vertex:
A 1–separator, i.e., a vertex whose deletion disconnects the graph.

bipartite:
A graph is *bipartite* if its vertices can be partitioned into two disjoint sets V_1 and V_2 such that V_1 and V_2 are stable, i.e., such that all edges have one end-vertex in V_1 and the other in V_2.

block:
A maximal biconnected induced subgraph of a graph.

block-graph, simple block-graph:
The *simple block-graph* $\widetilde{B}(G)$ arises from a graph G by taking the blocks of G as nodes. Two nodes are connected if the corresponding blocks share an articulation vertex.

The *block-graph* $B(G)$ of a graph G is a proper ①–②–tree, in which nodes labeled ① represent the blocks of G, nodes labeled ② represent

articulation vertices. A ②–node is connected to a ①–node by an edge, whenever the corresponding articulation vertex is contained in the respective block. Furthermore, add an additional ②–node for every block that contains only one articulation vertex. Note that $B(G)$ is a tree, and the leaves of $B(G)$ are ②–nodes.

bridge:
An edge e of G for which $G - e$ is disconnected.

chord:
An edge e that is incident to two vertices that occur in a path or a cycle and that are not adjacent in the path or cycle.

chordal:
A graph is *chordal* if every cycle of length at least 4 has a chord, i.e., if it does not contain an induced cycle of length 4 or longer.

clique:
A (sub)set of vertices is a *clique* if all pairs of vertices are adjacent. A graph is a *clique* or a *complete graph* if its set of vertices forms a clique.

complete graph, n–clique, complete bipartite graph:
A *complete graph* (or *n–clique*) on n vertices contains all $\binom{n}{2}$ edges and is denoted by K_n. A *complete bipartite graph* on n and m vertices, $K_{n,m}$, consists of two disjoint vertex sets V_1 and V_2 (with $|V_1| = n$ and $|V_2| = m$) such that every vertex in V_1 is connected to every vertex in V_2 by an edge and there is no edge within V_1 or V_2.

connected, biconnected, ℓ–vertex-connected, ℓ–edge-connected:
G is *connected* if there exists a path between every pair of vertices, otherwise it is *disconnected*. A maximal connected subgraph of a disconnected graph is called *connected component*. The number of connected components of G is denoted by $cc(G)$. A graph is *ℓ–vertex-connected* (or *ℓ–edge-connected*, respectively) if no deletion of $\ell - 1$ vertices (or edges, respectively) disconnects it. 2–vertex-connected graphs are also called *biconnected*.

cycle, simple cycle, induced cycle:
A *cycle* is a closed path, i.e., a path $\{u_1, u_2\}, \{u_2, u_3\}, \ldots, \{u_\ell, u_1\}$. A cycle is *simple* if no vertex appears more than once. A cycle C is *induced* in G if $G[V(C)] = C$, i.e., if C contains no chord. An induced cycle on n vertices is denoted by C_n, a cycle on n vertices that may contain a chord is denoted by $\mathcal{C}_n$.

degree:

The degree of a vertex v is the number of vertices that are adjacent to v.

diameter:

The *diameter* of a graph G is the length of a longest shortest path in G, i.e., $\max_{u,v \in V} d_G(u, v)$.

diamond:

A *diamond* is a graph formed by two possibly adjacent vertices u and v that are connected by $K \geq 2$ disjoint paths of length 2. Note that diamonds are biconnected.

directed graph, digraph, arc, oriented graph:

A *directed graph* (or *digraph*) is a graph in which edges (also called *arcs*) are formed by an *ordered* pair (u, v) of vertices u and v, $u \neq v$. We do not allow multiple (parallel) arcs for every ordered pair of vertices. A directed graph G is *oriented* if for no pair of vertices u and v both (u, v) and (v, u) are arcs in G, i.e., if there are no anti-parallel arcs.

distance:

The length of a shortest path from u to v in G, denoted by $d_G(u, v)$.

distance-hereditary graph:

A graph G is *distance-hereditary* if the distance in every connected induced subgraph G' of G is not longer than the distance in G, i.e., if $d_{G'}(u, v) = d_G(u, v)$ for all $u, v \in G'$.

forest:

A graph without cycles that is not necessarily connected, i.e., a collection of vertex disjoint trees.

geometric dual:

Given a planar graph G together with a planar embedding, its *geometric dual* G^* is constructed by placing a vertex in every face of G (including the outer face). For every edge e of G, add an edge e^* in G^* as follows. Let f and g be the two faces incident to e (f and g not necessarily distinct), then $e^* = \{f^*, g^*\}$ where f^* and g^* are the corresponding vertices in G^*. We refer to vertices of G^* as *nodes*.

Note that G^* may have multiple edges and loops, and that G^* is connected and planar. If G is biconnected then G^* does not contain loops.

graph, vertex, edge, incident, adjacent:

An *undirected graph* (or simply *graph*) $G = (V, E)$ consists of a set of *vertices* V and a set of *edges* E. We always assume these sets to be finite. Every edge e is an unordered pair $\{u, v\}$ of vertices $u \neq v$, i.e., we do not allow *loops*. Furthermore, we assume that every edge is unique, i.e., there are no *multiple edges*. $V(G)$ (or $E(G)$, respectively) denotes the set of vertices (or edges, respectively) of G. We say that $e = \{u, v\}$ is *incident* with vertices u and v, and both u and v are *end-vertices* of e. Two vertices u and v are *adjacent* if there is an edge which is incident with both of them. Two edges are *adjacent* if they share a common end-vertex.

independent trees:

Two spanning trees T_1 and T_2 of a graph G, both rooted at r, are *edge independent* (or *vertex independent*, respectively) *w.r.t.* r, if for every vertex $v \in V(G)$ the unique root paths $rp(v, T_1)$ and $rp(v, T_2)$ are edge disjoint (or internally vertex disjoint, respectively).

interval graph:

A graph is an *interval graph* if there is a bijection between its vertices and a set of intervals $\mathcal{I}$ on the real line, such that two vertices are connected by an edge if and only if their corresponding intervals intersect. Note that interval graphs are chordal.

level, parent vertex, sibling vertex:

Given a graph $G = (V, E)$ and a specified root $r \in V$, we partition V into *levels* $L_\ell := \{v \in V |\ d_G(v, r) = \ell\}$ for $0 \leq \ell \leq \max_v\{d_G(v, r)\}$. The *level index* of a vertex v is indicated by $l(v) := d_G(v, r)$. The level $L_{l(v)-1}$ is called *parent level* of vertex v. A vertex in $L_{l(v)-1} \cap N(v)$ is a *parent vertex* of v, and a vertex in $L_{l(v)} \cap N(v)$ is a *sibling vertex* of v.

monadic second order logic:

Monadic second order logic can be used as a language to describe graph properties, using the following constructions:

- quantification over vertices, edges, sets of vertices, sets of edges;
- membership and adjacency tests; and
- logic operations.

Extensions allow for example to optimize over the size of a free set variable. More details and formal definitions can be found for example in Courcelle (1997); Papadimitriou (1994).

Many problems and graph properties can be formulated in terms of (extended) monadic second order logic. These can be solved in linear time on graphs of bounded treewidth (see, e.g., Courcelle (1990); Arnborg et al. (1991); Courcelle and Mosbah (1993); Bodlaender (1996)).

neighborhood:

The *neighborhood* $N_G(v)$ of a vertex v in G is the set of all vertices that are adjacent to v in G. A vertex of $N_G(v)$ is also called *neighbor* of v.

outerplanar, ℓ–outerplanar:

An *outerplanar* graph is a planar graph which has a planar embedding such that all vertices lie on the same face, say the outer face. Given a planar embedding of a planar graph $G = (V, E)$, we assign labels to the vertices as follows: Vertices on the outer face are labeled 1. A vertex v has label i if it is on the outer face of the embedded subgraph in which all vertices labeled $i-1$ and lower have been deleted. The labels of the vertices can be computed in linear time using the algorithm of Lipton and Tarjan (1979).

Note that an outerplanar graph contains at most $2|V| - 3$ edges, and that the number of inner faces in a *maximal* outerplanar graph is $|V|-2$. An outerplanar graph is at most biconnected, and every block of an outerplanar graph is again outerplanar. Furthermore, every biconnected outerplanar graph has a unique outerplanar embedding. See Sysło (1979) for a survey on outerplanar graphs.

A graph is *ℓ–outerplanar* if it has a planar embedding such that ℓ is the maximum label of a vertex. The term outerplanar is thus equivalent to 1–outerplanar.

Given a planar graph $G = (V, E)$, an ℓ–outerplanar embedding of G such that ℓ is minimal can be found in polynomial time, see Bienstock and Monma (1990). In general, the degree of outerplanarity of a planar graph G, i.e., the smallest ℓ such that there is an ℓ–outerplanar embedding for G, is in the order of the $|V|$. Here, we always assume ℓ to be a constant parameter. Moreover, we always consider an ℓ–outerplanar graph together with an ℓ–outerplanar embedding.

path, disjoint paths, simple path, induced path, shortest path:

A *path* P from vertices u to v is a set $\{u, u_2\}, \{u_2, u_3\}, \ldots, \{u_{\ell-1}, v\}$ of edges. Vertices $u_2, \ldots, u_{\ell-1}$ are *internal* vertices. Alternatively, P is specified by the vertices that appear within the path, and P denotes the graph that consists exactly of the edges and vertices of the path.

Two paths are *edge disjoint* (or *internally vertex disjoint*, respectively) if they do not share an edge (or an internal vertex, respectively). A path is *simple* if no vertex appears more than once. A path P is *induced* in G if $G[V(P)] = P$, i.e., if P contains no chord. The *length* of a path is the sum of its edge weights. In the case of unweighted graphs, the length is the number of edges. A *shortest path* between two vertices u and v is a path having minimal length among all paths from u to v.

planar, face, outer face, inner face, size of a face:

A graph is *planar* if it can be embedded in the plane such that edges only intersect at common end-vertices. The planar embedding of G separates the plane into regions called *faces*. The unbounded face is the *outer face*, all others are *inner faces*. Let f be a face, denote by $s(f) = |f|$ the *size* of f, i.e., the number of edges that are incident to f.

Note that every edge is incident to one or two faces. If a graph is biconnected and planar then every edge is incident to exactly two faces. Two faces are *adjacent* if they share an incident edge. The number of faces $|F|$ is determined by $|F| = |E| - |V| + 2$, and the number of edges is bounded by $|E| \leq 3|V| - 6$ for $|V| \geq 3$.

There are linear time algorithms to check whether a graph is planar, and to construct a planar embedding (see, e.g., Hopcroft and Tarjan (1974); Booth and Lueker (1976)).

r–regular graph:

A graph where the degree of all vertices is r.

root, rooted tree, root path, ancestor, successor:

The *root* of a graph $G = (V, E)$ is a specified vertex $r \in V$. A tree is *rooted* if one particular vertex is the distinguished root. For a tree T rooted at r, the *root path* of a vertex v is the unique path from v to r, and is denoted by $rp(v, T)$. $rp(v, T)$ also denotes the set of internal vertices of the root path. For a vertex v, let w be the next vertex on the root path $rp(v, T)$. Then w is called *ancestor* of v, and v is called *successor* of w.

ℓ–separator:

A set of ℓ vertices whose deletion disconnects the graph.

split graph:

A graph is a *split graph* if its vertices can be partitioned into two disjoint sets V_1 and V_2 such that V_1 is a clique and V_2 is stable.

stable set, independent set:
A (sub)set of vertices is a *stable* or *independent set* if no pair of vertices is adjacent.

star:
A *star centered at a vertex* v is a graph in which v is adjacent to all other vertices and there are no other edges, i.e., it is a complete bipartite graph $K_{1,m}$, where v is the unique universal vertex.

①–②–star:
A ①–②–*star centered at a* ②–*node* is a proper ①–②–tree in which one central ②–node is adjacent to some ①–nodes, and each of these ①–nodes is adjacent to exactly one further ②–node.

subgraph, spanning subgraph, induced subgraph:
A *subgraph* $H = (V', E')$ of $G = (V, E)$ is a graph where $V' \subseteq V$ and $E' \subseteq E$. H is a *spanning* subgraph if $V = V'$. If R is a subset of V, then $G[R]$ represents the subgraph of G that is *induced* by R; i.e. $G[R]$ consists of the vertices of R and contains all edges of G that are incident only to vertices of R. $G - R$ is a shorthand for $G[V \backslash R]$. $G - E'$ where $E' \subseteq E$ is the graph obtained from G by deleting all edges in E'. If $E = \{e\}$ we write $G - e$.

tree, rooted tree:
A *tree* is a connected graph that does not contain a cycle. A *spanning tree* T of a G is a spanning subgraph of G that is a tree. Note that a tree with n vertices contains $n - 1$ edges, and is a minimal connected graph. Paths between any pair of vertices within a tree are unique.

***k*–tree:**
A graph G that is either a k–clique, or that contains a vertex v such that $G \backslash \{v\}$ is a k–tree and the neighborhood of v in G induces a k–clique.

①–②–tree, ①–②–subtree, ①–②–forest, proper ①–②–tree:
A ①–②–*tree* is a tree that consists of two different types of nodes, ①–nodes and ②–nodes, such that each ①–node (and ②–node, respectively) is adjacent only to ②–nodes (and ①–nodes, respectively). A ①–②–*subtree* is a subtree of a ①–②–tree. A ①–②–*forest* is a collection of ①–②–trees that are node disjoint. A ①–②–subtree T of a ①–②–tree B is *proper* if the following holds: whenever a ①–node b belongs to T then also all ②–nodes that are adjacent to b in B belong to T. A *proper* ①–②–forest consists only of proper ①–②–trees.

tree decomposition, treewidth:

A *tree decomposition* of a graph $G = (V, E)$ is a pair (X, T) where $T = (I, F)$ is a tree and $X = \{X_i \mid i \in I\}$ is a family of subsets of V, one for each vertex $i \in I$ of T, such that

- $\bigcup_{i \in I} X_i = V$;
- for all edges $\{u, v\} \in E$, there is an $i \in I$ with $u \in X_i$ and $v \in X_i$;
- for all $i, j, k \in I$: if j is on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$.

The *width* of a tree decomposition (X, T) is $(\max_{i \in I} |X_i|) - 1$. The *treewidth of* G is the minimum width among all tree decompositions of G. The notion of treewidth has been introduced by Robertson and Seymour (1983, 1986). See also Bodlaender (1993, 1997, 1998) for further results on the treewidth of graphs.

universal vertex:

A vertex v of a graph G is *universal* w.r.t. $V(G)$ (or *universal* for G) if is is adjacent to all other vertices of G.

weighted graph:

In an *edge weighted graph* (or simply *weighted graph*), we associate a non-negative weight $w : E \to \mathbb{R}^+$ with every edge e. In a *vertex weighted graph*, non-negative weights $w : V \to \mathbb{R}^+$ are associated with every vertex v. An *unweighted* graph can be seen as a weighted graph where all edges and vertices have unit weight.

Appendix C

Glossary of Complexity Theoretic Terms

For a comprehensive discussion and a formal treatment of the following topics we refer the reader for example to Aho et al. (1974); Garey and Johnson (1979); Papadimitriou (1994). For a survey on approximation algorithms and inapproximability results, see for example Hochbaum (1997) and Crescenzi and Kann (1995).

δ–approximation algorithm, approximation ratio, approximation scheme, inapproximability:

Let $\mathcal{A}$ be a polynomial-time algorithm that produces a feasible (though not necessarily optimal) solution for an optimization problem Π, and denote by $\mathcal{A}(I)$ the value of the solution that is achieved by $\mathcal{A}$ for the instance I. Furthermore, denote by $\mathrm{OPT}(I)$ the value of an optimal solution of$I$ in Π. $\mathcal{A}$ is a *δ–approximation algorithm* (where $\delta > 1$) for Π if for every instance I of Π,

$$\max\left\{\frac{\mathcal{A}(I)}{\mathrm{OPT}(I)}, \frac{\mathrm{OPT}(I)}{\mathcal{A}(I)}\right\} \leq \delta.$$

δ is called *approximation ratio* of $\mathcal{A}$. A *polynomial approximation scheme* is a family of algorithms that contains δ–approximation algorithms for *all* $\delta > 1$. Π is *inapproximable* (or *hard to approximate*) within some ratio δ if it is $\mathcal{NP}$-hard to find a δ–approximation algorithm for Π. Note that δ may be a function in the size of the instances.

complement of a decision problem:

Π^c is the *complement* of a decision problem Π if it has the same set of instances as Π and if an answer to an instance in Π^c is 'yes' if and only if the answer to this instance in Π is 'no'.

co-$\mathcal{NP}$:
The class of decision problems for which its complement is in $\mathcal{NP}$. It is open but widely believed that $co-\mathcal{NP} \neq \mathcal{NP}$.

decision problem:
A problem where the answer is either 'yes' or 'no'.

$\mathcal{DTIME}(f(n))$, $\mathcal{NTIME}(f(n))$:
For a non-decreasing function $f : \mathbb{N} \to \mathbb{N}$, the *deterministic time complexity class* $\mathcal{DTIME}(f(n))$ (and *non-deterministic time complexity class* $\mathcal{NTIME}(f(n))$, respectively) comprises those decision problems that can be solved within time $\mathcal{O}(f(n))$ by a deterministic Turing machine (and non-deterministic Turing machine, respectively).

$\mathcal{NP}$:
The class of decision problems that can be solved in polynomial time by a non-deterministic Turing machine. It is open but widely believed that $\mathcal{P} \neq \mathcal{NP}$.

$\mathcal{NP}$-complete, co-$\mathcal{NP}$-complete:
A decision problem Π is *$\mathcal{NP}$-complete* if $\Pi \in \mathcal{NP}$ (or $\Pi \in co-\mathcal{NP}$, respectively) and if, for every $\Pi' \in \mathcal{NP}$ (or $\Pi' \in co-\mathcal{NP}$, respectively), there is a polynomial transformation from Π' to Π.

Observe that $\mathcal{NP} = co-\mathcal{NP}$ if there is a co-$\mathcal{NP}$-complete problem that is also in $\mathcal{NP}$. Therefore, co-$\mathcal{NP}$-complete problems can be considered even harder than problems in $\mathcal{NP}$.

$\mathcal{NP}$-hard:
A problem Π (not necessarily a decision problem) is *$\mathcal{NP}$-hard* if for every decision problem $\Pi' \in \mathcal{NP}$ there is a polynomial transformation from Π' to Π. Note that an $\mathcal{NP}$-hard problem is at least as difficult as any problem in $\mathcal{NP}$.

$\mathcal{O}(g(n))$, $o(g(n))$:
A function $f : \mathbb{N} \to \mathbb{R}^+$ belongs to the class $\mathcal{O}(g(n))$ for a function $g : \mathbb{N} \to \mathbb{R}^+$ if there are constants $c \in \mathbb{R}$ and $n_0 \in \mathbb{N}$ such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$, i.e., if f grows at most as fast as g. $f \in o(g(n))$, if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, i.e., if f grows more slowly than g.

optimization problem:
An *optimization problem* is a problem (usually) having many feasible solutions which all have an associated value. The goal is to find a solution of optimal value. Here, optimal may mean minimum or maximum.

polynomial transformation, reduction:

Given two decision problems Π and Π', denote by I (and I', respectively) an encoded instance of Π (and Π', respectively). A *polynomial transformation* (also called *reduction*) from Π' to Π is a function f that transforms an instance of Π' into an instance of Π such that f can be computed in polynomial time, and the answer for I' is 'yes' in Π' if and only if the answer for $I = f(I')$ is 'yes' in Π. If such a reduction exists we write $\Pi' \propto \Pi$.

Note that, if $\Pi' \propto \Pi$, then we can use a polynomial algorithm for Π (if one exists) to find a polynomial algorithm for Π'.

$\mathcal{P}$:

The class of decision problems that can be solved in polynomial time by a deterministic Turing machine.

worst-case complexity:

The maximum running time of an algorithm over all instances of the same size.

Appendix D

Some $\mathcal{NP}$-complete Problems

In this section, we compile the problems that are used throughout this thesis as examples or in reductions. For each problem, we give an exact definition and state the complexity status as we need it in the context of this thesis. For a more detailed list see for example Garey and Johnson (1979); Crescenzi and Kann (1995). The problems are numbered as in Garey and Johnson (1979) and listed in the same order as they appear there.

Graph Theory

Problem D.1 Domatic Number, [GT3]

Given: A graph $G = (V, E)$ and a positive integer $K \leq |V|$.

Problem: Can V be partitioned into $\ell \geq K$ disjoint sets $V_1, V_2, \ldots, V_\ell$ such that every V_i is a dominating set for G (i.e., such that for every $1 \leq i \leq \ell$, every vertex in V is adjacent to a vertex in V_i)?

Domatic Number is $\mathcal{NP}$-complete, and remains so for any fixed $K \geq 3$ (see Garey and Johnson (1979)).

Problem D.2 Chromatic Number, [GT4]

Given: A graph $G = (V, E)$ and a positive integer $K \leq |V|$.

Problem: Is G K–colorable, i.e., is there a function $f : V \to \{1, \ldots, K\}$ such that $f(u) \neq f(v)$ whenever $\{u, v\} \in E$?

Chromatic Number is $\mathcal{NP}$-complete (see Karp (1972)), and remains so for all fixed $K \geq 3$. Chromatic Number also remains $\mathcal{NP}$-complete for *planar* graphs for $K = 3$ (see Garey et al. (1976)).

Problem D.3 MAXIMUM CLIQUE, [GT19]

Given: A graph $G = (V, E)$ and a positive integer $K \leq |V|$.

Problem: Does G contain a clique of size at least K?

MAXIMUM CLIQUE is $\mathcal{NP}$-complete (see Karp (1972)), but is solvable in polynomial time for *planar* graphs because planar graphs cannot contain a clique of size 5 or larger.

Network Design

Problem D.4 MINIMUM STEINER TREE, [ND12]

Given: A weighted graph $G = (V = R \dot{\cup} S, E)$ with a set of *terminals* R and a set of *Steiner vertices* S such that R and S are disjoint and with positive integer edge weights; a positive integer K.

Problem: Is there a subtree T of G that includes all the vertices from R (and maybe some vertices from S) such that the total edge weight of T is at most K?

Here, we only consider the MINIMUM STEINER TREE Problem in unweighted graphs, see for example Hwang et al. (1992) for a survey. MINIMUM STEINER TREE remains $\mathcal{NP}$-complete even for *planar* graphs (see Garey and Johnson (1977)). It can be approximated within a factor of 1.644 of the size of a minimum tree in polynomial time (Karpinski and Zelikovsky (1995)). But Bern and Plassmann (1989) show that there is no $(1 + \epsilon)$–approximation algorithm for arbitrary $0 < \epsilon < 1$, unless $\mathcal{P} = \mathcal{NP}$.

Problem D.5 MAXIMUM CUT, [ND16]

Given: A graph $G = (V, E)$ and a positive integer K.

Problem: Is there a partition of V into disjoint sets V_1 and V_2 such that the number of edges from E that have one end-vertex in V_1 and the other in V_2 is at least K?

MAXIMUM CUT is $\mathcal{NP}$-complete (see Garey et al. (1976)), but is solvable in polynomial time if G is *planar* (see Hadlock (1975)), even when G is edge weighted and the goal is to find a cut of maximum total edge weight.

Problem D.6 Longest Circuit, [ND28]

Given: A graph $G = (V, E)$ and a positive integer $K \leq |V|$.

Problem: Is there a simple cycle in G of length at least K?

Longest Circuit is $\mathcal{NP}$-complete (see Garey and Johnson (1979)).

Problem D.7 Maximum Length-Bounded Disjoint Paths, [ND41]

Given: A graph G, two vertices s and t, and positive integers $K, L \leq n$.

Problem: Does G contain L or more mutually edge disjoint paths from s to t, which all have length at most K?

Maximum Length-Bounded Disjoint Paths is $\mathcal{NP}$-complete, and remains so for all fixed $K \geq 5$; it is polynomially solvable for $K \leq 3$, and it is open for $K = 4$ (see Itai et al. (1982)).

Sets

Problem D.8 Minimum Set Cover, [SP5]

Given: A collection C of subsets of a finite set F, and a positive integer $K \leq |C|$.

Problem: Does C contain a cover for F of size at most K, i.e., a subset $C' \subseteq C$ with $|C'| \leq K$ such that every element of F belongs to at least one member of C'?

Minimum Set Cover is $\mathcal{NP}$-complete (see Karp (1972)). Furthermore, there is no $(1-\epsilon)\log|F|$–approximation algorithm for Minimum Set Cover for any $\epsilon > 0$, unless $\mathcal{NP} \subset \mathcal{DTIME}(n^{\log\log n})$, see Feige (1996). Johnson (1974) gave a $(1 + \log|F|)$–approximation algorithm.

Problem D.9 Minimum Hitting Set, [SP8]

Given: A collection C of subsets of a finite set F, and a positive integer $K \leq |F|$.

Problem: Is there a subset $F' \subseteq F$ with $|F'| \leq K$ such that F' contains at least one element from each subset in C?

As shown by Ausiello et al. (1980), Minimum Hitting Set is equivalent to Minimum Set Cover (Problem D.8). Thus, the hardness results and algorithms for Minimum Set Cover carry over to Minimum Hitting Set.

Satisfiability

Problem D.10 PLANAR 3–SATISFIABILITY, [LO2]

Given: A set U of Boolean variables, and a collection C of clauses over U with $|c| = 3$ for all $c \in C$; where the bipartite graph $G = (V, E)$ with $V = U \cup C$ and $E = \{\{x, c\} : x \text{ or } \overline{x} \text{ occurs in } c\}$ is planar.

Problem: Is there a satisfying truth assignment for C?

PLANAR 3–SATISFIABILITY is $\mathcal{NP}$-complete (see Mansfield (1983)).

Problem D.11 3–SATISFIABILITY, [LO2]

Given: A set U of Boolean variables, and a collection C of clauses over U with $|c| = 3$ for all $c \in C$.

Problem: Is there a satisfying truth assignment for C?

3–SATISFIABILITY is $\mathcal{NP}$-complete (see, e.g., Garey and Johnson (1979)).

Problem D.12 NOT–ALL–EQUAL 3–SATISFIABILITY, [LO3]

Given: A set U of Boolean variables, and a collection C of clauses over U with $|c| = 3$ for all $c \in C$.

Problem: Is there a truth assignment for U such that every clause in C has at least one true literal and at least one false literal?

NOT–ALL–EQUAL 3–SATISFIABILITY is $\mathcal{NP}$-complete (see Schaefer (1978)).

Problem D.13 ONE–IN–THREE SATISFIABILITY, [LO4]

Given: A set U of Boolean variables, and a collection C of clauses over U with $|c| = 3$ for all $c \in C$.

Problem: Is there a truth assignment for U such that every clause in C has exactly one true literal?

ONE–IN–THREE SATISFIABILITY is $\mathcal{NP}$-complete, and remains so for instances where no clause contains a negated literal (see Schaefer (1978)).

Bibliography

Aho, A. V., Hopcroft, J. E., and Ullman, J. D. (1974). *The Design and Analysis of Computer Algorithms*. Addison–Wesley.

Althöfer, I., Das, G., Dobkin, D., Joseph, D., and Soares, J. (1993). On sparse spanners of weighted graphs. *Discrete Computational Geometry*, 9:81–100.

Annexstein, F. S. (1991). Fault-tolerance in hypercube-derivative networks. *Computer Architecture News*, 19(1):25–34.

Annexstein, F. S., Baumslag, M., and Rosenberg, A. L. (1990). Group action graphs and parallel architectures. *SIAM J. on Computing*, 19(3):544–569.

Annexstein, F. S., Berman, K. A., and Swaminathan, R. (1996). Independent spanning trees with small stretch factors. Technical Report 96-13, DIMACS.

Annexstein, F. S., Berman, K. A., and Swaminathan, R. (1998). Approximation algorithms for optimal-depth independent spanning trees and *st*-numberings. In *Proc. 4th International Colloquium Structural Information and Communication Complexity, SIROCCO'97*, pages 12–23. Carleton Scientific.

Arnborg, S., Lagergren, J., and Seese, D. (1991). Problems easy for tree-decomposable graphs. *J. on Algorithms*, 12:308–340.

Arya, S., Das, G., Mount, D. M., Salowe, J. S., and Smid, M. (1995). Euclidean spanners: Short, thin, and lanky. In *Proc. 27th Annual ACM Symp. Theory of Computing, STOC'95*, pages 489–498.

Ausiello, G., D'Atri, A., and Protasi, M. (1980). Structure preserving reductions among convex optimization problems. *J. on Computer and System Sciences*, 21:136–153.

Baker, B. S. (1994). Approximation algorithms for $\mathcal{NP}$-complete problems on planar graphs. *Journal of the ACM*, 41(1):153–180.

Bandelt, H. J. and Dress, A. W. M. (1986). Reconstructing the shape of a tree from observed dissimilarity data. *Advances in Applied Mathematics*, 7:309–343.

Bandelt, H. J. and Mulder, H. (1986). Distance–hereditary graphs. *J. of Combinatorial Theory, Series B*, 41:182–208.

Bern, M. and Plassmann, P. (1989). The Steiner problem with edge lengths 1 and 2. *Information Processing Letters*, 32:171–176.

Bienstock, D. and Monma, C. L. (1990). On the complexity of embedding planar graphs to minimize certain distance measures. *Algorithmica*, 5:93–109.

Bodlaender, H. L. (1993). A tourist guide through treewidth. *Acta Cybernetica*, 11:1–21.

Bodlaender, H. L. (1996). A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM J. on Computing*, 25:1305–1317.

Bodlaender, H. L. (1997). Treewidth: Algorithmic techniques and results. In *Proc. 22nd International Symp. Mathematical Foundations of Computer Science, MFCS'97*, pages 29–36. Lecture Notes in Computer Science, vol. 1295, Springer.

Bodlaender, H. L. (1998). A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209:1–45.

Bondy, J. A. (1989). Trigraphs. *Discrete Mathematics*, 75:69–79.

Bondy, J. A. and Murty, U. R. (1976). *Graph Theory with Applications*. North–Holland.

Booth, K. S. and Lueker, G. S. (1976). Testing for the consecutive ones property, interval graphs, and planarity using PQ-tree algorithms. *J. on Computer and System Sciences*, 13:335–379.

Brandes, U. and Handke, D. (1997). $\mathcal{NP}$-completeness results for minimum planar spanners. In *Proc. 23rd International Workshop on Graph-Theoretic Concepts in Computer Science, WG'97*, pages 85–99. Lecture Notes in Computer Science, vol. 1335, Springer.

Brandes, U. and Handke, D. (1998). $\mathcal{NP}$-completeness results for minimum planar spanners. *Discrete Mathematics and Theoretical Computer Science*, 3(1):1–10.

Brandstädt, A. (1993). Special graph classes: A survey. Technical Report SM–DU–199, Universität Gesamthochschule Duisburg, Germany.

Brandstädt, A. (1994). *Graphen und Algorithmen*. Teubner.

Brandstädt, A., Chepoi, V., and Dragan, F. (1999a). Distance approximating trees for chordal and dually chordal graphs. *J. on Algorithms*, 30:166–184.

Brandstädt, A., Le, V., and Spinrad, J. (1999b). *Graph Classes: A Survey*. SIAM Monographs on Discrete Mathematics and Applications.

Bruck, J., Cypher, R., and Ho, C.-T. (1997). Fault-tolerant meshes with small degree. *SIAM J. on Computing*, 26(6):1764–1784.

Cai, L. (1992). *Tree Spanners: Spanning Trees that Approximate Distances*. PhD thesis, Dept. of Computer Science, University of Toronto, Canada.

Cai, L. (1994). $\mathcal{NP}$-completeness of minimum spanner problems. *Discrete Applied Mathematics*, 48:187–194.

Cai, L. and Corneil, D. G. (1995). Tree spanners. *SIAM J. Discrete Mathetics*, 8(3):359–387.

Cai, L. and Keil, M. (1994). Spanners in graphs of bounded degree. *Networks*, 24:233–249.

Chandra, B., Das, G., Narasimhan, G., and Soares, J. (1992). New sparseness results on graph spanners. In *Proc. 8th Annual Computational Geometry, Berlin, 1992*, pages 192–201.

Chen, D. Z., Das, G., and Smid, M. (1996). Lower bounds for computing geometric spanners and approximate shortest paths. In *Proc. 8th Canadian Conference on Computational Geometry, 1996*, pages 155–170.

Chew, L. P. (1986). There is a planar graph almost as good as the complete graph. In *Proc. 2nd ACM Symp. Computational Geometry*, pages 169–177. ACM Press.

Cicerone, S. and Di Stefano, G. (1998). Graphs with bounded induced distance. In *Proc. 24th International Workshop on Graph-Theoretic Concepts in Computer Science, WG'98*, pages 177–191. Lecture Notes in Computer Science, vol. 1517, Springer.

Cicerone, S., Di Stefano, G., and Handke, D. (1999). Survivable networks with bounded delay. Konstanzer Schriften in Mathematik und Informatik 81, Universität Konstanz, Germany.

Cole, R., Maggs, B. M., and Sitaraman, R. K. (1997). Reconfiguring arrays with faults part I: Worst-case faults. *SIAM J. on Computing*, 26(6):1581–1611.

Courcelle, B. (1990). The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Information and Computation*, 85:12–75.

Courcelle, B. (1997). The expression of graph properties and graph transformations in monadic second-order logic. In Rozenberg, G., editor, *Handbook of Graph Grammars and Computing by Graph Transformations*, volume 1: Foundations, chapter 5, pages 313–400. World Scientific.

Courcelle, B. and Mosbah, M. (1993). Monadic second-order evaluations on tree-decomposable graphs. *Theoretical Computer Science*, 109:49–82.

Crescenzi, P. and Kann, V. (1995). A compendium of NP optimization problems. Technical Report SI/RR-95/02, Universita di Roma La Sapienza. URL: `www.nada.kth.se/theory/problemlist.html`.

Cunningham, W. (1982). Decomposition of directed graphs. *SIAM J. on Algebraic and Discrete Methods*, 3:214–228.

Even, S. (1979). *Graph Algorithms*. Computer Software Engineering Series. Computer Science Press.

Farley, A. M. and Proskurowski, A. (1993). Self-repairing networks. *Parallel Processing Letters*, 3(4):381–391.

Feige, U. (1996). A threshold of $\ln n$ for approximating set cover. In *Proc. 28th Annual ACM Symp. Theory of Computing, STOC'96*, pages 314–318. ACM Press.

Fekete, S. P. and Kremer, J. (1998). Tree spanners in planar graphs. In *Proc. 24th International Workshop on Graph-Theoretic Concepts in Computer Science, WG'98*, pages 298–309. Lecture Notes in Computer Science, vol. 1517, Springer.

Garey, M. R. and Johnson, D. S. (1977). The rectilinear Steiner tree problem is $\mathcal{NP}$-complete. *SIAM J. on Applied Mathematics*, 32:826–834.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$–Completeness*. W H Freeman & Co Ltd.

Garey, M. R., Johnson, D. S., and Stockmeyer, L. (1976). Some simplified $\mathcal{NP}$-complete graph problems. *Theoretical Computer Science*, 1:237–267.

Golumbic, M. C. (1980). *Algorithmic Graph Theory and Perfect Graphs*. Academic Press.

Grötschel, M., Monma, C. L., and Stoer, M. (1995). Design of survivable networks. In Ball, M. O., Magnanti, T. L., Monma, C. L., and Nemhauser, G. L., editors, *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science*, pages 617–672. Elsevier Science Pub., Amsterdam.

Haberman, B. K. and Rouskas, G. N. (1997). Cost, delay, and delay variation conscious multicast routing. Technical Report TR-97-03, Dept. of Computer Science, North Carolina State University.

Hadlock, F. O. (1975). Finding a maximum cut of a planar graph in polynomial time. *SIAM J. on Computing*, 4:221–225.

Hakimi, S. and Yau, S.-T. (1964). Distance matrix of a graph and its realizability. *Q. J. on Mechanics and Applied Mathematics, Oxford University*, 22:305–317.

Handke, D. (1998). Independent tree spanners: Fault-tolerant spanning trees with constant distance guarantees (extended abstract). In *Proc. 24th International Workshop on Graph-Theoretic Concepts in Computer Science, WG'98*, pages 203–214. Lecture Notes in Computer Science, vol. 1517, Springer.

Handke, D. (1999). Independent tree spanners: Fault-tolerant spanning trees with distance guarantees. to appear in *Discrete Applied Mathematics*.

Harary, F. (1969). *Graph Theory*. Addison–Wesley.

Hayes, J. P. (1976). A graph model for fault-tolerant computing systems. *IEEE Transactions on Computers*, C-25(9):875–884.

Heydemann, M.-C., Peters, J. G., and Sotteau, D. (1994). Spanners of hypercube-derived networks. Technical Report CMPT TR 94-02, School of Computer Science, Simon Fraser University, Burnaby, Canada.

Hochbaum, D. S., editor (1997). *Approximation Algorithms for $\mathcal{NP}$-Hard Problems*. PWS Publishing Company.

Hopcroft, J. E. and Tarjan, R. E. (1974). Efficient planarity testing. *Journal of the ACM*, 21:549–568.

Howorka, E. (1977). A characterization of distance-hereditary graphs. *Q. J. of Mathematics, Oxford University*, 2(28):417–420.

Huck, A. (1994). Independent trees in graphs. *Graphs and Combinatorics*, 10:29–45.

Hwang, F., Richards, D., and Winter, P. (1992). *The Steiner Tree Problem*, volume 53 of *Annals of Discrete Mathematics*. North–Holland.

Itai, A., Perl, Y., and Shiloach, Y. (1982). The complexity of finding maximum disjoint paths with length constraints. *Networks*, 12:277–286.

Itai, A. and Rodeh, M. (1988). The multi-tree approach to reliability in distributed networks. *Information and Computation*, 79(1):43–59.

Johnson, D. S. (1974). Approximation algorithms for combinatorial problems. *J. on Computer and System Sciences*, 9:256–278.

Johnson, D. S. (1985). The $\mathcal{NP}$-completeness column: An ongoing guide. *J. on Algorithms*, 6:434–451.

Karp, R. M. (1972). Reducibility among combinatorial problems. In *Complexity of Computer Computations. Proc. Symp. Complexity of Computer Computations, New York*, pages 85–103. Plenum Press, New York.

Karpinski, M. and Zelikovsky, A. Z. (1995). New approximation algorithms for the Steiner tree problems. Technical Report TR95-030, Electronic Colloquium on Computational Complexity, ECCC.

Khuller, S. and Schieber, B. (1992). On independent spanning trees. *Information Processing Letters*, 42:321–323.

Klein, P., Rao, S. B., Rauch-Henzinger, M., and Subramanian, S. (1994). Faster shortest–path algorithms for planar graphs. In *Proc. 26th Annual ACM Symp. Theory of Computing, STOC'94*, pages 27–37.

Kortsarz, G. (1998). On the hardness of approximating spanners. In *Proc. International Workshop Approximation Algorithms for Combinatorial Optimization, APPROX'98*, pages 135–146. Lecture Notes in Computer Science, vol. 1444, Springer.

Kortsarz, G. and Peleg, D. (1992). Generating sparse 2–spanners. In *Proc. 3rd Scandinavian Workshop on Algorithm Theory, SWAT'92*, pages 301–309.

Kortsarz, G. and Peleg, D. (1994). Generating low-degree 2-spanners. In *Proc. 5th Annual ACM-SIAM Symp. Discrete Algorithms, SODA'94*, pages 556–563.

Kratsch, D., Le, H.-O., Müller, H., Prisner, E., and Wagner, D. (1998). Additive tree spanners. Konstanzer Schriften in Mathematik und Informatik 52, Universität Konstanz, Germany.

Leighton, F. T. (1992). *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufman Publishers.

Leighton, F. T., Maggs, B. M., and Sitaraman, R. K. (1992). On the fault tolerance of some popular bounded-degree networks. Technical Report CS-TR-385-92, Dept. of Computer Science, Princeton University, New Jersey.

Levcopoulos, C., Narasimhan, G., and Smid, M. (1998). Efficient algorithms for constructing fault-tolerant geometric spanners. In *Proc. 30th Annual ACM Symp. Theory of Computing, STOC'98*, pages 186–195.

Liestman, A. L. and Shermer, T. (1991). Additive spanners for hypercubes. *Parallel Processing Letters*, 1:35–42.

Liestman, A. L. and Shermer, T. (1993a). Additive graph spanners. *Networks*, 23:343–364.

Liestman, A. L. and Shermer, T. (1993b). Grid spanners. *Networks*, 23:123–133.

Lipton, R. and Tarjan, R. E. (1979). A separator theorem for planar graphs. *SIAM J. on Applied Mathematics*, 36:177–189.

Madanlal, M., Venkatesan, G., and Pandu Rangan, C. (1997). Tree 3-spanners on interval, permutation and regular bipartite graphs. *Information Processing Letters*, 59:97–102.

Mansfield, A. (1983). Determining the thickness of graphs is $\mathcal{NP}$-hard. *Math. Proc. Cambridge Philosophical Society*, 93:9–23.

Nishizeki, T. and Chiba, N. (1988). *Planar Graphs: Theory and Algorithms*, volume 32 of *Annals of Discrete Mathematics*. North–Holland.

Papadimitriou, C. H. (1994). *Computational Complexity*. Addison–Wesley.

Papoutsakis, I. E. (1999). Two Structure Theorems on Tree Spanners. Master's thesis, Dept. of Computer Science, University of Toronto, Canada.

Peleg, D. and Schaeffer, A. A. (1989). Graph spanners. *J. of Graph Theory*, 13:99–116.

Peleg, D. and Ullman, J. D. (1987). An optimal synchronizer for the hypercube. In *Proc. 6th ACM Symp. Principles of Distributed Computing, Vancouver*, pages 77–85.

Peleg, D. and Upfal, E. (1988). A tradeoff between space and efficiency for routing tables. In *Proc. 20th Annual ACM Symp. Theory of Computing, STOC'88*, pages 43–52.

Richards, D. and Liestman, A. L. (1995). Degree-constrained pyramid spanners. *J. of Parallel and Distributed Computing*, 25:1–6.

Robertson, N. and Seymour, P. (1983). Graph minors I: Excluding a forest. *J. of Combinatorial Theory, Series B*, 35:39–61.

Robertson, N. and Seymour, P. (1986). Graph minors II: Algorithmic aspects of tree-width. *J. on Algorithms*, 7:309–322.

Rosenberg, A. L. (1991). Cycles in networks. Technical Report UM-CS-1991-020, Dept. of Computer and Information Science, Univ. of Massachusetts.

Rotics, U. (1998). *Efficient Algorithms for Generally Intractable Graph Problems Restricted to Specific Classes of Graphs*. PhD thesis, Dept. of Computer Science, Technion, Haifa, Israel.

Rouskas, G. N. and Baldine, I. (1997). Multicast routing with end-to-end delay and delay variation constraints. *IEEE J. Selected Areas in Communications*, 15(3):346–356.

Santoro, N. and Khatib, R. (1985). Labelling and implicit routing in networks. *The Computer Journal*, 28(1):5–8.

Schaefer, T. J. (1978). The complexity of satisfiability problems. In *Proc. 10th Annual ACM Symp. Theory of Computing, STOC'78*, pages 216–226.

Soares, J. (1992). Graph spanners: A survey. *Congressus Numerantium*, 89:225–238.

Sung, T.-Y., Lin, M.-Y., and Ho, T.-Y. (1997). Multiple-edge-fault tolerance with respect to hypercubes. *IEEE Transactions on Parallel and Distributed Systems*, 8(2):187–192.

Sysło, M. M. (1979). Characterization of outerplanar graphs. *Discrete Mathematics*, 26:47–53.

Venkatesan, G., Rotics, U., Madanlal, M., Makowsky, J. A., and Pandu Rangan, C. (1997). Restrictions of minimum spanner problems. *Information and Computation*, 136(2):143–164.

Zehavi, A. and Itai, A. (1989). Three tree-paths. *J. of Graph Theory*, 13:175–188.

Zhu, Q., Parsa, M., and Garcia-Luna-Aceves, J. (1995). A source-based algorithm for delay-constrained minimum-cost multicasting. In *Proc. IEEE Conf. Computer Communications, INFOCOM'95*.

Index